An Embryonics Inspired Architecture for Resilient Decentralised Cloud Service Delivery

Thomas Welsh

A thesis submitted in partial fulfilment of the requirement of Staffordshire University for the degree of Doctor of Philosophy

June 2020

Abstract

Data-driven artificial intelligence applications arising from Internet of Things technologies can have profound wide-reaching societal benefits at the cross-section of the cyber and physical domains. Use-cases are expanding rapidly. For example, smart-homes and smart-buildings provide intelligent monitoring, resource optimisation, safety, and security for their inhabitants. Smart cities can manage transport, waste, energy, and crime on large scales. Whilst smart-manufacturing can autonomously produce goods through the self-management of factories and logistics. As these use-cases expand further, the requirement to ensure data is processed accurately and timely is ever crucial, as many of these applications are safety critical. Where loss off life and economic damage is a likely possibility in the event of system failure. While the typical service delivery paradigm, cloud computing, is strong due to operating upon economies of scale, their physical proximity to these applications creates network latency which is incompatible with these safety critical applications. To complicate matters further, the environments they operate in are becoming increasingly hostile. With resource-constrained and mobile wireless networking, commonplace. These issues drive the need for new service delivery architectures which operate closer to, or even upon, the network devices, sensors and actuators which compose these IoT applications at the network edge. These hostile and resource constrained environments require adaptation of traditional cloud service delivery models to these decentralised mobile and wireless environments. Such architectures need to provide persistent service delivery within the face of a variety of internal and external changes or: resilient decentralised cloud service delivery.

While the current state of the art proposes numerous techniques to enhance the resilience of services in this manner, none provide an architecture which is capable of providing data processing services in a cloud manner which is inherently resilient. Adopting techniques from autonomic computing, whose characteristics are resilient by nature, this thesis presents a biologically-inspired platform modelled on embryonics. Embryonic systems have an ability to self-heal and self-organise whilst showing capacity to support decentralised data processing. An initial model for embryonics-inspired resilient decentralised cloud service delivery is derived according to both the decentralised cloud, and resilience requirements given for this work. Next, this model is simulated using cellular automata, which illustrate the embryonic concept's ability to provide self-healing service delivery under varying system component loss. This highlights optimisation techniques, including: application complexity bounds,

differentiation optimisation, self-healing aggression, and varying system starting conditions. All attributes of which can be adjusted to vary the resilience performance of the system depending upon different resource capabilities and environmental hostilities.

Next, a proof-of-concept implementation is developed and validated which illustrates the efficacy of the solution. This proof-of-concept is evaluated on a larger scale where batches of tests highlighted the different performance criteria and constraints of the system. One key finding was the considerable quantity of redundant messages produced under successful scenarios which were helpful in terms of enabling resilience yet could increase network contention. Therefore balancing these attributes are important according to use-case. Finally, graph-based resilience algorithms were executed across all tests to understand the structural resilience of the system and whether this enabled suitable measurements or prediction of the application's resilience. Interestingly this study highlighted that although the system was not considered to be structurally resilient, the applications were still being executed in the face of many continued component failures. This highlighted that the autonomic embryonic functionality developed was succeeding in executing applications resiliently. Illustrating that structural and application resilience do not necessarily coincide. Additionally, one graph metric, assortativity, was highlighted as being predictive of application resilience, although not structural resilience.

Dedication

This thesis is dedicated to the memory of the first security engineer I knew - Michael Welsh. Your ramblings about security used to annoy me greatly. The irony is certainly not wasted.

Acknowledgements

Firstly I want to thank my family, particularly Nadia and Steve Marsh, whose unwavering support enabled to me write this thesis - and so much more. I would also like to thank my incredible wife Ruta, who provided me with love and support when I needed it the most. I would also like to thank the entire School of Computing at Staffordshire University who not only educated me in a diverse array of subjects but who also motivated me, gave me countless opportunities and provided a stable and inspirational learning environment for over a decade. I would especially like to thank my supervisor Elhadj, along with the rest of the Cloud Computing and Applications research group, for the countless discussions and support during this research.

List of Publications

1. **On Resilience in Cloud Computing : A survey of techniques across the Cloud Domain** - This paper presents a comprehensive review of cloud computing resilience techniques across both traditional centralised and decentralised models. A subset of this work can be found in Chapter 2. This paper was published in *ACM Computing Surveys* Journal May 2020. It supersedes an initially published survey paper: **Perspectives on Resilience in Cloud Computing: Review and Trends** - which was presented at the *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*.

2. **Embryonic Model for Highly Resilient PaaS** was presented at the *2018 Fifth International Conference on Software Defined Systems (SDS)*. It maps embryonic characteristics to cloud computing and resilience requirements in order to derive the initial embryonic cloud model that was developed for centralised cloud features. Some of this work is presented in chapter 3.

3. **Bio-Inspired Multi-agent Embryonic Architecture for Resilient Edge Networks** was published in the *IEEE Transactions on Industrial Informatics* Journal in May 2019. It presents the Cellular Automata based model of the embryonic architecture including simulation results for the majority of results in chapter 4. It presents the concept in the context of real-world use cases for resilient industrial IoT at the network edge.

4. **Embryonic-Inspired Resilient Service Delivery at the Hostile Mobile Edge** - Is currently under review in *Springer's Peer to Peer Networking and Applications* journal. It presents the performance evaluation results of the proof-of-concept embryonic architecture. This work is presented in chapter 6.

Contents

List of Figures

List of Tables

Acronyms

ABE Attribute-Based Encryption.

ACO Ant Colony Optimisation.

AI Artificial Intelligence.

AIS Artificial Immune System.

ALife Artificial Life.

ANN Artificial Neural Networks.

API Application Programming Interface.

CA Cellular Automata.

CD Client Device.

COAP Constrained Application Protocol.

CONST Constrained Devices.

CP-ABE Ciphertext-Policy Attribute-Based Encryption.

CPS Cyber Physical Systems.

CSP Cloud Service Provider.

DC Data Centre.

Div Divisions.

DNA Deoxyribonucleic acid.

DT Disruption Tolerance.

EDGE Edge computing.

FAOT Function Application Traffic Out.

FDPR Function Application Traffic In.

FR Failure Rate.

FSA Finite State Automata.

FT Fault Tolerance.

Func Functions.

GPRS General Packet Radio Service.

GPS Global Positioning System.

GTK Gnome Tool Kit.

GUI Graphical User Interface.

HTTP Hypertext Transport Protocol.

IaaS Infrastructure-as-a-Service.

IIoT Industrial Internet of Things.

IoT Internet of Things.

IP Internet Protocol.

LB Load Balancing.

LongRecv Longest receive time.

M2M Machine to Machine.

MANETS Mobile Area Networks.

MAPE-K Monitor Analyse Plan and Execute according to Stored Knowledge.

MC Multi-Cellular.

MCC Mobile Cloud Computing.

MDC Micro Data Centres.

MEC Mobile Edge Computing.

MoSCoW Must, Should, Could, Won't.

MQTT Mosquito Telemetry Transport.

MTBF Mean Time Between Failures.

MTTR Mean Time To Repair.

N Neighbourhood Size.

NFC Near Field Communication.

OBU On board units.

OCRQ Organisation Capacity Request.

OKA Organisation Keep Alive.

OPAR Organisation Peer Advertisement Request.

Out Output messages.

P2P Peer-to-peer.

PaaS Platform-as-a-Service.

PAPP API Push Application.

PCA Probabilistic Cellular Automata.

PCD Programmed Cell Death.

PFRQ API function load request.

PI Physical Infrastructure.

PN Physical Networking.

Proc Processed messages.

PyGTK Python Gnome Tool Kit.

Q Quantity of Functions.

QoS Quality of Service.

Recv Received messages.

REST Representational State Transfer.

RFID Radio Frequency Identification.

RM Resource Management.

RPL Routing Protocol for Low-Power and Lossy Networks.

RSU Road side units.

SaaS Software-as-a-Service.

SCH Scheduling.

SDN Software Defined Networking.

ShortProc Shortest process time.

SLA Service Level Agreement.

SO Service Orchestration.

SR Spawn Rate.

SRV Survivability.

Subs Subscriptions.

TotalProc Total process time.

TRAN Transportation Layer.

TTL Time to Live.

UML Unified Modelling Language.

VANETS Vehicle Area Networks.

VI Virtual Infrastructure.

VM Virtual Machine.

VMM Virtual Machine Monitor.

VN Virtual Networking.

VNet Virtual Network.

VS Virtual Storage.

WSN Wireless Sensor Networks.

XMPP Extensible Messaging and Presence Protocol.

"The world is either the effect of contrivance or chance; if the latter, it is a world for all that, that is to say, it is a regular and beautiful structure. Now can any man discover symmetry in his own shape, and yet take the universe for a heap of disorder? I say the universe, in which the very discord and confusion of the elements settles into harmony and order."

– Marcus Aurelius

Chapter 1

Introduction

1.1 Problem Definition

The ever increasing societal dependence upon information systems has catalysed the drive towards novel computer service delivery models. Traditionally, users of computer equipment would purchase, maintain and upgrade their own. However, more recently, the growing complexity of computer technologies and systems in tandem with economic and environmental factors have driven the creation of the *cloud-computing* paradigm. The notion of allowing resources to be provisioned from a 3rd party who will own, maintain, and support them (Mell & Grance 2011). Initially a business model to allow owner's of pooled resources to share their spare capacity with the public, cloud-computing rapidly grew into the foremost computing delivery paradigm of choice. Benefits include: increased economy due to lack of software licenses, enhanced staff expertise, hardware costs, security and maintainability from trusted expertise, economical and environmentally friendly resource management, on-demand resource scalability and more (Vu et al. 2020).

Cloud computing's service-oriented perspective focuses upon the Service Level Agreement (SLA) which defines the constraints in which the service must be delivered according to the costs agreed by the Cloud Service Provider (CSP) and the user who provisions them (Mell & Grance 2011). For example an SLA may dictate that a specific application functionality will be available to a specific quantity of users in a geographical location with a specific up time percentage, at a particular speed and latency, whilst supporting certain interfaces. Failure to meet the SLA would be in breach of a business contract and thus not reflect positively upon the CSP (Sun et al. 2019).

Modern applications are complex, composed of multiple layers, operating in a modular manner. An example application may involve the integration of a number of services : e.g. back-end data storage, data processing and analytics (business-logic), data-presentation and web services. Moreover, to accommodate management processes such as software updates and scalability, there has been a trend for cloud service architectures towards microservices. Where large numbers of loosely coupled small

services connect together to deliver a complete service (Gan et al. 2019). Therefore the success of an application's service delivery, from the user's perspective, will require the successful integration of a number of potentially disparate components, each with their own underlying requirements. To accommodate this, at scale, for multiple simultaneously running applications, cloud data-centre management relies upon autonomic resource management and mass resource redundancy to ensure these applications are delivered, particularly in the case of inevitable hardware failures (Colman-Meixner et al. 2016) (Panwar & Supriya 2019) . This ensures that cloud services must be delivered *resiliently*. Where Laprie (2005) describes resilience as "the persistence of service delivery that can justifiably be trusted, when facing changes". While Sterbenz et al. (2010) suggests "the ability of the network to provide and maintain an acceptable level of service in the face of various faults and challenges to normal operation". Within this context, a service could be considered to be executed resiliently if all required sub-components are made available in order to enable the total execution of the service, despite changes in the underlying environment. Therefore, the goal of a resilient service delivery platform would be to maximise the availability of these service sub-components in the presence of perturbations. The variation in this availability between sub-components could be considered as *connectedness*. Where a fully connected application is one where all sub-components can currently communicate with each other, thus enabling the application to successfully execute.

However, not all applications are created equally. Advances in energy efficient microprocessors and internet connectivity, combined with both an industrial, societal and commercial demand for ubiquitous computing devices has driven the creation of new business markets. The novel paradigm: the Internet of Things (IoT) where everyday objects are induced with sensors and actuators to enable their greater observation and influence upon the physical world has birthed the creation of smart-objects (Al-Fuqaha et al. 2015*a*). They may be integrated into buildings (Metallidou et al. 2020) to create smart-homes and smart-factories (Oztemel 2019), or aggregated together to create smart-vehicles. At scale these components can provide management of transport, waste, crime and energy in smart-cities (Ismagilova et al. 2019). While these applications still rely upon the supporting cloud service delivery infrastructure, they have drastically different characteristics from the conventional. The mobile and possibly hostile environments they operate within decree the use of wireless communications. This in turn creates further constraints for energy consumption and communication activity. To complicate matters further, many of these use-cases, such as autonomous vehicles, or manufacturing, require low-latency responses from their back-end application (Bellavista et al. 2019*a*). The addition of safety-critical operation requirements ensures that cloud service delivery infrastructures hosted in remote geographical locations do not offer data processing responses within the latency constraints of these safety critical applications (Desai & Punnekkat 2019). This fundamentally pushes the data processing closer to the end-devices, at the edge of the network or the devices themselves (Bellavista et al. 2019*b*).

Unfortunately the applications which were currently being hosted within resilient data-centres are now found operating on decentralised architectures, distributed across resource constrained devices, potentially operating upon mobile nodes and across wireless links. This has driven an even stronger requirement for resilient cloud service delivery, although now it operates upon hostile decentralised environments. This is also referred to as *decentralised cloud computing* (Ferrer et al. 2019).

Three main techniques for providing resilience to systems are redundancy, diversity, and autonomic management (Maruyama 2013). Additionally, introducing decentralisation minimises any central weak point and should be employed wherever possible. However providing resilience comes at an expense of both resource cost, and code/system complexity. Providing redundancy is the simplest form of resilience, but increased in a linear manner with the additional resource cost. Providing stronger techniques such as diversity come at the cost of complexity and sometimes additional resource cost also. Increasing all of these attributes has negative effects upon already resource constrained environments such as those within IoT and edge networks. Additionally, an increase in resources or software/system complexity also increases the attack surface, creating more places for components to fail or attackers to target.

For an extensive survey of the resilience techniques in cloud computing, the reader is directed to *Appendix B - Paper 1*. In summary, the requirements for providing resilient service delivery are defined below:

- **Decentralisation** is crucial. System centralisation or even hierarchy presents central weak points which may be exploited through subversion or component/system failure. Many systems, such as Peer-to-Peer (P2P) networks remain resilient through reducing or eliminating centralisation. Therefore in order to remain resilient, decentralised cloud services should operate in a decentralised or distributed architecture to minimise this central point of failure.

- **Autonomic management** is an enabling factor for decentralisation, to reduce centralised control. Additionally, self-management enables a system to adapt to a variety of changes and thus makes them inherently resilient. Selected autonomic processes will need to focus on service management and operate with minimal overhead.

- **Redundancy** is a critical component of resilience. Therefore, the ability to provide it should be universal. This means the capacity for service components or resources to scale and integrate as needed so that replicas will always exist to replace those failed or subverted. This will occur without the constraints of the resources available at the hostile IoT/edge networks.

- **Diversity** is another common technique for resilience. While simple characteristics such as diverse hardware may be included by default in decentralised architectures they may be difficult to control. As with redundancy, the focus of diversity should therefore be on the distribution of service components.

Autonomic management of resources is a common technique for both cloud resource and service management in addition to application to a variety of decentralised processes (Vu et al. 2020) They enable system self-management characteristics which are inherently resilient due to their self-adaptive nature. This comes at the expense of additional code complexity, which may pose difficulties for development and maintenance. To mitigate this, developers of autonomic solutions often take inspiration from biology. One such bio-inspired autonomic architecture, embryonics, has been shown to be particularly adept at self-healing (Benkhelifa et al. 2013) (Mange et al. 1998) whilst having an affinity for distributed data processing. This work selects embryonics as a potential architectural solution for providing resilient service delivery in hostile edge network environments.

An investigation will determine its efficacy in providing cloud service functionality in constrained environments. It will also seek to understand practical implications such as suitability for different platforms and the bounds of successful application delivery under different resilience requirements. Finally, optimal parameters according to use-case and methods of measuring the system's state and resilience will be determined.

1.2 Aim, Objectives and Research Methods

The aim of this thesis is to test the hypothesis "A P2P architecture, with characteristics inspired by embryonic development, will provide persistent decentralised cloud service delivery in the face of system perturbations." This aim will be fulfilled through achieving the following objectives and corresponding deliverables:

1. To conduct a literature survey which covers the area of service resilience for decentralised cloud computing. *(Deliverable 1 - Literature Survey)*

2. To model the characteristics of the chosen embryonic development process and develop a P2P service delivery architecture in its likeness. This architecture should be suitable for constrained and hostile decentralised cloud computing environments. *(Deliverable 2 - Embryonic Decentralised Cloud Model)*

3. To implement the proposed architecture and evaluate it's ability to provide persistent service delivery. *(Deliverable 3 - Embryonic Cloud Implementation)*

4. To determine the constraints of the system to understand the suitability of different applications and scenarios. *(Deliverable 4 - A data set and analysis detailing the system constraints)*

5. To investigate different metrics for analysing the embryonic system's state for inferring information about its resilience. *(Deliverable 5 - A data set and a proposal for quantitative measurement of the proposed system)*

6. To evaluate the resilience of the system using the predetermined metrics and employ as a basis for comparison. *(Deliverable 6 - A data set with corresponding analysis of the embryonic system's resilience)*

Figure 1.1 presents an overview of the methodology used to achieve these objectives. Firstly a literature review covers 3 areas: the IoT whose emerging use-cases and applications require resilient service delivery, cloud computing and its novel decentralised paradigms which provide the service delivery, and finally the state-of-the-art in resilience techniques for decentralised cloud computing. This also includes a review of resilience metrics to be employed later for evaluation of the system.

Next, a model for resilient cloud service delivery is derived through a mapping of resilient decentralised cloud requirements and embryonic characteristics. This is then validated qualitatively, including through use-cases. The self-healing characteristics of embryonic development, derived from cellular differentiation and division is then simulated through two complementary means. The first is a probabilistic model (for defining initial constraints) while the second is a Cellular Automata (CA) based simulation which is used to examine the resilience of the model. The results of this analysis inform the later development of characteristics enabling resilience.

Next, software engineering processes utilising Unified Modelling Language (UML) and test-driven development realise the practical implementation of a proof-of-concept of the embryonic cloud service delivery system. This is then validated against its requirements. In the next section, batches of tests are conducted to develop a dataset which can be employed to analyse the service performance characteristics of the proof-of-concept, in addition to an analysis of its resilience during different system states. These two analyses will feed into an overall qualitative and quantitative evaluation of the proof-of-concept embryonic system.

1.3 Scope of the Thesis

This research designs and implements a resilient decentralised cloud service delivery architecture, modelled on the self-healing characteristics of embryonic development.

Traditional cloud computing operates upon data-centres employing vast resources which are resilient by nature and therefore there is a low need for novel resilient service delivery solutions there. Emerging use cases for cloud service delivery operate upon decentralised networks (e.g. fog computing) with low resources situated at the edge of the network or even deployed across the end-devices themselves. The architecture within this work is specifically for deployment in these decentralised cloud environments which will see high-levels of node churn. For example, this could be through continued network link or system failure/subversion.

Resilience techniques can be found employed on any networking or system layer. For example, numerous solutions exist within hostile networking environments to provide routing and networking

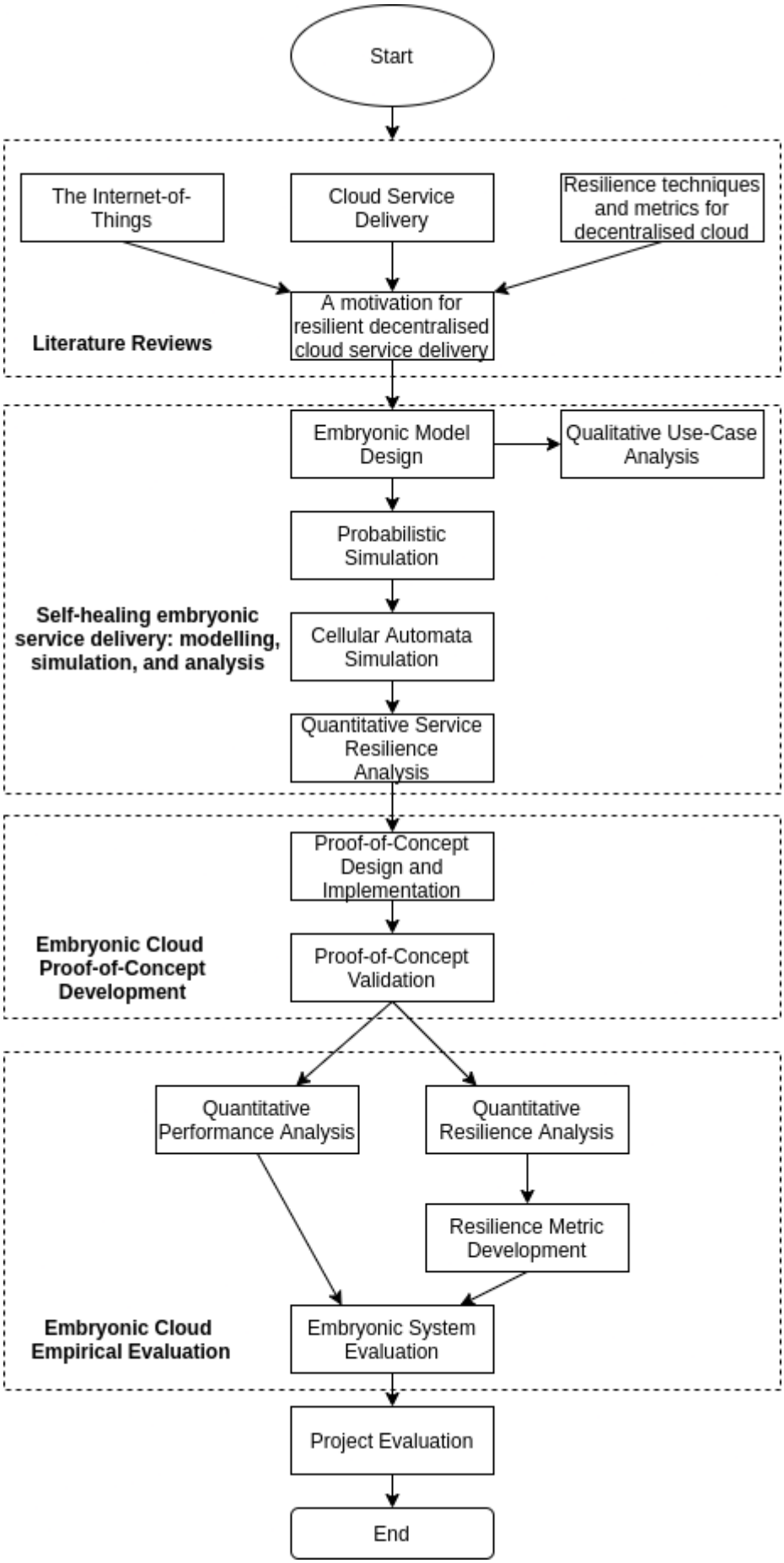


Figure 1.1: Research Methodology Process

protocols which are resilient. However, the focus of the resilience enabling characteristics in this work is purely on the service-layer. I.e. the availability of sub-components of functionality which will collectively contribute to the delivery of a service. A platform which delivers services resiliently should be successful operating upon a variety of lower level networking protocols and system architectures. This can increase the resilience of the system due to diverse underlying architectures although it will also increase the complexity of the system, which should be managed.

In evaluating the resilience of the system, a number of resilience metrics, from literature, are employed to develop a technique for determining the current system's state and use as a basis for comparing with the resilience of other systems. One of the metrics in particular, assortativity, was successfully employed in this manner. However the exact method developed is only for use with the embryonic system and, as yet, cannot be deployed for resilience analysis of other systems. Although the output of this method is easily employed for comparison.

1.4 Resources and Tools

To achieve the objectives, numerous resources and tools are employed. This work relied heavily on custom developed code and therefore the majority of tools employed are 3rd party software libraries. All code was developed and executed on an i5 Linux Laptop with 4GB of ram. [1]

- **Literature - All Thesis** - To manage the organisation and review of literature on the 4 topics of IoT, Cloud Computing and Biologically inspired autonomic methods, a number of tools are employed. Bibtex reference management software tools (Fenn 2006) are used to store, organise and search references of academic papers. Online resources including: Google Scholar, IEEE Explore, ACM Computing Library and Science Direct are used to search for suitable academic work.

- **Modelling and Diagramming - Chapters 1 to 6** - to aid in understanding and representing complex conceptual topics, visual representations were essential. A number of different tools are used. Diagrams.net[2] was employed for the vast majority of diagrams for visual representation of concepts, in addition to UML based software engineering diagrams. Data representation in the form of different graphs were developed predominately using R - the statistical package[3] but also using Libre Office Calc/Microsoft Excel[4].

- **Simulation Tools - Chapter 4** - Two different simulations are conducted, one stochastic based and the other a CA. Both of these simulations are written from scratch using Python [5] while a

[1] All code and datasets developed for this thesis can be found at : https://github.com/tomwelsh/embryonic-cloud
[2] https://apps.diagrams.net - Previously draw.io
[3] https://www.r-project.org/
[4] https://www.libreoffice.org/
[5] https://www.python.org/

Graphical User Interface (GUI) for the Cellular Automata simulation was developed using the Gnome Tool Kit (GTK)[6] with Python Bindings (PyGTK). Data analysis of both simulations is conducted using a combination of R Studio and Libre Office Calc/Microsoft Excel.

- **Prototype Development and Validation - Chapter 5** - The embryonic development prototype is written in Python 3. It relies on the ZeroMQ library which is employed for high performance, brokerless distributed message passing (Hintjens 2013). For system validation, a custom back end logging environment is employed using a custom written ZeroMQ based message sink which stores all of the system and test communication activity in a MYSQL database[7]. This database is accessed through a custom developed RESTFul web service. A custom developed web GUI written using the Python Flask Library[8] then allows presentation and examination of a current, or historically run system for validation and experimentation purposes. The alchemy.js[9] library is employed for visualation of discrete graph structures, while bootstrap.js[10] and jquery.js[11] were employed for presentation and data retrieval.

- **System Analysis and Evaluation - Chapter 6** - Experimentation and analysis of the resilience and performance characteristics of the embyronic architecture used the prototype implementation developed in chapter 5. However to perform experimentation and analysis, Python scripts were employed to execute tests with different parameters. After each test, a timeline of activity is created using custom developed Python scripts using the networkx library[12] for graph analysis, the numpy library[13] for mathematical operations and matplotlib library[14] for images. This tool creates a snapshot everytime the network changes and uses the aforementioned libraries, in addition to custom algorithms, to analyse different resilience and performance criteria. Data analysis was conducted entirely using R statistics package.

1.5 Structure of the Thesis

This thesis is structured as follows: Chapter 2 provides a literature review for this work. Firstly an overview of IoT technologies and their resilience challenges. It then discusses cloud computing and the need for resilient service delivery close to the IoT devices. Next, it presents the state-of-the-art in resilience techniques for decentralised cloud computing. Chapter 3 derives an embryonic decentralised cloud model and evaluates it against resilience and functional service requirements.

[6] https://www.gtk.org/
[7] https://www.mysql.com/
[8] https://flask.palletsprojects.com/en/1.1.x/
[9] https://graphalchemist.github.io/Alchemy//
[10] https://getbootstrap.com/
[11] https://api.jquery.com/
[12] https://networkx.github.io/
[13] https://numpy.org/
[14] https://matplotlib.org/

Chapter 4 simulates the embryonic cloud model using CA to evaluate its efficacy in meeting the resilience objectives. Chapter 5 presents a software engineered proof-of-concept implementation with validation for the embryonic architecture. Chapter 6 presents the results and analysis of a dataset consisting of the performance of the proof-of-concept, which is used to determine a method for its resilience measurement. Finally, chapter 7 concludes the work.

Chapter 2

Literature Review

2.1 Introduction

The literature review presented in this chapter lays the background research for the thesis and highlights the gap in literature and motivation for this work. Firstly the scope of this work is discussed through an overview of IoT technologies which have a high requirement for resilient data processing. Next discussed are the variety of cloud computing models, whose architectures typically delivers the services for IoT. Next, resilience techniques for decentralised cloud computing are discussed for their suitability in enabling the persistent and trusted delivery of services. Finally a discussion of current work within the area of resilient service delivery for decentralised cloud computing highlights gaps in the area of bio-inspired autonomic solutions; deriving a motivation for this work.

2.2 The Internet of Things

The Internet-of-Things (IoT) is a paradigm concerned with building a pervasive environment of smart devices (or things), seeking to enhance everyday life through ubiquitous connectivity (Atzori et al. 2010). This is accomplished via the interconnectivity of sensors and actuators, in order to facilitate smart decisions made via analysis of an inherent wealth of data. The IoT technologies are expected to offer unprecedented opportunities to interconnect human-beings. Additionally, the proposed platform for the future IoT will be through Machine-to-Machine (M2M) communications, whereby sensors and networks allow all 'things' to communicate directly with each other to share vital information. This will allow us to have a truly instrumented universe where accurate data is radially available to inform optimal decision making.

The IoT is typically considered to have partially evolved from the implementation of Radio Frequency Identification Devices (RFID) (Atzori et al. 2010). RFID consists of very low power, wireless tags used to electronically identify physical objects and animals. Whilst allowing the wireless intel-

ligent tracking of objects within confined spaces, RFID tags are passive and unintelligent; features disallowing the ability to log and understand their environment (Kortuem et al. 2010); preventing collaboration with other devices and generally stunting the evolution and further analysis of the inherent wealth of data. With the realisation that the interconnection of these devices, coupled with intelligent data analytics, may enhance services and facilities in the physical world; such devices evolved from being passive objects to interactive, cooperative and smart devices. Although, still retaining the original mantra of low-power and wireless communication, these devices combine sensors with RFID tags to produce wireless devices capable of sensing their environment and thus producing dynamic data. However due to the low power nature of these devices, their range is limited. Therefore, by harnessing the enabling technologies from wireless computing networks, the capabilities to produce wide-scale sensor networks were achieved (Atzori et al. 2010). Also, in order to economise on this sensor usage, it is important to implement these networks in an efficient manner, which is accomplished by applying ad-hoc and distributed networking protocols.

Despite the growing adoption and interest in IoT systems, the term IoT merely describes the idea of global connectivity among smart devices, i.e. it does not specifically define the way in which these devices should communicate. Therefore IoT might best be considered an umbrella term encompassing a variety of technologies and standards, both hardware and software, and does not denote any particular standardisation. IoT networks typically consist of heterogeneous, intercommunicating devices Or "things" and their networks.

2.2.1 Enabling Technologies

IoT networks are (in the majority) driven by and built upon wireless networking specifications. As stated previously, RFID is one of the founding hardware types for IoT devices. Other low-power wireless technologies used include Wireless Sensor Networks (WSNs), Near Field Communication (NFC), Zigbee, 6LoWPAN etc. most of which are considered personal area network technologies due to their low range and bandwidth. Networks may also be constructed upon slightly longer range such as Wi-Fi and Bluetooth (Zanella et al. 2014). In addition, IoT devices may utilize wide area protocols (Ali et al. 2017) such as General Packet Radio Service (GPRS), 3G, 4G, WiMax etc. or bridging with wired protocols to facilitate access to the internet and other external networks (Borgia 2014) Whilst these protocols and technologies are not specifically designed for IoT, their integration and potential use is illustrative of the array of protocols which will require consideration. An extensive survey of these technologies is given by Al-Fuqaha et al. (2015*a*) and Al-Fuqaha et al. (2015*b*).

IoT may be thought of as a 3-layer model. Consisting of the *perception*, *transportation* and *application* stages (Borgia 2014) (Frustaci et al. 2017). Where the perception stage consists of the sensing technologies such as RFID and Global Positioning System (GPS) and short range transmission such as Bluetooth and 802.15.4. The transportation stage consists of longer range communication such

as Internet Protocol (IP), 802.3, 4g, etc. Whilst the final application phase, consists of platforms such as cloud architectures for data management and actuators e.g. traffic management systems.

Due to the resource constraints of the devices, some protocols have been designed specifically to support low power hardware. For example IEEE 802.15.4 is a low power physical and media access specification for resource constrained wireless hardware, Zigbee and 6LoWPAN are both built upon this specification (Gutierrez et al. 2001). With networking protocol packets being mostly too large for constrained resources, 6LoWPAN was developed as a low resource replacement. Specifically designed to connect constrained devices to the internet; 6LoWPAN provides compression in order to accommodate IPv6 over IEEE 802.15.4 or other low power physical and media access protocols. In the literature, 6LoWPAN is often discussed with the Routing Protocol for Low-Power and Lossy Networks (RPL), a multi-functional routing protocol for constrained devices, where both are considered the most common IoT based networking set-ups (Raza et al. 2013).

Within security specifically, this lack of standardisation creates issues when attempting to develop generalised research solutions to determine exactly what must be secured. Therefore, this section described an overview of the characteristics of IoT technologies; including the networking technologies used and the specific device features. IoT based networking stacks may be considered as a typical layered networking stack; with each layer being dependent upon the other.

As IoT based networks may still be quite diverse, it is important to consider all types of IoT protocols. A non-exhaustive overview of protocols and standards which may be seen in current IoT systems are depicted in Table 2.1. For a more comprehensive coverage of IoT enabling protocols the author is directed to the work of Borgia (2014).

2.2.2 Applications

Applications in IoT are commonly known within the home such as smart lighting and smart appliances (Fridges, toasters, etc.). As the technology progresses these applications steadily integrate to become larger smart homes with integrated intelligent security, power, environmental controls etc. (Hassija et al. 2019)

As the need for a globalised access to networks of heterogeneous device types was realised in all facets of society, the IoT was born as a vision of global interconnectivity where embedded devices and sensors facilitate a new age where internet connected devices improve our everyday lives. This is famed to be accomplished via the mass collection and analysis of data, however with this enhanced interconnectivity comes further issues.

Moving out of the home, these technologies integrate with more serious concerns for security and energy management in smart buildings (Hassija et al. 2019). On a wider scale the introduction of IoT into public applications such as transportation, crime, and waste gives rise to smart cities (Gharaibeh et al. 2017), and smart energy (Sun et al. 2015) where the services delivered affect large groups of

Name	**Layer**	**Description**
COAP	Application	Constrained Application Protocol
HTTP	Application	HyperText Transport Protocol
MQTT	Application	MQ Telemetry Transport
XMPP	Application	Extensible Messaging and Presence Protocol
REST	Application	Representational State Transfer
IPV4/6	Network	Internet Protocol 4 / 6
RPL	Network	Routing Protocol for Low power and Lossy Networks
6LoWPAN	Network	IPv6 over Low power Wireless Personal Area Networks
802.15.x	Link / Physical	IEEE Wireless Personal Network Standards
802.11	Link / Physical	IEEE Wireless Local Area Network Standards
802.3	Link / Physical	IEEE Local Area Network Standards
2G/3G/4G/5G	Link / Physical	2nd-5th Generation Mobile Telephony Standards
RFID	Link / Physical	Radio-frequency identification
NFC	Link / Physical	Near Field Communication
WiMax	Link / Physical	Broadband Wireless Metropolitan Area Networks
ZigBee	Link / Physical	High-level Wireless Personal Area Network Standard
GPS	Other	Global Positioning System

Table 2.1: A non-exhaustive list of standards and protocols used in IoT

Networking Layer	Attack Facilitating Features
Physical	External deployment, open wireless medium, embedded design, constrained resources
Link-Layer	Contention based access / collision avoidance
Network	Multi-hop routing, decentralization, broadcast transmissions
Application	Insecure-lower levels, lack of encryption

Table 2.2: Network Layer Insecurities

people. Furthermore, due to the popularity of IoT, many safety-critical applications are now deployed on the IoT technologies described above. The term Industrial IoT (IIoT) refers to the application of IoT technologies within industrial settings. This includes the concept of smart factors, where IIoT technologies autonomously manage machinery and products (Xu et al. 2018). Failures within the system can be extremely costly to a business and therefore they come with enhanced requirements for safety, security, and resilience.

2.2.3 Security and Resilience Challenges in IoT

Security within computer networks has always been a major issue. With sensor based networks being used in a variety of critical infrastructures and applications, the need to secure them is arguably greater than ever (Hassija et al. 2019). With the introduction of data protection laws decreeing the responsible collection and storage of data (Seo et al. 2017); coupled with issues relating to privacy of the individual and above, the secure handling of data contained within IoT based networks is vital to everyone. In addition, digital forensics is becoming an essential tool for the police as well as anyone wishing to protect their own legal interests. Therefore the correct logging of computer network activity is a must. IoT is an emerging technology, famed with being able to change and improve societal life, as such its security is a crucial issue.

Therefore, due to the heavy data collection and processing aspects of IoT, it is particularly prevalent to ensure data security (Availability, Integrity, Confidentiality). Types of attacks on data may be classified as being *passive* or *active* (Modares et al. 2011). While passive attacks are concerned with the theft of data or privacy subversion, active attacks are concerned with the destruction, or subversion of data within the network. Table 2.2 lists the features at each networking layer which have been known to create security related issues within IoT networks.

A number of inherent characteristics of IoT cause security issues to be prevalent and varied from conventional security issues. These mostly stem from the perception layer, due to the constrained nature of these devices. According to Trappe et al. (2015), all of these security issues could be

thought of an extension of device power limitations. Something which conventional security solutions do not suffer from due to their non-mobile nature, allowing them to draw from fixed (and potentially unlimited) energy sources. As a unconstrained energy source is able to support large amounts of memory and processing, Cryptographic principles, which are the foundation of information security, require considerable processing and memory for key storage and processing in order for them to be effective (Trappe et al. 2015).

However, technology and implementation related issues are not the only area which causes IoT devices to be insecure. Profit-driven business and a novel, competitive market causes device manufacturers to consider security as an afterthought, if at all (Frustaci et al. 2017). Due to the predominate sensing nature of the devices, theft of data is considered the largest risk. Unfortunately, the data is often seen to be too trivial for concern. However this tends to be far from the truth e.g. Smart Meters can betray privacy and even physical security breaches through the leaking of data (Asghar et al. 2017). A deeper concern is with smart cities, where data privacy issues may cause "an unequal society" through discrimination (Eckhoff & Wagner 2017).

Whilst data security aspects are an essential consideration, the rise of IoT in safety critical applications means that ensuring that the services that they are part of operate resiliently is vital. The networking features of IoT listed in Table 2.1 all require networking solutions for resilience. However, many of the complex smart services discussed which employ IoT networks have low-latency requirements (Bellavista et al. 2019*a*). Low latency coupled with data security and safety critical applications strengthens this requirement even further. While IoT technologies are adept at sensing and actuating upon data, their inherently constrained nature used to reduce power consumption makes them unsuitable for heavy data processing. The same constraints that reduce their capacity to provide strong security mechanisms reduce their feature set in other means. Additionally, not only do these constraints ensure they lack the ability to provide data processing functionality, they also lack the ability of centralised and unconstrained resources to provide service management, orchestration and scheduling facilities (Al-Fuqaha et al. 2015*b*). Finally, the open wireless mediums and mobile nature of nodes ensures they are susceptible to partial link failure, requiring network protocols to survive non-deterministic and hostile environmental conditions. While these have developed over the years (Paradis & Han 2007) to be resilient in terms of maintaining network communication (Al-Fuqaha et al. 2015*b*), they can be costly in terms of energy usage.

Therefore IoT applications depend upon a supporting back-end in which to store and process data to provide intelligence to their services. The foremost delivery model for IoT is cloud computing (Biswas & Giaffreda 2014) (Al-Fuqaha et al. 2015*b*) and therefore cloud computing provides resilience to the IoT service by providing additional resources in a stable and persistent manner. It also creates another layer in which to provide security and resilience services.

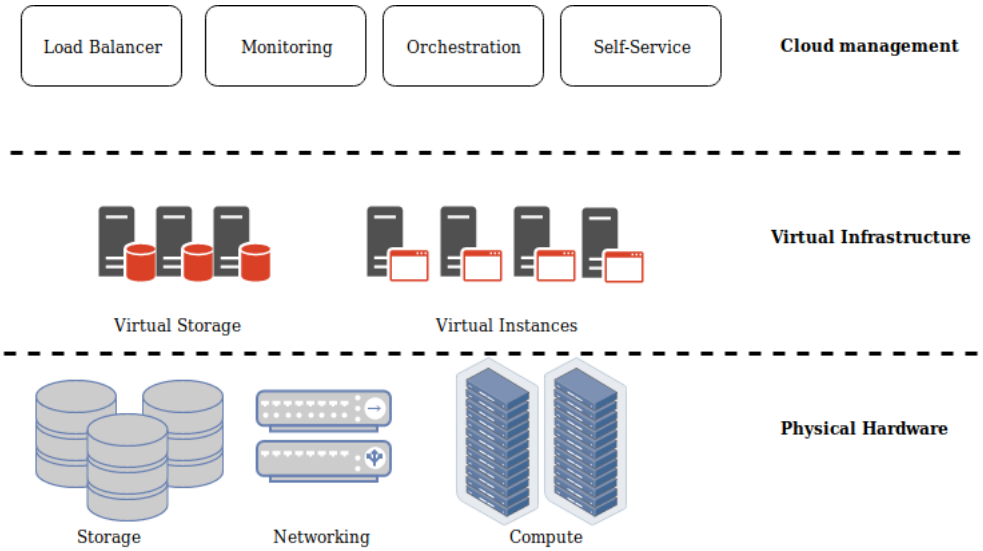


Figure 2.1: Cloud data centre layers

2.3 Cloud Service Delivery

Cloud computing is a service-driven computing model whereby an end-user will provision and use computing resources from a Cloud Service Provider (CSP) in line with an agreed upon Service Level Agreement (SLA). The service hosted by the CSP could take many forms. Consisting of networking, storage or computational components (Mell & Grance 2011). Similar to traditional computing environments, cloud environments are multi-layered. The composition differs depending upon the CSP infrastructure, the application's use-case or the particular model used for analysis.

A typical cloud datacentre (DC) (Figure 2.1) would consist of the underlying physical infrastructure: servers, storage arrays and networking hardware. Virtualised Infrastructure (VI), a pool of resources: virtual machines (VMs) and/or containers running atop of virtual machine monitors (VMM)s with Virtual Storage (VS) devices and Virtual Networks (VNs). These resources are situated upon the Physical Infrastructure (PI) hardware, connected by Physical Networking (PN). A management layer coordinates physical Resource Management (RM) and the service life cycle. Performance is managed through distributing services using Load Balancing (LB). Services are created and managed using Service Orchestration (SO) and executed using Service Scheduling (SCH). Further service-oriented capabilities such as security are also provided.

In the NIST definition for cloud computing (Mell & Grance 2011) the prominent service delivery models are defined as a layered architecture:

- **SaaS** - Software as a Service - the CSP provides the software application only. The end-user will use the application and provide some configuration/customisation.
- **PaaS** - Platform as a Service - the CSP provides a programming interface typically with logical storage, networking etc. The user end-user must program their own application logic.

- **IaaS** - Infrastructure as a Service - the CSP provides only the infrastructure, such as virtual machines, virtual networking etc. The user must configure, install and maintain their own systems.

Within each model, the level of responsibilities (for the management/configuration/security etc.) of the service being delivered will vary between the Cloud Service Provider (CSP) and the end-user who provisions the service. This division of responsibility is an important concept to be aware of within the context of resilient services, as the level of control given to the management and configuration of the service may determine the user's abilities to affect its resilience.

Within the datacentre, these models and roles fit well. However services are frequently being found away from the datacentre. A number of emerging use-cases require data processing to be closer to the end device, at the edge of the network. In tandem with new technologies and use-cases, new forms of cloud computing have therefore developed to accommodate emerging paradigms such as the IoT and big data. These involve distributing the cloud services across devices or network architectures dissimilar to the typical DC only model. Combining these two models gives a distinction between centralised cloud computing, which operates from datacentres, and decentralised cloud computing (Figure 2.2) where cloud services are delivered across more constrained devices closer to, or at, the network edge (Ferrer et al. 2019) (Appendix B Paper 1).

2.3.1 Decentralised Cloud Computing

A variety of different architectural models which push the resources to the network edge are employed in order to reduce latency and enable communication in real-time (Bilal et al. 2018). Despite their differences, these variants all largely attempt to accomplish the same goal. What varies is their use-case, and the underlying technologies in which the new processing occurs (Baktir et al. 2017). They have evolved to provide data processing for end-devices to mitigate latency issues resulting from processing in the DC hosted clouds.

A summary of these emerging architectural cloud models are below:

- **Fog Computing** - seen first as an extension to the cloud but now as complimentary or independent from it. It involves a hierarchy of services where some processing/storage is executed closer to the edge of the network whilst analytics can occur in the cloud. This can occur in small-scale clouds but also on a variety of different hardware such as base stations, routing hardware, etc. (Roman et al. 2018) (Mouradian et al. 2018) (Bilal et al. 2018) (Pan & McElhannon 2018)

- **Mobile Cloud Computing (MCC)** - the concept of resource augmentation from a mobile to a remote device in order to maximise resource efficiency and power consumption. Originally intended for centralised cloud DCs, the potential for processing at the edge and on other mobile devices is now seeing interest (Wang et al. 2017) (Bilal et al. 2018) (Roman et al. 2018).

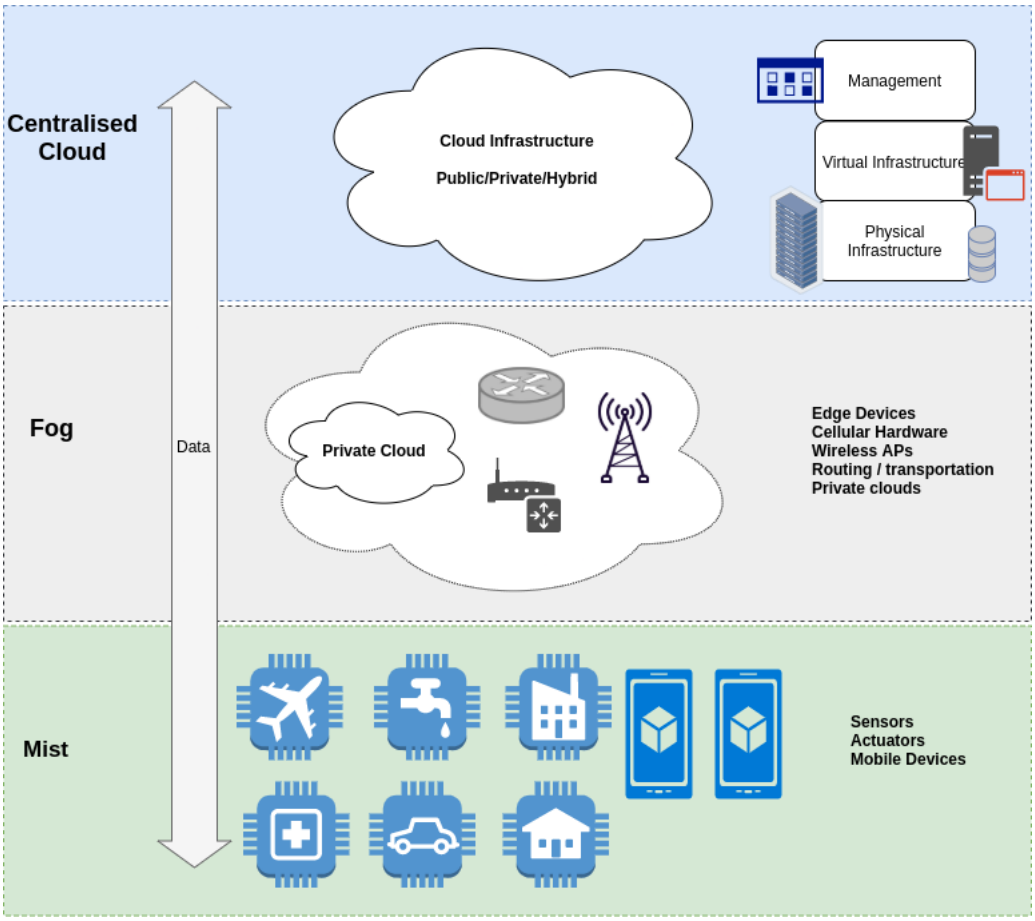


Figure 2.2: Centralised and Decentralised Cloud Layers

- **Cloudlets** - involve the deployment of small clouds, used to reduce short falls in mobile cloud computing (Bilal et al. 2018) (Ai et al. 2018) .
- **Edge Computing EC and Mobile Edge Computing (MEC)** - provides cloud services at the edge of the network such as gateway devices or even the end-user devices. This increases performance through latency reduction, traffic optimisation and enhanced services e.g. location-driven. MEC specifically operates upon cellular networks such as 5G nodes (Roman et al. 2018) (Mach & Becvar 2017) (Mao et al. 2017). (Pan & McElhannon 2018) (Bilal et al. 2018) (Wang et al. 2017) .
- **Mist Computing** - pushes data processing services as far as possible to the sensor and actuator devices (Preden et al. 2015) (Vasconcelos et al. 2018)

These definitions illustrate the variety of related disciplines which have developed to accommodate different service delivery use-cases. While many still maintain characteristics of cloud computing, specifically shared service delivery models, their inherent characteristics such as a resource constrained nature, wireless mediums, and mobile nodes prevent feature rich applications from being processed, although the latency issues have been solved. The goal of these architectural models, in essence, is to

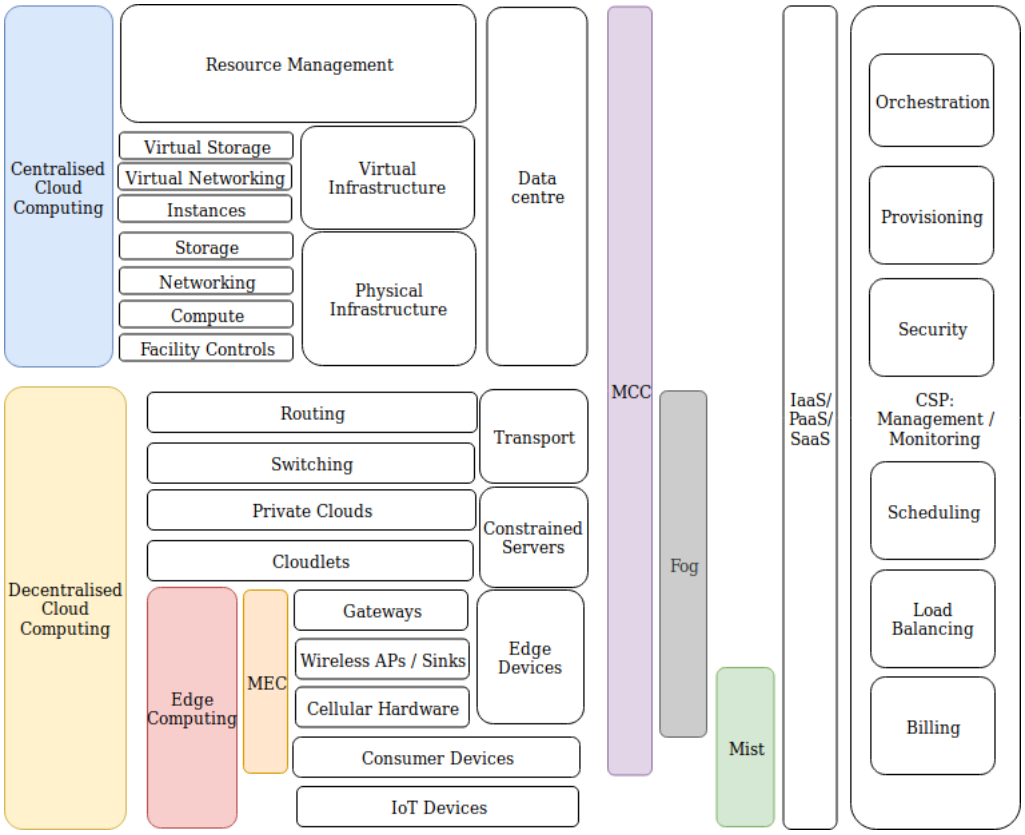


Figure 2.3: Centralised and decentralised cloud computing model. Illustrating the relationship and overlap between different cloud models and architectural components. MEC=Mobile Edge Computing, MCC=Mobile Cloud Computing

permit services to be delivered in environments which are considered to be more resource constrained and non-deterministic than those of cloud services in a DC. To accomplish this, they employ protocols for communication, data storage, and processing which are more robust in terms of failure and latency. Figure 2.3 illustrates the comparison in component use and the way in which emerging cloud models are distributed across the underlying architecture. The constrained resource nature of these devices ensures their feature set is drastically reduced and security issues are greater, when compared to a resource rich centralised cloud. Levering Fog computing nodes as a data processing platform for IoT is discussed in a number of places (Baccarelli et al. 2017) with use-cases such as energy management (Faruque & Vatanparvar 2016), smart-cities (Lyu et al. 2018), they benefit from only simple data pre-processing on the nodes closest to the edge. Additionally, to improve the chance of successful service delivery, particularly in more hostile open environments, these techniques require techniques to further enhance their security and resilience.

2.3.2 Applications for Resilient Decentralised Processing

A number of use-cases are highlighted within the application of Industrial IoT (Aazam et al. 2018). Mining (Singh et al. 2018), transportation, crime (Neto et al. 2018) and agriculture (Heble et al. 2018) are examples where large scale IoT systems may operate in environments with changing and non-deterministic environmental conditions in addition to device mobility.

There are many types of networks which must persist in their service delivery yet their underlying environmental conditions are non-deterministic, resulting in Intermittently Connected Networks. Examples of these include: Mobile Ad-hoc networks, such as with mobile devices or vehicle communications, wireless sensor networks used for environmental monitoring often in extreme conditions and Exotic Media Networks in highly disruptive locations such as space. Most solutions rely on employing delay or disruption tolerant networking approaches such as store and forward, parallel routing, error correction etc. (Khabbaz et al. 2012).

Networking solutions are not compatible with all types of harsh environments. Some use-cases involve the destruction of nodes such as adversarial attacks in warfare (Amin et al. 2015) or node subversion during cyber-attack (Makhdoom et al. 2018). This increase in use-cases integrated with Cyber Physical Systems (CPS) ensures that breaks in service could be catastrophic, with the potential for huge financial loss, or even loss of life. This safety-critical motivation and complexity of security issues drives a focus upon services to be delivered *resiliently*, where the service delivery must persist in the face of internal and external threats.

2.4 Resilient Techniques for Cloud Computing

Due to the constrained nature of IoT devices, data processing, storage and representation must be provided by a third party platform, typically the cloud. However, the high latency, non-deterministic wireless mediums and high volume of data make this relationship difficult. Decentralised cloud computing (e.g. Fog, Edge, Mobile Cloud etc.) is the medium in which cloud services, mostly temporary data processing, are provided closer to the edge of the network and/or distributed across end devices. Processing data in this form has a requirement for resilience due to device mobility, open wireless mediums, constrained device resources, heterogeneous device types, cyber-physical systems and hostile environmental conditions. This resilience may be accomplished through a number of different and complementary techniques.

The Resilinets model (figure 2.4 proposed by Sterbenz et al. (2010)) is often refereed to within resilience literature and helps to classify resilience techniques in terms of their disciplines. The authors provide a comprehensive analysis of resilience, defining it as superset of a number of sub-disciplines with two major sets consisting of challenge tolerance and trustworthiness. The authors explain that some of these disciplines are means and others are quantifications of resilience. Taking this clas-

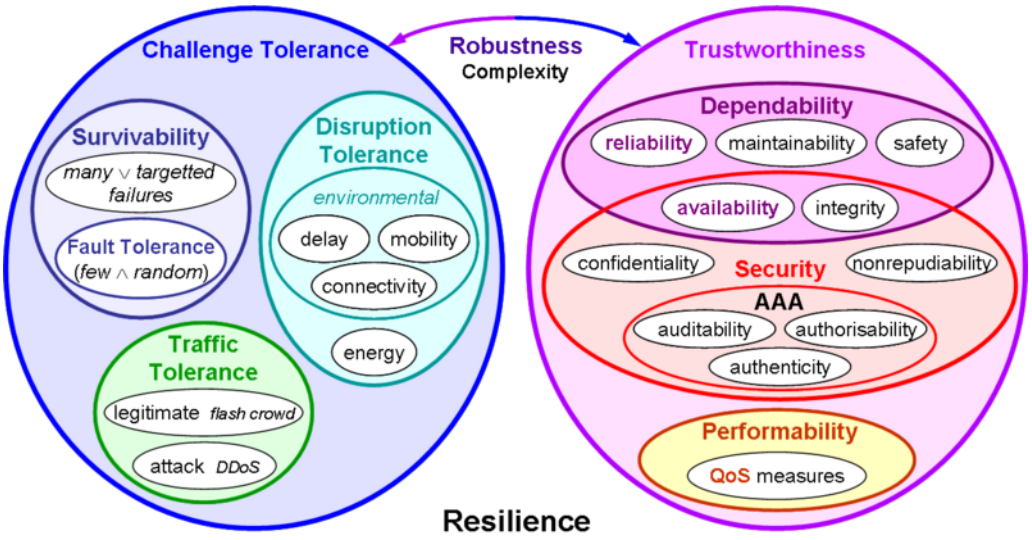


Figure 2.4: The resilinets model of resilience disciplines (Sterbenz et al. 2010)

sification further, figure 2.5 classifies these disciplines into means or quantifications to more easily determine their influence in work to be surveyed.

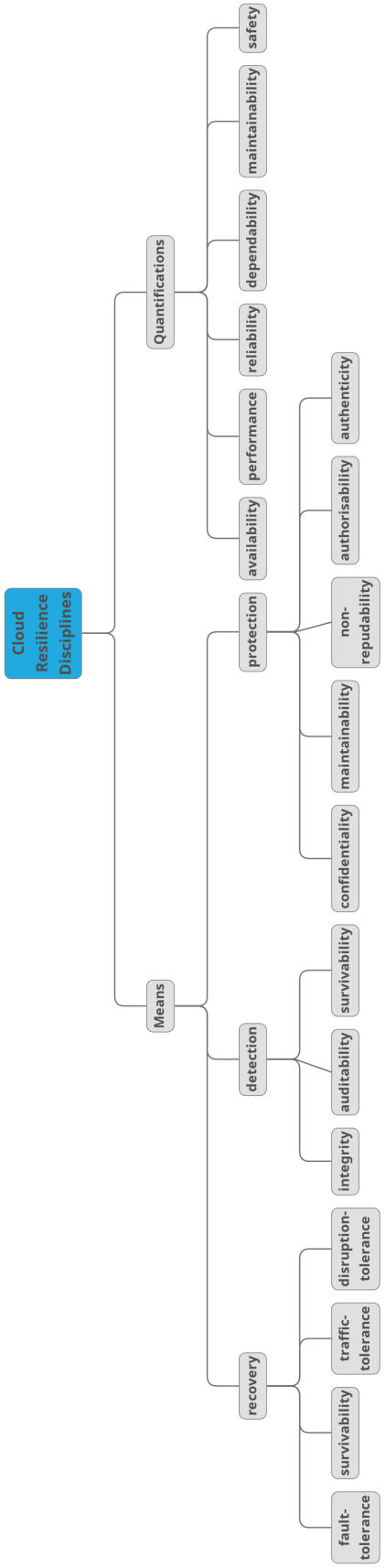


Figure 2.5: Resilience Disciplines

A number of resilience techniques exist for resilient cloud computing. Appendix B - P1 presents a comprehensive survey of these techniques across all cloud layers. It highlights 3 main categories: redundancy, diversity, and autonomic management. A summary is found below:

2.4.1 Redundancy

Providing redundancy is a simple and effective technique for resilience. It works through providing sufficient back up quantities of any resource such that if the resource being used failed, is subverted or is not currently available, the redundant copy of the resource can be employed in its stead. Redundancy can be applied almost anywhere, it can start at the physical layer by having multiple cloud datacentres (Goścień & Walkowiak 2017), or through duplicate network hardware and links (Luo & Liu 2011), or even duplicate storage and servers. Virtual resources are replicated in a similar manner, although given the inherent functionality of virtual infrastructure the resilience techniques are more varied and some optimisation is given. For example, snapshots of instances of virtual machines or containers can be periodically taken and reverted to when necessary (Lombardi et al. 2010)(Jhawar & Piuri 2013). Similarly, virtual storage replication can take advantage of coding techniques to reduce space (Jaiswal et al. 2014). Redundancy techniques are a wholly effective solution to resilience. Unfortunately they come at considerable cost, which typically increases linearly with the quantity of redundant resources. Some virtualisation techniques are developed to mitigate this cost effect to a degree. However, maintaining replicas of instances and storage that are accurate can be challenging, increasing strain on networking resources. Storing long expired replicas is also a concern. Redundancy techniques are simple to implement and are generally suitable for environments such as centralised cloud, which have considerable amounts of spare resources available.

2.4.2 Diversity

Diversity techniques for resilience consider that threats specific for one situation, architecture, software, location, protocol etc. are typically designed only for that circumstance. Having diversity in any characteristic therefore increases the resilience of the system. On the lowest levels, diversity of hardware (both physical and virtual) prevents issues related to code bugs, exploits, or hardware component defects from affecting the entire system. For example, a flaw affecting one vendor's network switch will unlikely affect a different vendor. Similarly software bugs affecting x86 architecture are unlikely to affect x64 Similarly, this concept can be employed at the software layer. Execution diversity techniques can compile multiple versions of the same software, with identical functionality, yet with different underlying code logic (Kanter & Taylor 2013). A service might use multiple different types of protocols in tandem to accomplish the same functionality, these could be largely different or simply different versions. A service itself might use multiple clients, for example a web service might run Apache and IIS simultaneously (Guo & Bhattacharya 2014). Finally location diversity is

a commonly employed technique, for example distributing cloud services across multiple geographical locations (Souza Couto et al. 2014) or network paths (Secci & Murugesan 2014).

Similarly to the cost of deploying redundancy, diversity costs are high. Although this is dependent upon the technique employed, if the diversified resource is simply replacing a resource then the major cost is the complexity of the solution. This includes the level of work for pre-deployment, such as developing a new technique to diversify software. There is additional software complexity required to ensure disparate technologies interoperate. Sometimes this can be through the multiple layers of abstraction inherent to computing, such as differences in hardware being less relevant if the service is developed in a high-level cross-platform language. A caveat of this approach is the increased code-complexity for interoperability and pre-deployment can create a larger attack surface, therefore this should be minimised as much as possible.

2.4.3 Autonomic

Autonomic computing was initially proposed by IBM (Ganek & Corbi 2003), and describes a visions in which computer systems can dynamically manage themselves through self-adaptation. It is inspired by the biological nervous system's ability to manage low-level vital functions of a biological being. An autonomic computing system is described as having a number of self-* properties. At the highest level self-managing, but also self-healing, self-protection, self-optimisation and self-configuring. These autonomic systems employ a control loop. The most common activities are to Monitor, Analyse, Plan and Execute according to stored Knowledge (MAPE-K) actions in order to adjust their processes according to the current situation and environment (Rutten et al. 2017). Self-* properties create resilient systems by nature, self-healing and self-repair are some obvious attributes. Whilst the fundamental ability to adapt to a variety of internal and external perturbations using self-management permits autonomic systems to maintain their operation in the face of changes. Cloud computing itself is inherently autonomic, particularly for service-level resource management (Faniyi & Bahsoon 2015). Management processes within the cloud such as service scheduling and orchestration, and resource provisioning are all good target for resilience optimisation. Additionally, networking at both the networking and service layers are strong targets to accommodate variable performance. Autonomic techniques tend to come at a cost of high computational complexity, which can be distributed across many nodes or conducted on a single node. The heavy reliance upon these processes requires them to operate with a high degree of accuracy and dependability. Therefore they will likely require a high cost of development pre-deployment. For these reasons, these techniques are often developed in the likeness of their original motivation, biology. Many differenet aspects can be seen in literature such as animal cell-based models (Hariri et al. 2011), networking approaches based upon the distributed processing capabilities of ant colonies (Baran & Sosa 2000), intrusion detection methods (Martí & Schoenauer 2018) or service orchestration methods (Ha 2018) based upon artificial immune systems,

alternatives to public key infrastructure based upon family genetics (Wang et al. 2010) and many more.

2.4.4 Comparison

Figure 2.6 classifies the previously discussed resilience techniques for coud computing. Table 2.3 compares the cost and complexity of each of the cloud techniques in terms of their pre-deployment, deployment, maintainability and code complexity. Redundancy has a low pre-deployment cost as it requires minimum effort to integrate redundant resources. However the deployment and maintainability costs are both high, as needing to replace the resources is costly. The overall complexity is low due to homogeneous resources. Diversity has a medium to high pre-deployment cost, as this depends on the particular technique but will almost always require the interoperability of diverse technologies. Its deployment is medium, as it often requires less cost than a complete replica, but there is additional code to employ, likewise with the maintainability. The complexity of the solution is also medium, due to the additional code required to link diverse resources yet this tends to be minimal. Autonomic techniques have a high pre-deployment cost, largely due to the issues associated with their development. However their deployment cost is low as the code is capable of self managing itself. Maintainability is medium as it may be difficulty to change the code while it is operating. Finally the code complexity is ranked as minimum, as while autonomic techniques tend to use simplistic control loops, their implementation may cause them to be deployed in a distributed manner.

	Description	**Pre-Deployment**	**Deployment**	**Maintainability**	**Complexity**
Redundancy	Replicated resources	Low	High	High	Low
Diversity	Heterogeneous configurations	Medium-High	Medium	Medium	Medium
Autonomic	Adaptive self-management	High	Low	Medium	Medium

Table 2.3: Cloud Resilience Technique Cost Comparison

While these techniques are easily employed in the centralised cloud, the resource constrained nature of the decentralised cloud makes them non-trivial. Therefore techniques to enable resilience in decentralised cloud environments focus on novel applications number of different cloud service related attributes.

2.4.5 Survey of Resilience Techniques for Decentralised Cloud Computing

Following from the classification of resilience techniques, this section surveys a number of different works which attempt to enhance service resilience in the decentralised cloud.

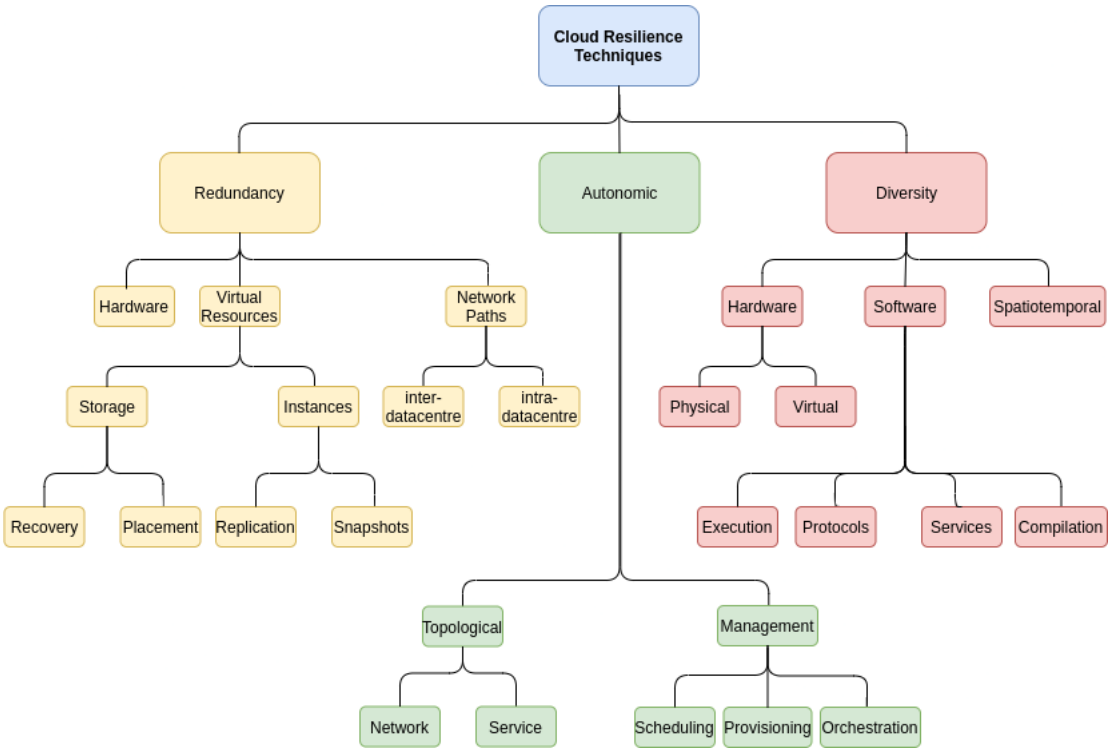


Figure 2.6: Classification of techniques used in cloud resilience

Service orchestration (SO) is an important process to conduct resiliently in fog computing. In order to optimise constrained device resources, only the minimum amount of nodes necessary will be provisioned for an end-user. This requires service requirements to be broadcast for a network which provides a number of security issues, particularly confidentiality. Viejo & Sánchez (2019) use Ciphertext-Policy Attribute-Based Encryption (CP-ABE), whereby nodes will have keys corresponding only to the attributes they are allowed to process. Their network is structured hierarchically so that nodes pass messages to those they can control further down the tree. The nodes will require a generalised key. For example, a message containing "temperature" will also need a "weather" key to process it. These messages form policies such as "temperature, zone 1" which are then encrypted separately and transmitted. If any messages can be decrypted by a node it means that further nodes in the hierarchy can also be decrypted so the service discovery can continue. Once the service has been orchestrated between the required nodes, the client and nodes exchange keys to communicate securely. The cryptography heavy nature of this solution is questionable in terms of energy cost. Chejerla & Madria (2017) instead chose to develop a scheduling algorithm that uses a game-theoretic and Bayesian approach to mitigates against attack in real time, for Cyber Physical Systems (CPS). Scheduling is a known key focus of cloud resilience although assuming the rationalisation of an attacker such that they can be modelled is contentious.

Rios et al. (2017) explain that modelling fog networks in a hierarchical manner, with a singular provider, is oversimplified and detrimental to its security. They should instead be considered as a

federated architecture with numerous service providers within different trust domains. The authors propose an architecture (SMOG) to provide resilience in fog networks. It consists of number of baseline characteristics such as *secure interconnection*, *authentication and authorisation*, *protection of virtualised environments* and *situational awareness*. They list enhanced characteristics as *trust services*, *distributed decision making*, *privacy capabilities* and *digital evidence management*. They explain that these base line requirements are largely missing from literature and are necessary to ensure a secure and resilience fog. This work highlights that for some scenarios, resilience requirements can require multifaceted and complex solutions. Highlighting their inherent inadequacy.

The edge nodes are without doubt a point of failure in any decentralised cloud network. Le et al. (2017) give a solution to partial failures in MEC (e.g. connectivity loss) between the edge nodes. Their architecture is again hierarchical, with mobile nodes storing local back up data dispersed amongst them. If partial failure with the edge nodes occurs, the devices switch to a P2P model, processing data collaboratively. This is an alternative mobile computing model and the results show good time reduction performance when the task is disrupted across the nodes. However the power consumption is likely to be highly variable according to the difference between nodes and therefore the suitability will not be universal.

Modarresi & Sterbenz (2017) consider Fog Computing as a solution for resilient IoT/edge computing in. They argue that the uncertainty surrounding resource, link and bandwidth availability ensures that typical edge computing is not resilient for IoT processing. For example, too many clients can overload resources and thus cause a denial-of-service. They argue that the introduction of fog nodes between the edge and the cloud creates greater autonomy within the network. If a connection is lost between the edge and the cloud, the fog maintains this network and increases the *survivability* of the ecosystem. Further to this, they suggest that the diversity of standards, protocols and network links, which cause fog computing to be quite complex, is actually beneficial to its resilience due to the increase in variety. They also indicate that through fog reducing traffic further in the core and distribution network, its implementation provides *traffic tolerance*. Finally, *disruption tolerance* is enhanced through a reduction in latency permitting applications to be processed quicker and thus any disruption has less impact. The authors support these statements with numerous simulations inclusive of the fog environment.

Hussein et al. (2017) provide a mobile edge computing solution which applies Software Defined Networking(SDN) to 5G provide resilient processing to Vehicle Area Networks (VANETS). Safety concerns are paramount in vehicles and as such so is the resilience of VANETs. Their proposed solutions provides enhanced security through an additional security layer using SDN. As opposed to a traditional centralised SDN approach or a traditional distributed VANET approach, they present a hybrid method. A centralised 5G base station is used to manage SDN security functions distributed across a number of roadside controllers. This approach illustrates a strong example of custom networking

hierarchy technologies being supported at the edge for specific use-cases and resilience requirements.

Modarresi et al. (2017) deploys SDN again in tandem with fog for resilience. This time fog nodes are used to detect anomalies in network traffic and notify the SDN controller. This can make security-focused decisions about what traffic to drop or restrict. A strongly illustrative example of the application of fog for greater network resilience although does not help to strengthen resilience of the fog nodes themselves.

Benson et al. (2018) takes a middleware approach to provide continued operation of critical events from IoT devices when their connection to the cloud fails . Their system contains two components, the first periodically probes different paths to the cloud, detecting possible faults or failures. The second provides multicast message dissemination according to information received from the first component. They again use SDN to provide this information and use it create "resilient overlays". This middleware approach enables varied support for IoT devices as the middleware works seamlessly.

Kahla et al. (2018) provide a solution to low trust in IoT environments. They leverage moving target defence to migrate targeted or subverted virtual instances to another host fog machine. It is not clear how this would prevent a number of different attacks or heal the instance once it had migrated although the autonomic aspect of integrity verification is commendable.

Eisele et al. (2017) state that resilience is necessary to consider in edge environments due to both resource and network uncertainty. Whilst security is important due to the resource constrained nature of edge devices preventing virtualisation providing adequate isolation. They propose a novel programming paradigm: RIAPS (Resilient Information Architecture Platform for Smart Grid) which provides a platform for distributed applications to be deployed resiliently. The platform provides a diverse number of different services and managers (such as for security, persistence, fault management etc.). Whilst the platform appears to be complex and thus has an increased attack surface, given the number of required components, it illustrates the notion of an underlying platform providing resilience to higher levels.

Aral & Brandic (2018) use bayesian belief networks to mine dependencies between replicated edge nodes. Their solution uses past server performance from logs and temporal dependencies to highlight the probability of when failures may occur concurrently. Although their current solution is theoretical it shows strong optimisation through replica reduction.

Araujo Neto et al. (2018) tackles a somewhat different resilience problem . Where a Fog enabled service does not suffer a fault but an outage related to the CSP's SLA. They focus on Amazon's Spot Instances which are transient servers acquired by the user when the maximum they wish to pay (bid) is greater than the value of the instance. Due to the nature of this acquisition the continued operation of these servers cannot be guaranteed. Therefore in this fog platform the failure results from the unavailable CSP back-end. To mitigate this, they propose an agent-based case-based-reasoning solution which aims to predict the survival time of an instance. This enables checkpoints to be made in

order to resume the work in case of application fault. Their solution could be modified for application processing closer to the edge, although the resource requirements for checkpoints must be considered.

Ozeer et al. (2018) have a similar focus on recording and reverting to application states. They take an uncoordinated approach, recording application events with a corresponding recovery timer. Expiration indicates lack of synchronisation with the physical world and can't thus be ignored. Event details are logged in a global and failure-free storage system to permit recovery to any node from a central location. This centralised storage suffers from central point of failure. The authors present a competent yet complex solution consisting to enable system fault tolerance. The question of how failures are to be handled in the system handling the failures is still open.

Khalifa et al. (2014) move away from a traditional cloud architecture, improving the resilience of Hybrid Mobile Clouds. Mobile clouds require a static system due to the dynamic network characteristics. The proposed architecture is interesting due to its flexibility in running on diverse devices, essentially ignoring the underlying hardware. The resilience requirements are aided through a resource prediction mechanism and an early failure detection mechanism to facilitate handover of vital services. The system proves successful, although performance is still dependent upon the quantity of fixed nodes within the cloud, making the system not purely mobile. However, overall it exhibits a good example of how cloud systems can be built upon non-deterministic environments.

2.4.5.1 Analysis of Decentralised Cloud Resilience

The requirement for resilience at the decentralised cloud layer is greater than the centralised due to data-centre hardware being resilient by nature. Such constrained environments have less ability to fall back upon redundancy and cryptographic methods in order to provide their resilience and generally operate in a hostile environment. Literature on techniques for decentralised cloud service resilience are spread across a number of disciplines. Survivability and fault tolerance are present but a number of security disciplines can be seen in addition such as disruption and traffic tolerance. These techniques often focus on networking such as routing or middleware, which is intuitive given the hostile networking environment. These are the strongest resilience techniques given the history of wireless sensor networks' focus on resilient networking. However while they enable network routing and communication, they do not provide service support. The foundation of IoT networks (WSNs and MANETS) have seen bodies of literature (Zhou et al. 2008) (Benkhelifa et al. 2018) attempting to optimise resilient communication, security etc. It could therefore be argued that the entire focus of these disciplines is in delivering a resilient platform given the hostile environment in which they operate. However the decentralised cloud disciplines defined previously consist of more than simply IoT networks. There is now an entire ecosystem where continuously evolving use-cases demand rich data processing at any and all layers from the IoT device, to the transportation/Fog layer back to the centralised cloud. These networks are heterogeneous and non-deterministic which further

complicate matters. Traffic will traverse multiple governance domains, operate on a diverse plethora of hardware/software configurations and requirements for performance and resilience will change in fractions of a second according to external and internal requirements.

The data-driven nature of these environments, coupled with the inherently low security drives data security methods. Techniques such as CP-ABE and anomaly intrusion detection are low-resource alternatives to traditional security solutions, designed to operate in hostile environments. Unfortunately they are also resource intensive and costly in terms of computation, storage and therefore energy. Redundancy techniques are still prevalent despite the lack of resources through the use of instance checkpointing, although the efficacy of this is questionable. While diversity related techniques are deployed in a minor way due to disparate hardware involved. Finally, decentralised autonomic management techniques are shown to be effective and are suited to the decentralised environment. However their complexity is in question. Additionally, none of the works surveyed accomplish the effect of providing all resilient service management features.

Some key challenges to the future or resilient cloud computing are highlighted below:

1. **Use-case Diversity** - While cloud environments are inherently employed to provide resources for diverse use-cases, resilience techniques tend to be developed for specific use-cases. This highlighted the need for cloud environments to provide *adaptive resilience* according to the need. Integrating a plethora of techniques and selecting the most appropriate is thus an ongoing challenge for the current and emerging cloud.

2. **Uncertain and dynamic governance and responsibility** - Traditional cloud delivery models (SaaS/PaaS/IaaS) define clear responsibility boundaries between the CSP and the user. They can assist in determining which actor can affect resilience at which layer. However, in decentralised cloud disciplines, particularly those with node mobility (e.g. MEC and fog computing) these actors can dynamically change according to physical boundaries and network requirements. Ensuring the capacity to both understand and monitor who has responsibility for resilience extemporaneously is crucial to providing resilience in decentralised clouds.

3. **Evolving cloud paradigms** - Summarising a key concern during this survey is the manner in which cloud computing, as a concept, is continuously in flux. Driven by both changing use-cases and continuous strives for optimisation, the deployment of new and emerging cloud paradigms poses a challenge to service resilience. Where the resilience of new techniques should be considered during their development and not post-deployment.

2.4.6 Resilience Metrics and Evaluation

As with the resilience disciplines, measurement of cloud resilience could follow traditional performance-based resilience metrics such as Mean Time Between Failures (MTBF) and Mean Time Taken to Repair

(MTTR) and the corresponding availability which is easily calculated from the two. However these metrics could be considered primitive at best (Colman-Meixner et al. 2016) considering the complexity of these environments. The resilinets model (Sterbenz et al. 2010) provides a method of determining which resilience features are available through binary selection of distinct features (e.g. the network provides confidentiality or it does not). Other non-cloud specific resilience metrics also suffice, such as graph metrics. Graph metrics are noted for their ease of comparing distinct architectures as they examine the structural characteristics of a network. Alenazi and Sterbenz evaluate a number of graph metrics for resilience are (Alenazi & Sterbenz 2015*a*) and (Alenazi & Sterbenz 2015*b*). These include elementary metrics such as the quantity of nodes, node connectivity (the average number of connected nodes to each other node), node centrality (the most important nodes) etc. They also include those specifically for network resilience through removal of links and nodes e.g. network criticality and effective graph resistance. All of the above are arguably strongest when examining a distinct service as opposed to the entire cloud environment. Some of the works surveyed according to layer perform some simulation of a model in order to evaluate resilience within the context of that particular use-case, the following works focus upon more generalised models for resilience. An example of some of these metrics found within literature which have been used for measuring resilience in networks to varying degrees can be found in table 2.4

Name	**Description**
N	The total number of nodes
Total Path Diversity	Links between alternative paths between two communicating nodes
Average Node Connectivity	The average number of nodes each node is connected to
Weighted Spectrum Distribution	Permits comparison between graph through ranking feature importance
Average Shortest Path Length	The average shortest path between two nodes
Eccentricity	The longest of the shortest path length between two nodes
Betweenness	The number of shortest paths through a node or link
Clustering Coefficient	The connectedness of a node's neighbours
Node Degree centrality	Indicates how important a node is. i.e. the network will be disrupted if removed
Assortivity	Describes node similarity
Network Criticality	Used to measure the resilience of a network against structural changes
Effective Graph Resistance	Used to measure the resilience of a network against structural changes.

Table 2.4: Some example graph metrics seen in literature for measuring resilience of different network types

Jabbar (2010) states that resilience is more difficult to measure than traditional security metrics due to the need to evaluate how effectively the service is still being delivered. They propose that

resilience should be measured as a state space considered in terms of degradation. Where a service is more resilience if it contains more states in which it stays operational and not severely degraded. Such a high-level approach may be applicable to complex environments.

Ghosh et al. (2010) provides a model for resiliency based on stochastic reward nets. The work is interesting in that the metrics for resiliency focus upon evaluating how effectively the job is scheduled through Quality of Service (QoS) metrics. Those given are the rate of rejected jobs, and the delay in VM provision. Following from the definition of resiliency: "quantification of service delivery during changes", the authors evaluate changes as fluctuations in job arrival rate and the quantity of physical machines. Their results showed a faster provision rate was more resilient. Also that removal of a hot physical machine has an adverse effect upon resiliency, whereas removal of a cold one has a minimal effect.

Ju et al. (2013) evaluate the resilience of OpenStack. They develop a novel fault injection framework for both the architecture and its services. They uncovered 23 different bugs which developed into faults in the system. Highlighting the lack of effective resilience considerations within the stock cloud management software.

Tu & Xu (2013) present a resilience model built for a typical IaaS cloud, using Eucalyptus. They explain that resilience and robustness are strongly connected in complex systems, where both properties describe the system's ability to react to disturbances but vary in how they do so. Considering the cloud as a multi-component, hierarchical system, the model evaluates the component interaction and interdependency upon resource consumption. Resilience is modelled by the strength of interactions between the components, where the strength is the percentage needed to consume from another component. A disturbance within the system results in a large queue, exhausting resources causing the services to fail. The authors describe system wide resilience as the quantity of processes which fail due to the inability to consume. They note that this system does not take into account factors which may influence the interaction strength such as one to many and many to one resource consumption interactions. It is mentioned that resiliency is accomplished through redundancy, which has an adverse affect upon cost. To fit in line with the author's model, they explain that increased redundancy, weakens the requirements for resource consumption links between individual components. Whilst redundancy is a key component of resiliency, it is not the only method, and poor implementations can even reduce resiliency under certain circumstances. The authors then attempt to understand more about this effect, examining of replication algorithms with modularisation of a cloud system. Their results show that as size increases, modulation is more important to prevent duplicate replication updates. However they also mention that poor modularisation implementation can create a single point of failure and thus become an enabling factor for poor resilience.

Scholler et. al. present an architectural model which enables insight into the security implications of cloud architectures (Scholler et al. 2013) (Hecht et al. 2014). Their motivation is that current

cloud services do not accommodate security and resiliency for critical infrastructures. Their model distinguishes between the different roles, (such as the physical provider, service developer and service user) as well as the different infrastructures (the physical and virtual) to assess the given requirements against the system. It promotes greater logging for audit purposes, as well as increased transparency between the physical and virtual layers, in order to increase trust between the users. Arguably, many issues within current cloud architectures ensure their unsuitability for a wide range of critical infrastructure services.

Sousa et al. (2014*b*) conduct an evaluation of Quality of Resilience evaluation criteria within the cloud, in order to activate appropriate proactive resilience measures. They propose to use multiple criteria to evaluate the resilience, partly due to the wide variety of requirements associated with resilience and also because many proactive mechanisms require further information. The authors implement proactive resilience systems using multiple criteria for the cloud, MeTH (Sousa et al. 2014*a*) and TOPSIS (Tran & Boukhatem 2008). The results showed that both methods improved the resilience of protocols which were unable to detect cloud layer faults but MeTH provided the greater performance in both fault and non-fault scenarios.

A classification of types of resilience metrics found within cloud computing are described below:

- **Binary feature** based metrics are those relating to the resilinets model such as confidentiality which either exist in the service of cloud or do not.
- **State-based** are those which examine the degradation of service to determine when resilience has failed.
- **Performance-oriented** metrics are the traditional type such as MTTR or QoS which typically involve examining one distinct service.
- **Graph-based** metrics examine issues in topologies such as network criticality.
- **Multi-criteria** metrics aggregate and summarise a variety of metrics into one to take into account very complex systems.

2.5 Resilience for Decentralised Cloud Services

It was highlighted in the previous sections that there is a wider uptake of IoT enabled services which are more safety critical in nature and whose failure or gaps in service delivery could be costly to life or economy. Additionally, the emergence of many applications requiring lower latency has caused data processing to be pushed to the edge of the network or to the end-devices themselves. Traditionally data-centre hosted cloud services are now required to operate in a decentralised manner yet simultaneously in more hostile environments.

This drives the need for enhanced resilience for decentralised cloud services. This is particularly pertinent to those use-cases with strongly constrained resources and the mobile, wireless and potentially hostile environments cause failure between computing network entities. Enabling resilience will then also permit feature rich data processing within non-deterministic environments, and typically over mobile wireless links. Section 2.4.5 analysed a number of these techniques and highlighted that predominately they provide resilience through costly redundancy (Le et al. 2017), security techniques (Viejo & Sánchez 2019) (Kahla et al. 2018) or rely upon centralised or hierarchical management (Viejo & Sánchez 2019) (Le et al. 2017), which present single points of failure. Most importantly, no techniques currently exist which permit data processing to exist distributed across the resource constrained nodes, in a resilient manner while also providing: redundancy, diversity and standard service management techniques. Some techniques proposed novel architectures for resilient service delivery, but are generally complex and require additional servers (Modarresi & Sterbenz 2017) being less suitable for the constrained environments.

2.5.1 The Requirement for Autonomic Service Management

Lacking in particular are autonomic techniques specifically focused on service delivery, which have seen much success within centralised cloud computing (Panwar & Supriya 2019). They are also found with good success in the lower level networking protocols in IoT such as for self-organising and self-healing routing protocols (Dressler & Akan 2010) (Zheng & Sicker 2013). Autonomic techniques strongly fit the requirement for providing resilient decentralised cloud services for the following reasons:

- **Decentralised nature** - Autonomic techniques have been shown to be adept at self-managing environments void of centralised control (e.g. with wireless sensor networks (Marsh et al. 2004)). This is suitable firstly because decentralisation is a strong requirement for resilience and many of the current techniques reviewed are centralised or hierarchical. Secondly, due to the hostile network environments operating themselves in a largely decentralised manner.

- **Low resource cost** - In comparison to redundancy and diversity techniques, autonomic management has a low resource cost per each device. After a medium-high resource cost to develop the technique, overall the cost is low. This is accomplished through distributing computational complexity across the network and using relatively simple control loops. This is suitable for nodes at the network edge (or distributed across the end-devices) due to their lower resource capacity.

- **Inherent Resilience** - autonomic techniques create resilient systems by nature of their ability to adapt to changes in their environment. Many of the techniques for decentralised cloud resilience discussed are solutions adapted for resilience. The ability to self-heal, self-repair, and self-organise according to environment perturbations is inherent to autonomic systems.

- **Compatibility with Cloud Service Management** - the cloud service management functions (discussed in section 2.3) are typically employed in an autonomic manner in order to allow resources to be provisioned autonomously, provide resource optimisation and be able to react to service and environment changes.

Therefore by building service delivery platforms at the edge which are autonomic by nature, service delivery will also be resilient by nature. Many autonomic techniques have been shown in different areas of computing. As mentioned in section 2.4.3, autonomic computing was inspired by biological autonomic systems. Developing bio-inspired systems in this manner mitigates the issue of having a high pre-development cost, due to the effectiveness of a bio-inspired technique being already proven.

Artificial Neural Networks (ANNs) are arguably one of the most commonly known bio-inspired techniques. ANNs are modelled upon biological neural networks, which are the fundamental structure and function of the animal brain. The programs in use today are founded upon the propositional logic model proposed by McCulloch & Pitts (1943). This model simplifies the neuron (the atomic structure of a neural network) as a threshold function which "fires" to another node if a threshold value is breached from its input neurons. From this base model, ANNs are constructed from multiple neurons, in varying quantities of layers and with varying threshold functions. Used as method for learning built upon the connectivist theory of intelligence. Initially they were thought to drive a revolution in computing as artificial brains. However initial progress was relatively slow, with the perceptron being a famous failure in that it couldn't learn the XOR logic due to being a non-linear problem. The effectiveness of ANNs is mostly being seen today, with the rise of deep-learning algorithms beginning to match the effectiveness of biological algorithms with processes such as complex pattern recognition in images (Kulwa et al. 2019) or document classification (Marinai et al. 2005). While ANNs provide distributed learning capability, they are typically used for converging on a particular result and therefore are not suitable architecturally for building distributed software systems, due to their non-dynamic nature.

Another popular biologically inspired technique is modelled on the theory of evolution such as with Genetic Algorithms (Davis 1991) and Genetic Programming (Koza et al. 1994). These processes find optimal solutions through the selection of the best offspring which are produced through recombination of previous generations as well as mutation. These new solutions allow the population to be adaptive but also robust to changing environmental conditions. This process allows survival of the overall species through the diversity which is introduced through mutations and allows adoptions. Whilst proven in a variety of situations they typically find solutions in an offline setting. Similar to ANNs these techniques are less suitable to be employed as a distributed system architecture due to their convergence and high resource cost.

Swarm computing is a common autonomic technique which is self-organising and self-optimising (Tambouratzis 2009). Swarm intelligence is developed from a large number of decentralised and

distributed, cooperating agents. Ant Colony Optimisation (ACO) is perhaps one of the most well-known examples. Proposed by Dorigo et al. (2007), he provides a fitting description of ACO as well as all other swarm algorithms when he states they are, for "distributed problem solving and optimization based on the result of low-level interactions among many cooperating simple agents that are not aware of their cooperative behaviour". ACO in particular was designed for finding the shortest path in a graph but since its conception it (and other swarm algorithms) have been used for solving different problems within computer systems. Networking optimisation (Hsin et al. 2014), intrusion detection (Gao et al. 2005) and pattern recognition (Tambouratzis 2009) are just some examples of these. These techniques are typically employed to solve a specific problem, as opposed to have support for diverse logic processing.

Artificial Immune Systems (AIS) are modelled upon biological immune systems. Forrest et al. (1994) proposed a system based upon the theory of an immune system's ability to distinguish infected cells from uninfected cells known as *negative selection*. T-cells within an immune system attach to an infected cell in numbers in order to destroy it, however each T-cell is designed to specifically attack only one type of infected cell. The t-cells are created with random features and any features which might bind to healthy cells cause the t-cells to be destroyed prior to entering the blood stream. Forrest et al. (1994) successfully used this scheme within an integrity checking system. Additional work in AIS adapts this scheme to include a concept within immunology known as "danger theory" which posits that t-cells detect danger vs. safe as opposed to self vs non-self. The difference being that the dangerous cells would exhibit a signal which would declare them dangerous. In computers, for example, this might be a symptom such as suspicious activity. AIS techniques are typically used for a classification problem and (as with swarm computing) lack the ability to arbitrarily compute.

Another bio-inspired autonomic technique is embryonics (Benkhelifa et al. 2013) (Sipper et al. 1997) (Miorandi et al. 2010). They are distinguished by their ability to inspire the creation of systems composed of multiple-cells which divide from the mother cell, each one containing the same code (or zygote) which allows it to perform any function as the other, ensuring it is capable of self-replication, and thus the organism's ability to self-heal. The way in which the cells differentiate varies according to their surrounding neighbours and therefore the systems develop according to the functions required as a whole and from their neighbours, although flaws in the original genome will be present in every single cell. The characteristics of adapting and self-healing through cellular differentiation and division are advantageous for resilience. In addition, embryonics share an affinity for distributed processing using a cell-based architecture. Therefore, this work selects embryonics as a proposed autonomic architecture for providing resilient decentralised cloud service delivery which is discussed below.

2.5.2 Embyronics for Resilient Decentralised Cloud Service Delivery

Embryogenesis inspired electronics have shown that resilience may be achieved by modelling a system on these self-healing capabilities (Benkhelifa et al. 2013). Some early examples of similar systems in computing are Von-Neumans self-replicating automata (Von Neumann et al. 1966). They have been applied to electronic engineering to provide circuits with the ability to self-heal (Sipper et al. 1997). Embryonic software has been shown to provide similar self-healing properties for distributed systems (Miorandi et al. 2010) and is therefore proven in its application for the management of complexity for resilience. Therefore the concepts of *cellular differentiation* and *cellular division* of animal embryonic development are employed to provide self-healing functionality in a multi-agent system. Within embyronics, cells (which compose the system) employ the MAPE-K control loop to permit distributed self-organising, self-healing and adaptive behaviour.

Embryonic techniques have a strong affinity with the requirements for resilient decentralised cloud service delivery:

- Embryonics are distributed systems composed of many atomic units. The atomic component is the cell, whose goal is to deliver a specific functionality. Collectively, many cells functioning together create more complex functionalities which emerge from the interactions of these cells. This characteristic is easily suited to distributed service delivery environments, such as cloud-based systems. This is because services within cloud-based systems are composed of many interacting, loosely coupled units which each provide a distinct functionality. These are usually virtualised machines or containers, which collectively can be employed to create more complex services out of these atomic functions.

- Embyronic systems have fundamental characteristics of cellular differentiation (functionality specialisation) and division (replication). These characteristics give rise to self-healing behaviour. The system is able to respond to internal and external changes and self-heal in order to maintain the overall functionality of the system. This provides inherent service resilience and is a strongly beneficial requirement of a resilient cloud platform.

- In addition to the self-healing functionality, an additional behaviour which emerges as a result of cellular differentiation and division is self-organisation. Differentiation occurs according to the need for specific functionality and therefore cells collectively organise. This ability to self-organise meets another requirement of resilience decentralised cloud service delivery. Whereby service management and orchestration can occur void of centralised control.

This work proposes that through modelling distinct application functions as embryonic cells and then employing the characteristics of cellular division and differentiation, a service delivery architecture for decentralised cloud environments which is resilient by nature can be developed. The rest of this thesis will promote the investigation of this approach and will evaluate its efficacy.

2.6 Conclusion

This chapter reviewed the necessary background literature in resilient decentralised cloud environments for this thesis. It began with a review of IoT technologies, whose devices and technologies are the users of cloud services. Due to emerging low-latency and safety critical use-cases, there are now greater requirements to have cloud service delivery physically closer to or even upon these devices. Due to the constrained nature of these devices and the hostile environment in which they operate, providing services resiliently is essential to their operation. A review of different resilient techniques in decentralised cloud computing highlighted that currently no architectures permitted decentralised cloud service delivery distributed in a resource efficient and resilient manner. In particular, it was highlighted that autonomic techniques create environments which are resilient by nature but solutions of which are largely lacking. Due to the initially high development cost for these techniques, they are often inspired by biologically processes and systems. Embryonics was highlighted as a technique with an affinity for the proposed environments and resilient requirements so was selected as a technique for investigation within this work.

Chapter 3

Embryonic Model for Resilient Cloud Service Delivery

3.1 Introduction

In the review of the state-of-the-art in decentralised cloud resilience, the need for autonomic management solutions and their lack in literature highlighted their suitability for this task. Many autonomic techniques are developed through inspiration from biology due to their inherent self-adaptive characteristics. Embryonics was selected as a bio-inspired technique for this study due to its characteristic affinity for resilience and decentralised cloud environments. Consequently, this research leads into the investigation of the suitability of embryonic techniques for providing resilience in decentralised cloud environments.

In order to determine if embryonic concepts can be applied to cloud environments in the interests of providing a resilient platform, this chapter defines requirements for decentralised cloud functionality. It then maps these to derived characteristics of embryonic development. A conceptual model is then developed closely to the functional requirements in line with their available resources and environments. Finally, an analysis of the conceptual model is given to assess it's suitability in achieving the previously defined requirements.

3.2 Resilient Cloud System Requirements

As a follow up from the previous definition of resilience, now the requirements for a novel highly-resilient cloud architecture are given. These are broken down into two categories, functional requirements to provide a cloud architecture and requirements for resilience.

3.2.1 Functional Cloud Requirements

In order to provide a cloud service, the following characteristics are required of the architecture:

- As the purpose of a cloud service platform is to replace the traditional computing facilities, a resilient decentralised cloud will also provide standard computing capabilities: computation, storage, communication. Each resource is varied according to the user's specific needs and constraints.
- Continuously emerging and evolving cloud disciplines are driven by evolving user-cases. A resilient cloud model will be required to deliver a resilient service across emerging decentralised architectures. In particular, those which are particularly resource-constrained, mobile and wireless.
- The system should expose an interface which permits a user to upload, execute and interact with an application which provide an on-demand and automated service for the provisioning of the aforementioned computing resources.
- The system should permit an application to scale up and out according to load, cost and other service requirements, in an autonomous manner. This would be within the maximum constraints of the system.
- The system should have the capacity to provide resource monitoring for metering and billing This is essential for the underpinning agreement of the cloud service which is the SLA. It is also necessary to maintain the economy driven model of cloud environments.

3.3 Embryonic Development Characteristics

To meet the requirements, this study focuses upon testing the resilience of a cloud architecture inspired by embryonics. Embryogensis is the development of a biological being from a mother cell - the zygote. This biological process provides a high degree of resilience through redundancy and self-healing. Embryonic inspired electronics have shown that resilience may be achieved by modelling a system on these self-healing capabilities (Mange et al. 1998) (Benkhelifa et al. 2013), whilst embryonic software has shown similar self-healing properties for distributed systems (Miorandi et al. 2010) and is proven in it's application for the management of complexity for resilience. This section will briefly discuss the basic process of embryonic development and cellular self-repair so as to derive some features to enable future modelling. This is a high-level abstraction and is by no means comprehensive.

The development of a biological animal is accomplished through *embryogenesis*. It is a process of iterative *cellular-division*, whereby each cell will contain the information, DNA (Deoxyribonucleic acid) or genome, to create additional cells. The cells begin as stem-cells, which are *pluri-potent*, in that

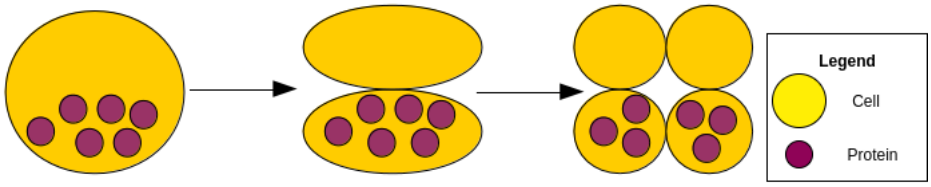


Figure 3.1: Asymmetric Segregation of Protein Determinants

they can develop into any cell. Whilst all cells in an organism will share the same DNA, different cells will express different genes, which in turns causes differing proteins to be made and in effect will cause the cells to develop differently; also known as *cellular differentiation* (Wolpert 2008). This process enables extremely complex, multi-organ biological systems to develop from a single cell and with the process of self-check and cell-division, this organism has the ability to self-repair and self-organise (Mange et al. 1998).

Cellular-differentiation may be instigated through a number of means which may be either *internal* or *external* to the cell. Internally, the selection comes about through the presence of transcription factors, which are proteins found within the zygote. They are spatially-distributed according to the DNA and as the cells divide, those that remain in the same location denote the proteins which will be activated. This process is knows as *asymmetric segregation of protein determinants* (figure 3.1) (Knoblich 2010).

Externally, the cell can receive a prompt from other cells in a process known as *inductive signalling* or induction. This may occur in three forms: a) diffusion from a group of cells b) direct contact with another cell c) via a gap-junction between cells (figure 3.2) (Rudel & Sommer 2003). These cellular signalling methods are analysed in further detail in the next section.

Once the organism has developed, it maintains the ability to self-repair certain tissues through the application of stem-cells, also known as *somatic stem cells*. Typically, these cells, which are from an already developed animal, will be *multipotent* at best and thus only able to differentiate into a subset of the collection of possible cells. Pluri-potent stem-cells are less common although may be induced through artificial means (Mitalipov & Wolf 2009) (Schöler 2016). Table 3.1 lists the coverage of different cell potencies.

Self-repair will occur due to cell-death, which in turn may be instigated in a number of forms. Programmed Cell Death (PCD) or *apoptosis* which is cell-suicide and can occur due to an intrinsic prompt e.g stresses to the cell causing chemical changes, or damage to the DNA; or may be due to an extrinsic prompt from proteins binding externally to the cell. An alternative method is *necrosis* which is due to an external prompt such as trauma or infection. Necrosis is considerably more traumatic to the processes and other cells within the body than apoptosis (Majno & Joris 1995).

To summarise, embryonic systems consist of the following key characteristics:

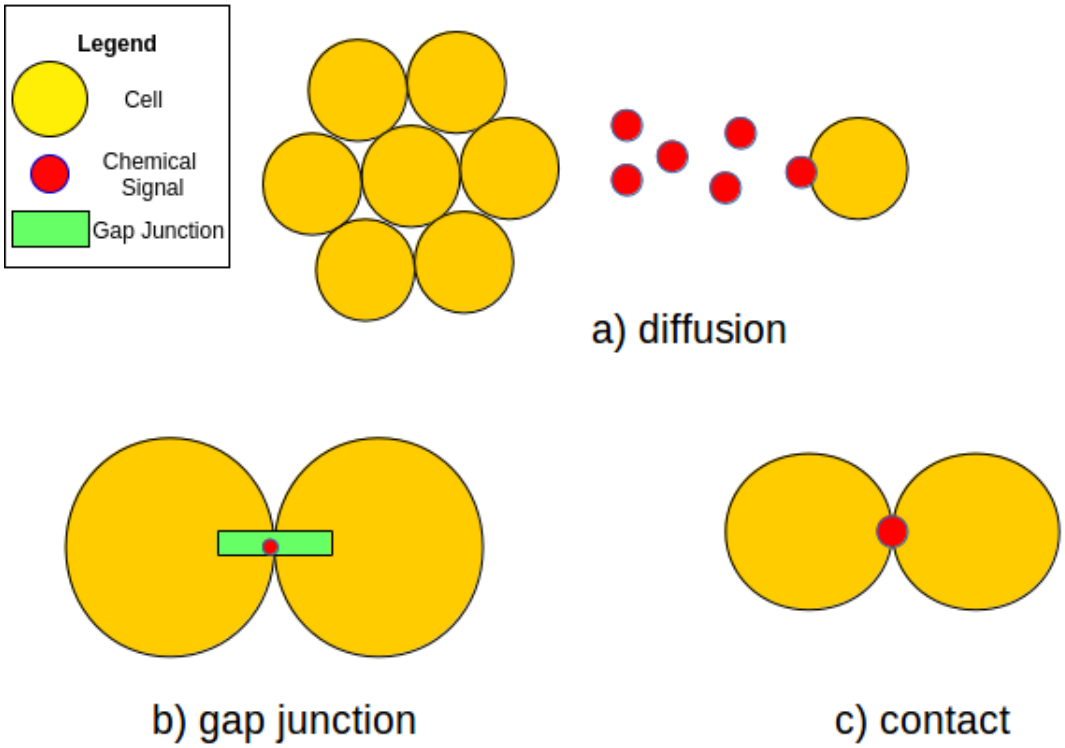


Figure 3.2: Inductive Signalling

Potency	**Coverage of Cell Types**
Totipotent	Embryonic
Pluripotent	Any type of cell
Multipotent	Multiple but similar
Oligopotent	A minority of cells
Unipotent	Only the self cell

Table 3.1: Cell types and the range of coverage for different cells.

- **Genome** - all cells contain the same genetic material. The genes which are enabled denote the proteins which will be made and thus the function and form of the cell.
- **Division** - a cell may self-reproduce through division.
- **Differentiation** - cells become specialised to a number of different functions and through different means.
- **Self-repair** - an organism can repair through the application of stem-cells although the potency of each cell varies the type in which they can reproduce.

The aggregation of these characteristics will be the basis for the structure and architecture of the resilient cloud architecture.

3.3.1 Cellular Signalling

In addition to the development and self-repair functionality detailed previously, it is also necessary to understand the communication methods of multi-cellular systems, which is particularly relevant for development and self-healing.

Cells must have the right receptor (ligand) to receive a message and thus not all cells will receive all messages. Therefore signals are only acted upon by a cell if the cell has the correct receptor and thus supports that signal type. Therefore signals are received and processed according to their purpose, as opposed to be directed towards a specific cell (Dressler & Akan 2010).

There are a variety of different forms of cellular signalling. Predominately they differ in the distance and recipient of the message, although some usages and features vary. These are *autocrine* where a cell communicates with itself or the same type of cell. *Paracrine* used for communication with cells within the immediate vicinity. *Endocrine* which is used for long distance/scale is based on hormones which provides global control such as encouraging growth and physiology. *Juxtacrine* which must be direct contact with an adjacent cell and could operate through either: gap junctions or direct contact via bind due to corresponding receptors (Ben-Jonathan & Liu 1992). A comparison of these signalling methods is presented in table 3.2.

3.4 Embryogensis for Resilient Decentralised Cloud Computing - Feature Mapping

The natural resilience of embryogenesis is indubitable due it's self-healing functionality. This section illustrates how the characteristics of embryogesis can aid in achieving the previously defined requirements for a resilient decentralised cloud architecture. In order to correctly leverage these characteristics into a resilient architecture, the appropriate features must be mapped to the previously

Type	**Propagation**	**Usage**	**Message Distance / Scale**
Autocrine	Self / Same	Development (reinforcement) / Pain / Inflammation / Self-destruction	Low
Juxtacrine – Gap Junction	Local	Differentiation / State Info (coordination)	Very Low
Juxtacrine – Contact	Local	Differentiation / Immune (safe non-safe)	Low
Paracrine	Vicinity	Differentiation / Behaviour	Diverse but Streamlined/Quick/ Degrades Rapidly/ High Concentration
Endocrine	Long Distance/Scale	Most common. Hormonal used for global control. Physiology and Growth	Slow and Long Lasting Low concentration

Table 3.2: Cellular Signalling Methods

defined requirements. Table 3.3 refers to the combined features of a resilient cloud architecture (as in section 1) which are mapped to the appropriate embryonic features and corresponding implementation method.

Number		Requirement	Represented as	Implementation
Functional	**F1**	Provide standard computing capabilities: computation, storage, communication.	Cell	Network Node
	F2	Deliver across decentralised cloud models.	Constrained / Hierarchical Functionality	Virtual Instances
	F3	Expose an interface which permits a user to upload, execute and interact with a an application.	Cell receptors	RESTful API, Message-oriented communication
	F4	Provide an on-demand and automated service.	Multi-cellular Organism, Cellular Signalling	Autonomic management
	F5	Permit the application to scale up and out according to load, cost and other service requirements.	Multi-cellular Organism, Cellular Signalling	Autonomic management
	F6	Provide resource monitoring for metering and billing.	Higher-level Platform	Cloud manager
Resilience	**R1**	Purely distributed architecture to exclude central point of failure.	Multi-cellular Organism	Multiple nodes with network overlay
	R2	Provide universal redundancy.	Cell division	Self-reproducing software
	R3	Provide diversity in the service network.	Cell diversity	Cross platform support, geo-distribution
	R4	Enable the selection of appropriate resilience features such as security based characteristics.	Differentiation	Software libraries
	R5	Permit the ability to verify the integrity of any node.	Autocrine DNA check	Cryptographic node hash
	R6	Provide dynamic self-organisation for service-composition and self-healing.	Multi-cellular Organism, Cellular Signalling	P2P Overlay Network

Table 3.3: Requirements to Feature Mapping

3.4.1 The Cell

The cell is the only architectural component. Total system functionality is distributed across cells to create a true P2P network. Cells differentiate according to the functionality required by the system and are networked together to provide platforms for the cloud architecture. Therefore their software functions and their collective network structure may vary, but their internal architecture does not.

The components of the cell are illustrated in figure 3.3 and described as follows:

- **Genome** - will differentiate to a particular software function, as required by the global network. It will only process data of its function type, ignoring other messages. Once the data has been processed it passes the output to the node, for distribution to other cells or the end user.
- **Node** - uses the publisher/subscribe communication pattern to pass messages between other cells. There are two types of messages, data messages to be processed by the corresponding function and organisational messages. Organisational messages involve broadcasting known local node addresses, the current function types the cell can see, and requests for differentiation.
- **RESTFul Application Programming Interface (API)** - is exposed to the end-user/devices. It is used to push data to be processed onto the MC platform and also to return results.

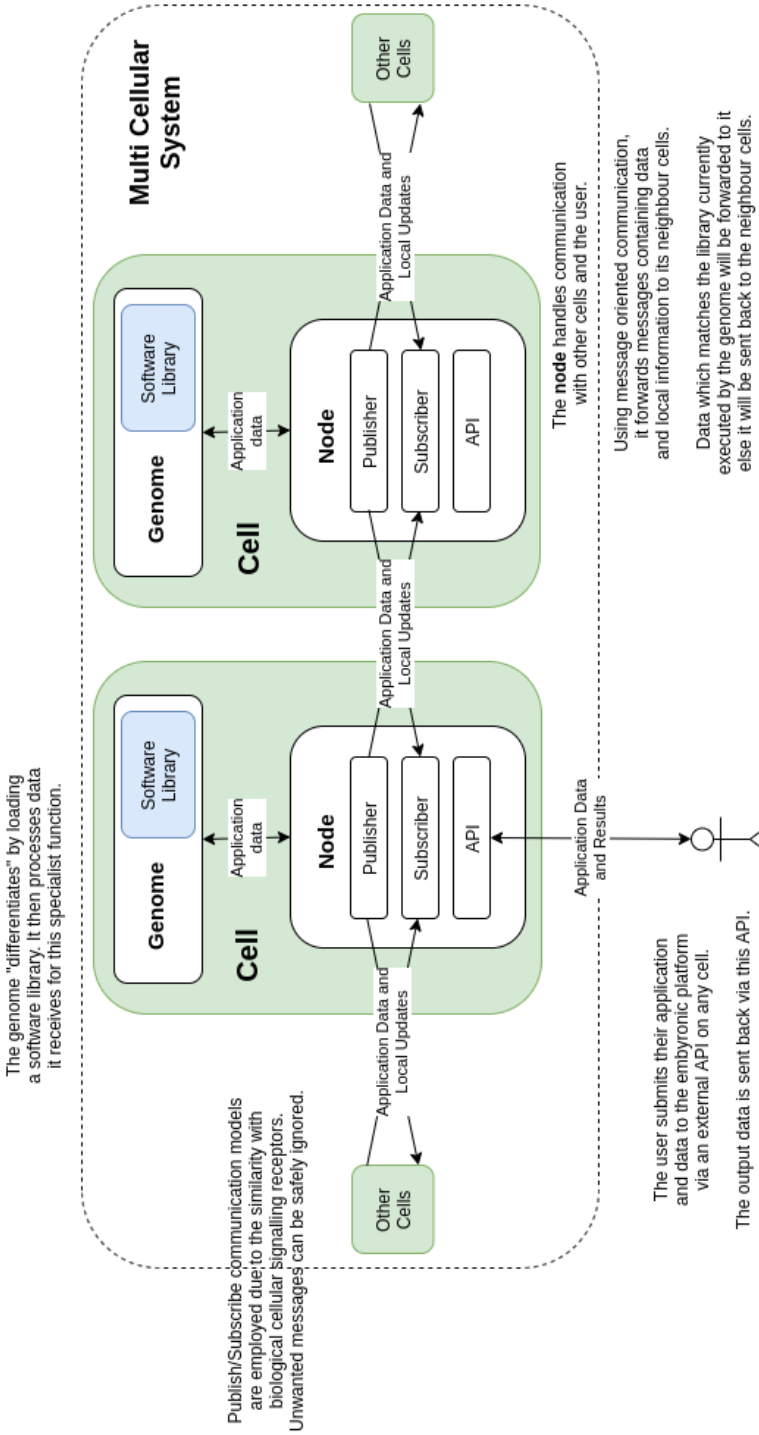


Figure 3.3: Cell architecture

3.4.1.1 Cell Functionality

The cell consists of a number of concurrently processing components. The different functionality of the cell is described below.

Cell Inception - Upon creation the cell must communicate with its neighbours. If it is not the first cell it will be given the address of its mother node upon start-up. This address is stored as its closest node and will use it to request further information about the network. If it is the first cell it will divide to provide further functionality as required. The cell receives "DNA" from its mother cell which defines the parameters of its life. This can include how many times it should divide, and the range of applications (genes) it can execute. Figure 3.4 presents this process in a flow chart.

Node - Organism Functions - The node will listen from its peers for signals. If a message contains data appropriate for its genome then it will be passed for processing. It forwards messages to its vicinity according to the message hop count and current resilience requirements.

Node - API Functions - The node will listen for requests, and pass responses to, an external client interfacing with the API. These will be processed and/or forwarded to other cells as appropriate.

Differentiation - involves activation or deactivation of software functions in order to provide specialism for a particular purpose. Appropriate genes will be toggled which will enable the corresponding functionality. The cell will then process requests for this functionality from the organism as well as advertise capability when appropriate. The differentiation process will occur as the cell is spawned, through a prompt from the MC organism or self-initiated according to an internal decision made according to information about requirements and the state of the network.

Self-check Integrity Verification Periodically, or when prompted, the cell will perform a hash-check on its internal genome to verify its integrity and will respond to challenges from adjacent nodes. Failure indicates that the underlying code is subverted or corrupted. If this check fails then the node will be cut off from the rest of the network or will self-destruct.

Self-reproduction Through an external prompt or internal decision the cell will self-reproduce, generating a new version of its application and notify the new cell of its heritage with corresponding DNA to establish communication.

3.4.2 Multi-Cellular Organism

The multi-cellular (MC) organism is an autonomic network with self-organising and self-healing properties. It is a composition of an arbitrary and dynamically determined number of atomic cells with the purpose of executing numerous, scalable applications simultaneously and in a resilient manner. Therefore it only consists of one architectural component, the cell, yet maintains the resilient service delivery provision through collective self-organisation. Its functionality is discussed below.

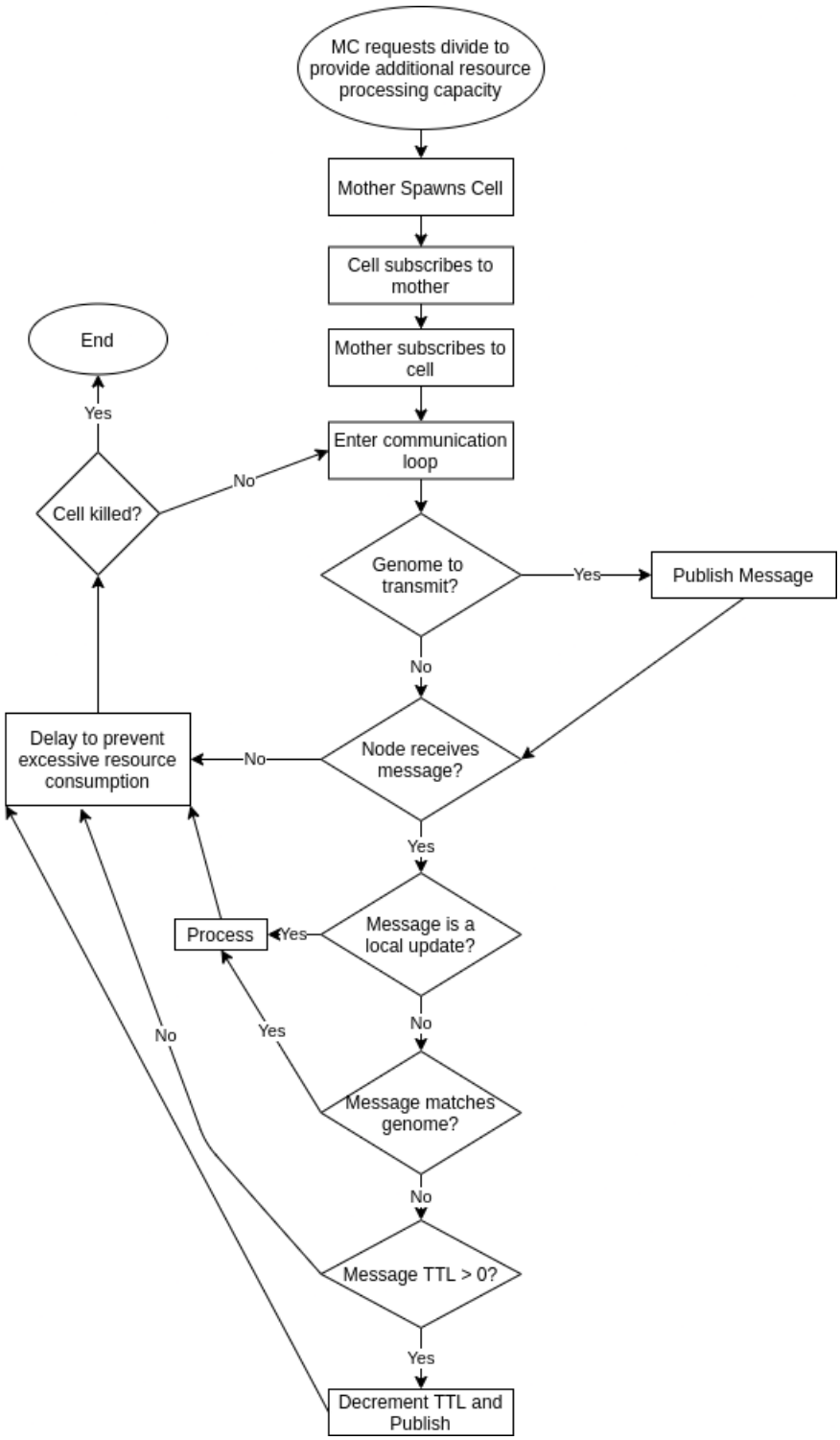


Figure 3.4: Cell Inception and Communication Loop. This flow chart illustrates the process by which the mother cell self-reproduces. This child subscribes to it's mother then loops, waiting for messages. It will receive all messages but process only those relevant to it.

3.4.2.1 Communication

Operating in a hostile, mobile and wireless environment requires a system to accommodate high-levels of node churn. Due to this, the ability to maintain routing tables is deemed unsuitable, therefore flooding is used to distribute messages. Subsequently this architectural model employs local-only communication to mitigate network link contention. Each cell will receive messages from all other cells it is subscribed to, however it will only process those messages which are related to its particular function, or any global messages. This flooding based communication will put excess strain on communication links due to the increase in sent traffic. Therefore a number of operations must be tuned to minimise this link contention whilst still enabling services to be delivered in a resilient manner. For example, a cell may choose to propagate messages further to other cells that subscribe to it, depending upon the platform characteristics and Time To Live (TTL) of the message. Cells will also not make collective decisions, request information or respond to queries in order to reduce communication. Instead, each cell will act in a local manner, using gossiping methods (Haas et al. 2006) to provide information to its local peers about the current environment and can then choose to act accordingly.

To accomplish this, the communication functionality employs local only (juxtacrine), i.e. nodes will only communicate with their immediate neighbours and therefore no medium or long range communication, or routing, will occur. An example message could consist of the cell informing its neighbours of its current state (e.g. genome function and availability) or data for applications.

Due to the constraints upon the networking features, the applications placed upon this system will have less functionality. The state of each application will now be contained in the transmitted message, which will allow it to be executed upon any node and thus permit a higher degree of dynamism in the system, such that the loss of one node should not adversely affect any one application. The consequence of this is providing less power and customisability to each end-user application. However whilst this might seem less useful, such a constrained system would be more applicable to decentralised cloud architectures. It has been shown that Fog and other edge architectures are more suited to the use of SaaS/PaaS architectures due to the devices' constrained nature (Dastjerdi & Buyya 2016) (Vandebroek 2016).

3.4.2.2 Application Management

A service is an application composed of a number of different distinct software functions in a micro-services like fashion. This division of distinct functionality relates to the concept of cellular differentiation. The application data resides in the passing of messages and therefore no crucial data persists on any cell. Figure 3.5 presents the message based application execution which has reduced functionality when compared to the unconstrained architecture. As opposed to writing and executing a script or application (e.g. in an interpretive language such as Python or Ruby) the user must now define the

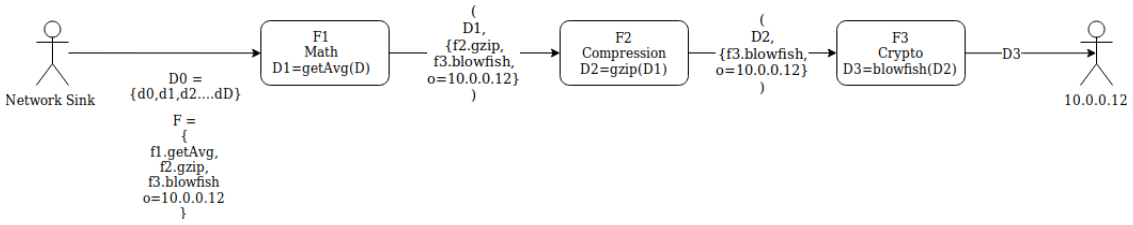


Figure 3.5: This diagram illustrates the constrained application execution. The application starts as a tuple of data (D) and functionality F). The data is passed to the top function of the list which is removed as the data is passed to the next cell. The current state of the application therefore resides in the message.

data and function calls in a tuple, which will then be iterated upon until it reaches its final destination and is sent back onto a network external from the MC.

The primary goal of the embryonic platform is to persistently deliver applications regardless of threats to its operation. Micro services based paradigms partition applications according to distinct functionality in a loosely coupled manner where an application is composed of a number of different microservice functions (Thönes 2015). An application is argued to be successfully delivered if all functions can communicate, permitting data to flow from start to finish. This may be quantified as its *connectedness*, an application which is fully connected at any point in time such that the next function is fully connected. An application which has temporarily lost connectivity is partially connected. This requirement for connectedness drives the autonomic management of the system which attempts to distribute the spread of functions in an optimal manner.

A service is delivered successfully if all required functionality of the application can be reached, in the required order within the networking hop constraints. These constraints will be set according to the network resilience requirements in order to reduce excessive communication due to the flooding-routing. The purpose of the platform is to autonomously operate in a way which maximises the connectivity between different applications according to the user's requirements in the face of underlying internal and external changes to the system. Due to the state of applications residing in the messages and not the cells, the application management is markedly less complex and mostly consists of optimising the distribution of function types across the network according to prompts from cells and the user. The autonomic functionality to accomplish this is described below.

3.4.2.3 Self-organisation and Self-healing

In order to optimise the structure of the network to maximise the connectedness of functions executing upon genomes, the underlying platform self-organises in a decentralised manner. Self-healing of the network occurs through cells dividing according to requests for applications, or through noticing that capacity has been reduced. Through simply dividing and differentiating, cells refill capacity

and therefore the overall system self-heals. This autonomic functionality is chosen over a centralised organisation mechanism for resilience purposes. Through reducing any central point of failure the application can continue to operate in the face of node subversion/destruction. However these decentralised mechanisms come at increased operating overhead and communication complexity. The original protocol design (first defined in Appendix B Paper 2) proved to be too complex and posed a number of security flaws. Therefore a minimalist approach was taken in an attempt to reduce the amount of possible organisational functions, attack service and communication overhead. This drove the architecture to be more suitable for the most resource constrained environments. The chosen technique leverages gossiping protocols, which in contrast to the original design which involved a number of request and response communication functions, in the new design, cells will continuously gossip in a localised manner about their current view of the network and opt to make their own decisions about how to act.

Cells periodically broadcast their view of the network consisting of their current function and the last time they saw functions local to them, in keep-alive packets. A cell will check the updates received to determine if their particular function is over-represented. This could be due to local visibility, extremely low update times or a more complex algorithm. If the cell determines that its current function is over-represented it will differentiate to one deemed under-represented. For example, if a cell were to see many neighbours processing cryptographic data they might differentiate to a function for compressing data, after seeing corresponding requests being not-processed. Figure 3.6 presents an example, where cell 1 broadcasts keep alives to its neighbours. After the first broadcast both cells which have differentiated to function 3 will notice that another function 3 is being regularly updated. After a random amount of time one or both of the cells will choose to differentiate to one that can't be seen in order to minimise the likelihood of updates not being seen and having both cells differentiate to the same. This is a race condition as it is possible that both cells will differentiate causing 3 to no longer be represented well. A number of factors will be put in place to mitigate this such as random timeouts, verification of time between keep alives or more complex distributed verification algorithms.

User Interaction The previously discussed functionality which enables the architecture to self-organise and self-heal is only relevant when considered within the context of user/node interaction which provides SaaS functionality. This section illustrates the way in which the service requests from a user, drive the autonomic aspects of the platform.

Figure 3.7 illustrates the entire autonomic processes of the platform via interaction from the end-user. A user (person or device) will submit their application and data to the API of any cell. The cell will then check its latest information about the network state (achieved through gossiping). If the full capacity to process each function within the application does not exist or the delay is too long, the cell will request it's neighbour cells to differentiate, divide, or divide itself if none are able to. Once enough functionality is available to process the application, the user will begin transferring the

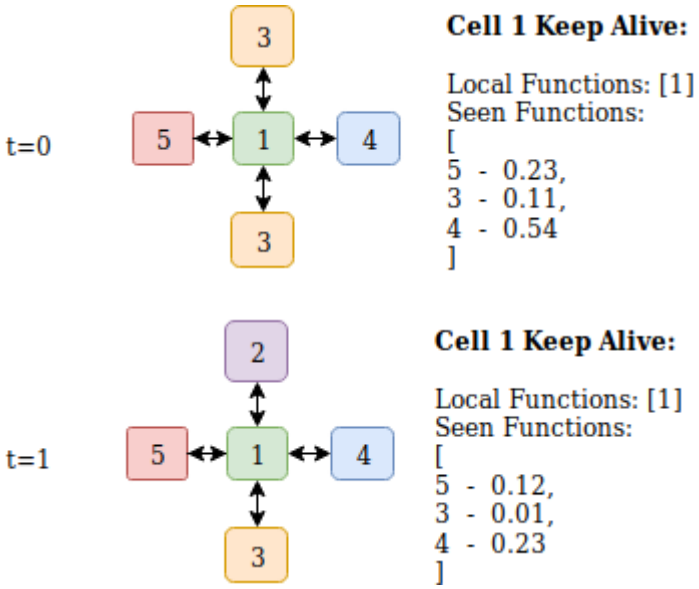


Figure 3.6: This diagram illustrates to time steps in the platform showing the keep alive being broadcast by cell 1 to its surrounding neighbours. In the first instance one or more cells differentiated to function 3 will notice that there is another cell in the vicinity at which point they will differentiate to an under represented function in the next instance.

first message which will be subsequently propagated through the network until its processing has been completed. This enables the platform to scale up to process data as needed. After a predetermined time of no data processing, cells may self-destruct according to a predefined timeout. This can assist in freeing up resource capacity for later divisions.

3.5 Use-Cases

This section will discuss a number of real-world use-cases deemed suitable for the architecture. Specifically these are cases where constrained devices, require low latency cloud like data processing in a resilient manner. Most of which may suffer from partial link failures due to mobility or total failure due to node subversion or destruction.

A number of use-cases are highlighted within the application of IIoT where a large number of nodes collect data on the edge (Aazam et al. 2018). Mining (Singh et al. 2018), transportation, crime (Neto et al. 2018) and agriculture (Heble et al. 2018) are examples where IoT systems may operate in environments with dynamic and non-deterministic environmental conditions in addition to device mobility. The next few subsections present some fictional scenarios which have been based on a number of these use-cases. These scenarios are discussed whilst illustrating the application of the proposed embryonic architecture to illustrate its benefit over alternative solutions.

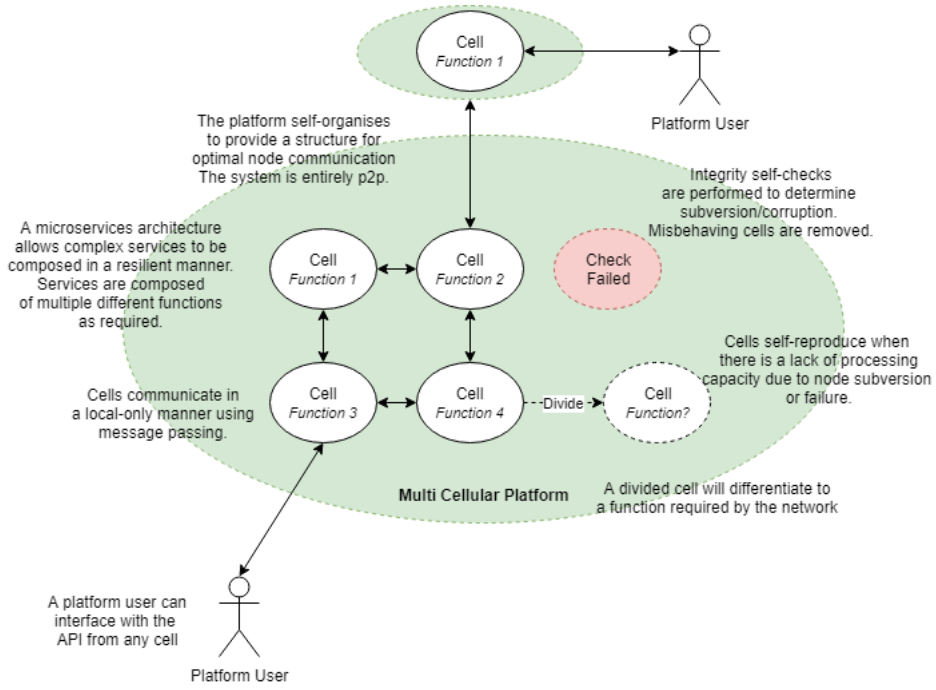


Figure 3.7: The architecture inspired by animal embryonic development, composed of Multi-Cellular Platforms (MC). It provides resilient service delivery in hostile environments through leveraging the concepts of self-healing, cell division and differentiation.

3.5.1 Smart Transport with VANETS

Vehicle Area Networks (VANETs) are an emerging concept which will become more commonplace with the rise of autonomous vehicles. Cooperative driving between smart vehicles, optimised traffic management and emergency response will be enabled through the sharing of information concerning threats, location of vehicles and other actors, and environmental conditions etc. The highly mobile nature of the nodes, delay intolerant nature of the data processing and safety critical nature of the applications are only some problems which VANETs must overcome. As with the work by Pereira et al. (2019), VANETs which leverage a fog architecture consist of On Board Units (OBUs) which provide data processing on the vehicle, Road Side Units (RSUs) which provide data processing at the base station, and the back-end cloud platform which provides enhanced data processing and storage capabilities. Figure 3.8 illustrates this architectural model with the MC nodes overlayed.

The MC platform can provide a variety of low-latency data processing tasks to enable vehicles to communicate with each other without having to reach the cloud. As node mobility inevitably causes partial-failures, the self-healing aspects of the platform will enable data processing to continue.

3.5.1.1 Potential Application - Cooperative Vehicle Telemetry

VANETS may support autonomous vehicles in the future by permitting them to collaborate through the sharing of information. This might, for example, include the awareness of possible incidents such

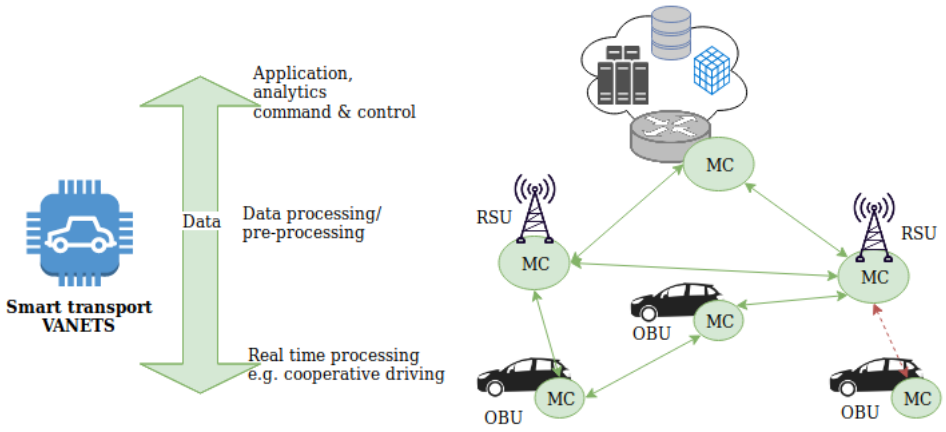


Figure 3.8: Multicellular architecture leveraged as a SaaS platform for smart transport and VANETS. At the device, the platform provides resilient, distributed data processing to enable co-operative driving amongst mobile device. As data moves towards the cloud it can be pre-processed to optimise resource consumption where it will finally reach the cloud in order to provide full features analytics and control.

as collisions or bad weather conditions. It might also involve the sharing of real time traffic information to mitigate congestion or simply to process and send this important data to the cloud to be processed to provide benefit to all. This use-case presents such a scenario, where critical car telemetry data needs to be processed and sent to the cloud in a resilient manner, which is described below.

As the car is conducting its journey from point A to point B, data is periodically forwarded to the cloud about it's current location, speed, known environmental conditions (e.g. the state of the road and weather) and any other important environmental information. This information is important for legal and insurance purposes so must be processed correctly following strict data-provenance methods.

During a routine trip the vehicle periodically moves in and out of range of nearby RSUs so data is collaboratively processed between nearby vehicles. A collision occurs between two vehicles whereby both vehicles are no longer able to process or send data. The speed of the collisions ensures that the most recent data was not processed but fortunately it had been pushed to other cells on external devices and continues its journey to the cloud. Figure 3.9 illustrates this scenario.

Without the multi-cellular network the data would struggle to reach the cloud or it would only be half processed, the collaborative nature of the data processing ensures this. Additionally, redundant messages and links increase the likelihood of the data arriving despite the loss of link to the back end cloud and through the originating system's destruction. In this case the data will provide crucial evidence for crash investigation in addition to warnings for local vehicles.

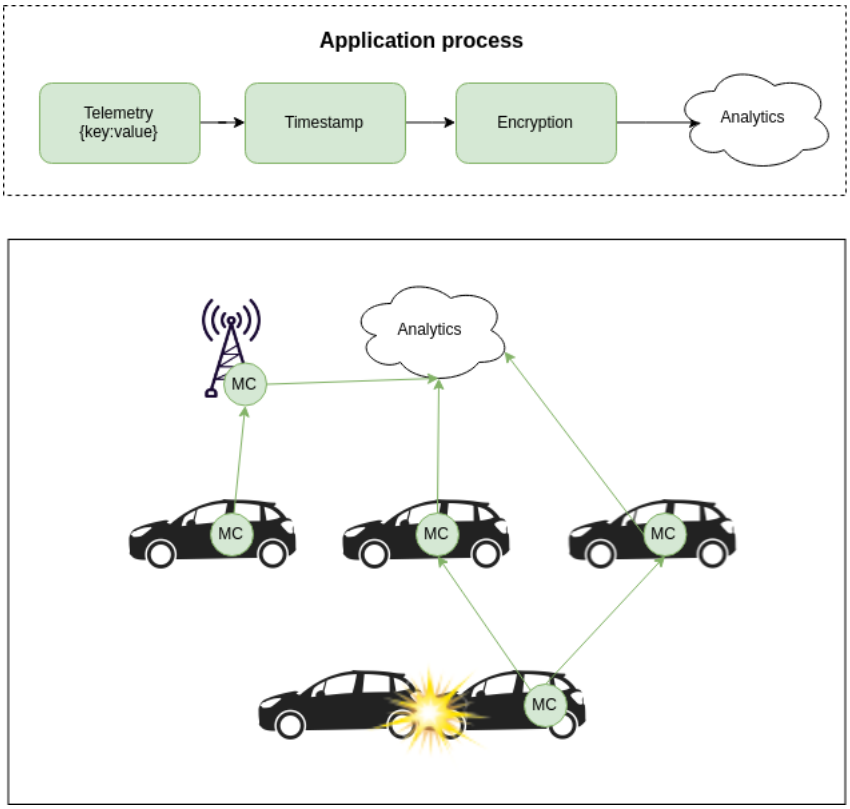


Figure 3.9: A use case illustrating resilient and collaborative data processing

3.5.2 Adversarial Warfare

Military applications for information processing are seeing increased deployment in order to provide an advantage over the enemy. Application types are many which include secure communication, ordinance targeting, remote system commands or even cooperative unmanned vehicle coordination (Tortonesi et al. 2012). Consequently, transmitting nodes may be jammed or destroyed in order to disrupt supporting operations (Amin et al. 2015). Aerial vehicles can be used to provide bridging between distant terrestrial nodes, although link failure can be induced by device mobility and external attack. Therefore communication between terrestrial nodes may be frequently disrupted.

This scenario therefore consists of an environment in which node failure is an almost certainty and data processing requirements are critical. Therefore there is a very high resilience requirement. Traditional mitigation techniques focus upon disruption tolerant networking approaches. In this instance the MC architecture intends to provide persistent service delivery despite conditions of the underlying infrastructure.

3.5.2.1 Potential Application - Secure and Resilient Communications

In this scenario, the MC architecture is distributed across all nodes in the battlefield. The application used is a simple one, consisting of a destination identifier (a cryptographic hash) and a communication

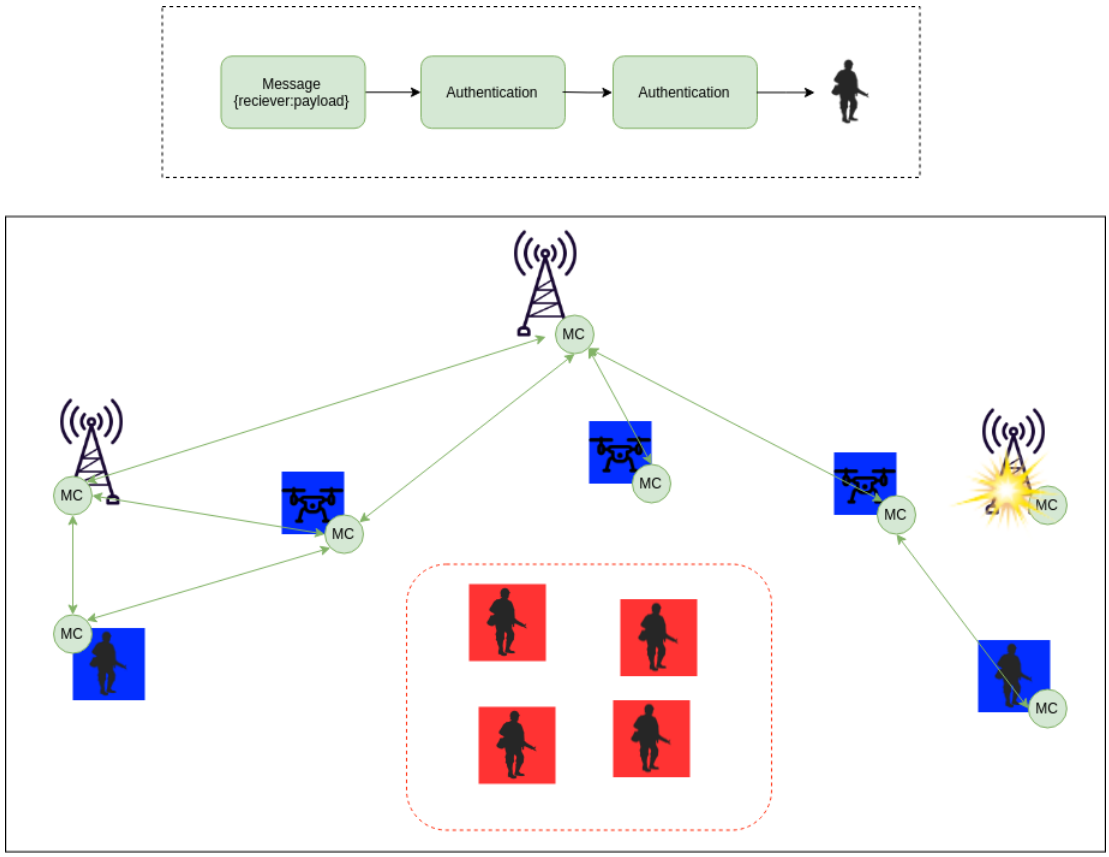


Figure 3.10: Example MC Architecture in an adversarial environment.

payload. As the message is pushed from node to node it is authenticated to further enhance the security of the message. Messages that cannot be authenticated will be dropped in order to prevent misinformation from the adversary. Aerial support units both leverage the platform for communication and data processing whilst additionally providing redundancy for communication links.

Figure 3.10 illustrates such a scenario where the blue team have deployed MC nodes across all available architecture. The soldiers on the left and right can communicate in a resilient manner using the MC architecture as their destruction or disruption does not prevent the data from continuing to be transmitted. If a node is to be destroyed, the initial payload can still be processed on future nodes. Finally, constrained nodes (such as the mobile units) can leverage the richer processing of stationary nodes to provide enhanced data processing in a cloud-offloading manner.

3.5.3 Industrial IoT in hostile environments

There are numerous cases discussed previously where the application of IoT to industrial environments results in these technologies being deployed in hostile conditions. Similarly to warfare situations, networks of nodes operating in these environments will likely have an adversary which results in the destruction of nodes or communication links between them. However in these instances that adversary is more often than not weather phenomena, although the potential for human-driven attack still exists. The disruption may be predictable, permitting the system to adapt to provide higher resilience in times of need.

The requirements for these environments may vary although the likelihood of data collection and decision making to perform some actuation is high. Additionally the potential for financial loss due to the result of incorrect information causing the mismanagement of industrial machinery is also high. Therefore resilient, autonomous, and accurate data processing is required.

3.5.3.1 Potential Application - Robotic Marine Data Processing

In this scenario, a network of autonomous sensors and robots are operating in a remote marine location. Their goal is to collect data concerning a variety of environmental parameters including the local weather, the chemical consistency of the water and the known quantity of some marine life. The nodes are constrained to minimise power usage and widely dispersed to create an accurate data sample of a wide area. These combinations of characteristics, combined with the hazardous weather, variable power and node mobility ensures that link failures are frequent.

The MC architecture is therefore deployed across all nodes in the environment. In this instance, environment data is sensed, then data processing consists of a number of steps which culminates in a validation of the data before a collective decision is made by all MC nodes. As with previous scenarios, once transmitted the data can still be processed through the iterative functions of the application despite link or node failure. Redundancy in processing will enhance the ability of the

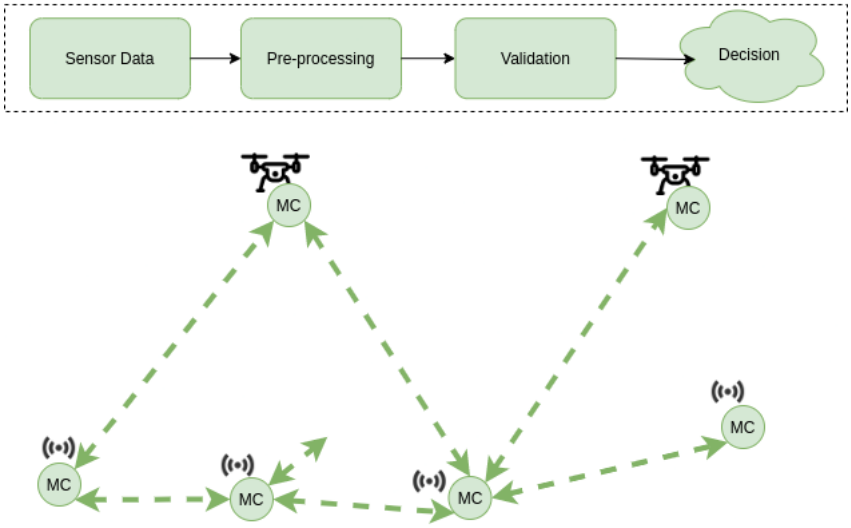


Figure 3.11: Example MC Architecture in an industrial IoT hostile environment

system to successfully process the application. Finally the distributed data processing will allow resource constrained nodes to perform complex calculations in a timely manner which might otherwise not be possible. Figure 3.11 illustrates this scenario.

3.6 Summary

This chapter has presented a conceptual model for a resilient decentralised cloud architecture inspired by animal embryonic development. A review of embyrogensis characteristics gave way to a mapping between decentralised cloud resilience requirements and this features. The purpose is to determine if a cloud platform could be developed in it's likeness, enabling cloud functionality and also maintain a high-level of resilience using these embryonic development characteristics. A model was developed which employed cellular signalling methods and reduced networking communication to a flooding-based local-only communication. As routing-based methods were unsuitable due to the potential for high levels of node churn. The chapter ended with some suitable use-cases.

Chapter 4

Cellular Automata Embryonic Simulation

4.1 Introduction

In the previous chapter, an architectural model based upon embryonic development was presented for a decentralised cloud service. The architecture was designed to provide cloud computing functionality yet in a manner which has greater resilience than its traditional form. It was proposed to investigate the embryogensis-derived model's ability to provide resilience at the decentralised cloud layer. Driven by the concept of emergent, self-organising behaviour as a result of local communication, this model removes the need to maintain routing tables or traffic by employing flooding based routing and tactical node placement.

In order to understand more about the resilience characteristics of this embryonic cloud model, a Cellular Automata (CA) based simulation will examine its ability to maintain network connectivity under varying environmental and application conditions. The results of this stage provide quantitative data which supports whether or not a cloud architecture modelled on embyronics can deliver applications resiliently. Successful delivery within hostile environments will provide support for this model being deployed to provide cloud services in domains which currently are unsuitable.

This study starts by defining the CA model and linking it to the previously defined embryonic architecture. The model is implemented in Python. It was then validated using a number of means such as graphical validation, extreme conditions and internal validity. Batches of tests were then conducted in an empirically sound manner to derive a dataset of 10 lots of each possible configuration permutations resulting in 27000 different test runs. These are described in further detail later in this chapter. This enables insight to determine the constraints of applications processing upon this architecture. A quantitative analysis of this dataset then provides insight into this question. Finally,

the results of this analysis drive further considerations for a proof-of-concept implementation.

4.2 Cellular Automata Model Simulation Description

This section describes the Cellular Automata (CA) based model for the embryonic architecture including the independent and dependent variables with motivations for their use. CA are discrete models of dynamical and complex systems. They are employed for modelling and simulating a variety of discrete scenarios of complex systems. Within this simulation a stochastic CA is employed, due to having varying factors, such as the decision to spawn, the function, to spawn and the simulated FR being dependent upon a random distribution. These stochastic CA are sometimes referred to as Probabilistic Cellular Automata(PCA) (Kephart & Chess 2003). When appending service characteristics and algorithms for service self-organisation, these can be suitable for simulation of the previously defined embryonic decentralised cloud model.

4.3 Cellular Automata Model Simulation Description

4.3.1 Definitions

The following definitions should be considered within the context of this model.

- **Cell** - representing a node within the network and the cell in the cellular automata. It also represents the biological cell within the model. It differentiates to a particular software function and will execute code of that function. Or it will be dead (state 0)
- **Function** - A logical software function e.g. cryptographic functions, web service
- **Application** - Composed of multiple functions to provide a specific service. An application's functions must be able to reach each other in a consecutive manner.
- **Update Function** - Probabilistic function which determines the state of each cell within the simulation. Executed upon each cell. Updated asynchronously and randomly to account for similarities to real world networking.
- **Multi-Cellular Organism (MC)** - a collection of multiple cells which allows 1 or more applications to be executed via functions distributed across the cells.
- **Connectedness** - the quantity of fully connected applications. I.e. those where all functions can be reached consecutively in order to fully process the data.

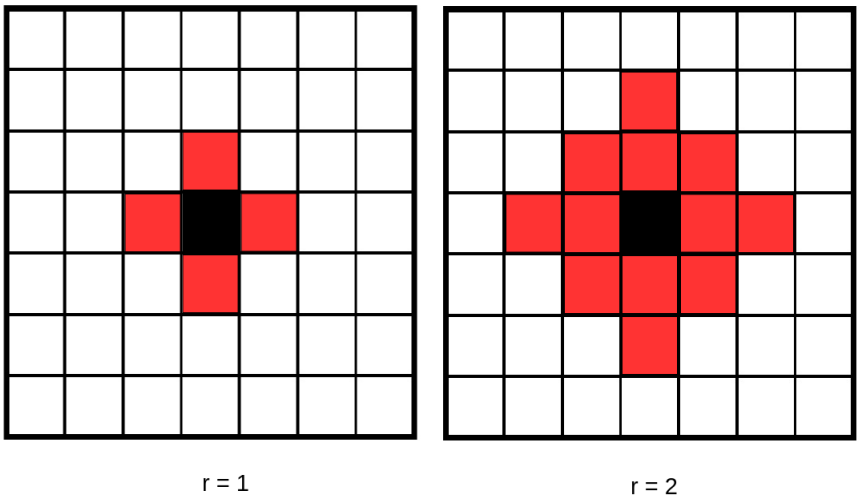


Figure 4.1: Connected0 and Connected1 modelled as von-neumann neighbourhoods r=1 and r=2 respectfully. The red squares indicate the nodes in the central squares neighbourhood, the nodes which will receive communications.

4.3.2 Variables

The purpose of the simulation is to understand more about the constrained model's ability to resiliently deliver services. The external or internal changes to the system will be represented by the failure of nodes, according to a given stochastic variable. Whilst the resilience of the system will be measured through the number of services that are fully connected at any step within the simulation. This therefore will illustrate to what degree the network is still able to deliver its service under varying changes.

The following dependent variables (connectedness values) will be used to measure this resilience.

- **Connected0** - % of fully connected services with 0 networking hops allowed
- **Connected1** - % of fully connected services with 1 networking hop allowed

Neighbourhoods are von-neumann, where r=1 is a neighbourhood size of 5 inclusive of the central cell and r=2 is a neighbourhood size of 13, inclusive of the central cell. Connected0 and connected1 networks are von-neumann neighbourhood sizes of r=1 and r=2 respectfully (figure 4.1). Von-neumann neighbourhoods were chosen over larger neighbourhood sizes, such as moore neighbourhoods, as it is essential to reduce the number of communications being broadcast to nodes due to the flooding routing present in the network.

Figure 4.2 illustrates a network which is successfully connected with 0 hops allowed. Whilst figure 4.3 illustrates two possibilities. The 1st red node on the left is not fully connected in 0 or 1 hop networking, whilst the network on the right is fully connected with 1 hop networking allowed.

Connectedness is calculated as follows: at each step in the simulation, all starting nodes are checked to see if they can reach the next consecutive function. This process continues until the next consecutive

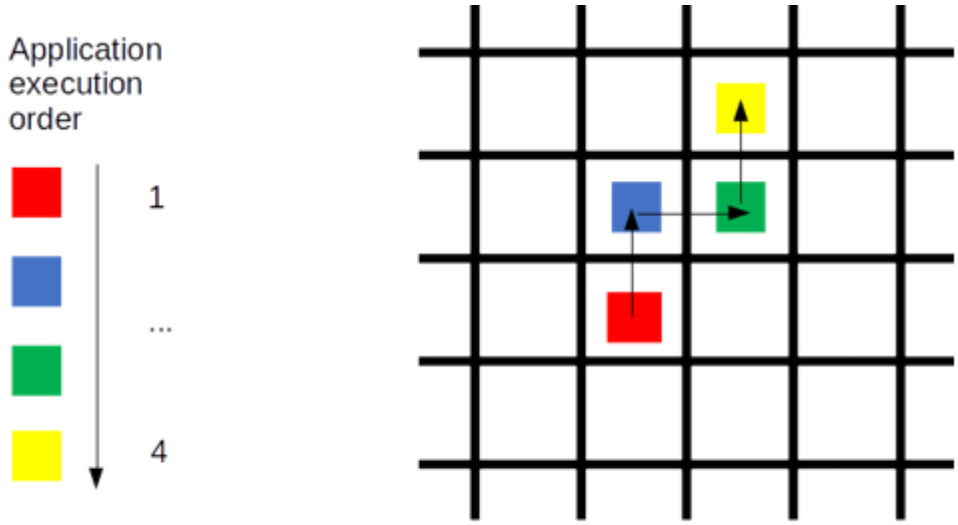


Figure 4.2: An example of a fully connected network. The arrows indicate the connection flow between nodes.

function cannot be found (i.e. no function 4 can be found after function 3), or if the final function (i.e. function 5 in a list of 5 functions) is found creating a fully connected application. The quantity of fully connected networks is then recorded for that step in the simulation. Figure 4.4 illustrates two sub networks in a test network. One where the starting node can create a fully connected network and the other where it cannot.

The following application related independent variables are used within the simulation to understand more about the self-healing characteristics:

- **Spawn Rate (SR)** - % chance a node will grow another cell
- **Failure Rate (FR)** - % chance a node will fail
- **Quantity of Functions (Q)** - the variety of different function types
- **Neighbourhood Size (N)** - quantity of adjacent nodes to each cell.

The following independent variables are employed to understand more about the complexity characteristics of the system and their effect upon the resilience of the network.

- **Starting Location** - the starting location of the MC in the grid. e.g. central, top-left, bottom-left etc.
- **Neighbour Start Size** - the initial size of the neighbourhood.

The initial cell update function is shown in Algorithm 1, as follows: firstly a random variable will decree (according to a pre-set probability of failure) if the node fails. If not the node checks its local

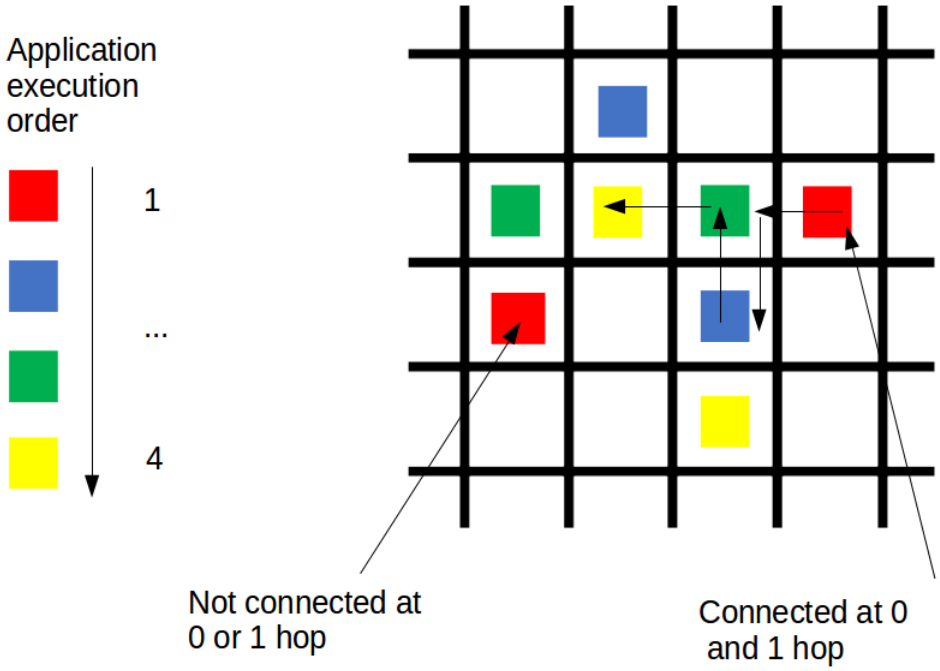


Figure 4.3: A network is connected if all nodes can communicate in consecutive order, otherwise it is unconnected. The left red node is an example of an unconnected network as it is not able to reach the next node (blue) in 0 or 1 hops. The red node on the right is a fully connected network with 1 hop allowed as it can reach its second node (blue) through communicating via the green node.

Figure 4.4: The nodes in the red box are examples of a network which is not fully connected within the connected0 tests. The two starting nodes (node 1) can reach a single node 3 using local only communication (i.e. via a node 2) but not to the final required function (node 5). In contrast the networking with a starting node highlighted in the green box can reach up to node 5.

neighbourhood. If a node is found to contain the same function as the current node, then the current node will differentiate according to an under-represented function.

Algorithm 1: Cell Update - No differentiation Optimisation

```
Data: NeighbourhoodState, CellState, Functions
Result: CellState, DivideCellState
Check Neighbourhood States;
if Cell is alive then
    if Cell does not die then
        if CellState is in NeighbourhoodState and N > 4 then
            CellState = random from Functions not in NeighbourhoodState;
        end
        if Cell should divide then
            Check Functions in NeighbourhoodState;
            for Function in Functions do
                if Function not in NeighbourhoodState then
                    DivideCellState = Function;
                else
                    DivideCellState = random from Functions;
                end
            end
            Divide with DivideCellState;
        end
    else
        CellState = dead;
    end
end
```

4.4 Simulation Model

This section presents the simulation of the CA-based model presented in the previous section. Including implementation characteristics, validation and results. The model was implemented in the Python scripting language using test driven development and executed on an i5 Linux system with 4GB of RAM. The grid of cells was represented as a 2D array, where the state of each cell is in the range 0 to function quantity.

Parameter	Description	Value
Time Steps	The quantity of discrete increments per test, chosen through experimentation	500
Grid Size	Discrete area of the simulation test. Chosen in line with similar applications and to constrain test sizes for later processing.	10x10
Nodes	Total number of possible network nodes in each test.	100
Independent Variables	The number of variables which are varied. Chosen through analysis of the embyronic model characteristics.	5
Test Runs	Number of test runs per each variable. Chosen as a middle ground between empirical consistency and experiment run time.	10
Total runs	Total number of tests which were executed	27000

Table 4.1: Simulation Parameters

4.4.1 Validation

The simulation model was validated through a number of different yet complimentary means.

4.4.1.1 Graphical Validation

The two graphical representations were used to validate the model simulation. The independent variables related to resilience could be adjusted prior to the simulation. The interface would then permit stepping through the simulation (up to the maximum 500 time steps). The graphical representations (figures 4.5 and 4.6) would then enable stepping through varying configurations. This was used to examine the system operating under varying levels of strain. For example, operating under a maximum SR and zero FR allowed easily validation that the software was representing the model correctly, as these two scenarios are easily verifiable. Conversely, using a high FR and low SR caused the system to be completely inactive. Both of these cases illustrate validation of the model through extreme condition testing.

A time series graph will display the quantity of node types at each point at the end of the simulation. For example figure 4.7 indicates a low stress network where there is minimal change in the environment (low node failures) whereas figure 4.8 indicates a network with a high degree of failures. The failed nodes (dashed line) at some points increase drastically. The self-healing ability of the system can clearly be seen through its ability to maintain an even distribution of function types and return to a nominal baseline despite this. This can be seen through the cyclic peaks. The peak representing active nodes will fall as they fail, but rise again once the system has self-healed through self-replication. Interestingly the cyclic nature of these graphs illustrate an elementary method of determining the

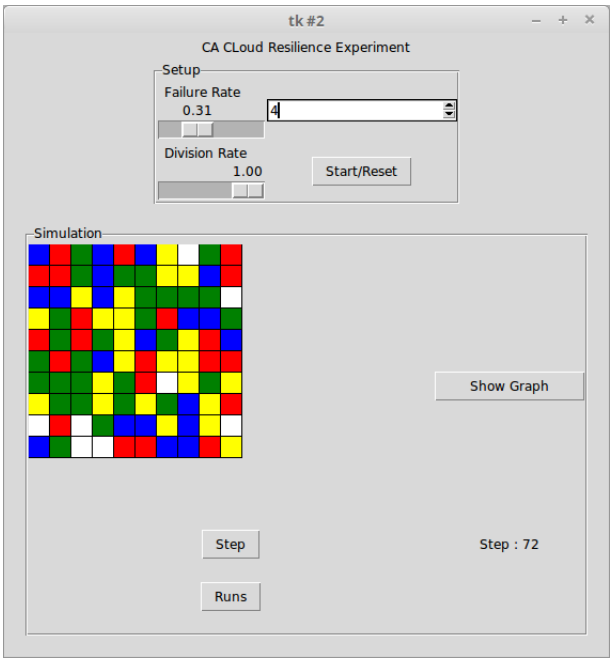


Figure 4.5: GTK GUI which permits experimentation in a graphical manner. The coloured nodes allow ease when examining the function of the networks.

```
Step: 499

Connected nets: 8
[5, 22, 23, 17, 14, 19]
[2, 0, 3, 0, 3, 1, 5, 1, 3, 5]
[5, 2, 4, 1, 5, 3, 1, 2, 0, 4]
[2, 5, 3, 2, 1, 4, 3, 4, 1, 1]
[3, 2, 1, 4, 2, 2, 5, 1, 2, 4]
[2, 1, 5, 3, 5, 1, 3, 4, 0, 2]
[5, 4, 3, 2, 3, 2, 0, 1, 2, 4]
[3, 1, 4, 5, 4, 5, 1, 5, 1, 2]
[5, 2, 3, 2, 3, 4, 2, 4, 5, 1]
[1, 3, 2, 5, 1, 5, 1, 3, 2, 4]
[2, 1, 3, 1, 5, 2, 5, 2, 1, 5]
Averages:
[4, 18, 19, 19, 18, 19]
0.512479389909
```

Figure 4.6: ASCII Graphical interface. The grid is composed of a number of cells which can be in states 0 to the number of functions. A fully connected network is one in which the functions can be reached consecutively.

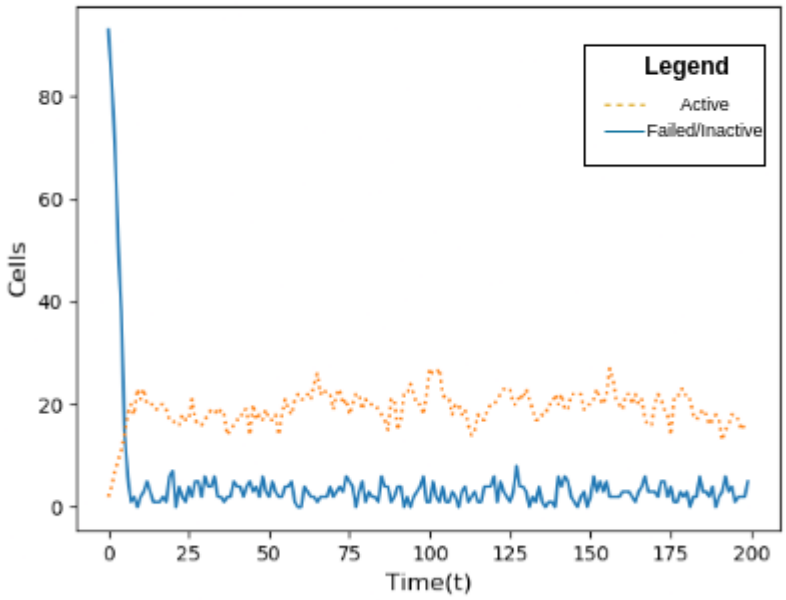


Figure 4.7: An example test run of a Low Stress Network where time(t) is a discrete time step. This test was not included in the data set and was run as an example for only 200 steps. Each line indicates quantity of functions at each point in time where blue is failed or inactive cells. The low variability illustrates the stability of the network.

state of the system. Known from complexity theory, a system will iterate through different states or attractors. In these examples, the first part of the simulation shows the network converging into a relatively baseline state. Note that at these higher FRs the majority of tests do not get to this stage and fail. Once the converged state is reached the system will cycle between performing and degraded performance as illustrated by the increase and decrease in cells. These stages, and the transitions between them, will prove useful when deriving a method for measuring resilience later in this work.

4.4.1.2 Extreme Condition Validation

Within the empirical experimentation dataset (presented in the next section) unstressed networks (0 FR) were tested. Verification occurred by examining only those tests where the $FR == 0$ and $SR == 1$, with the means of each quantity of function being approximately Grid size / Function quantity. Table 4.2 presents these values. Which illustrates that in a non hostile environment where the software is aggressively reproducing, the random selection algorithm evenly distributes the function types. However as there is no cell destruction this network will converge to a solution with minimal dynamism and informs very little about the selection process' effect upon resilience.

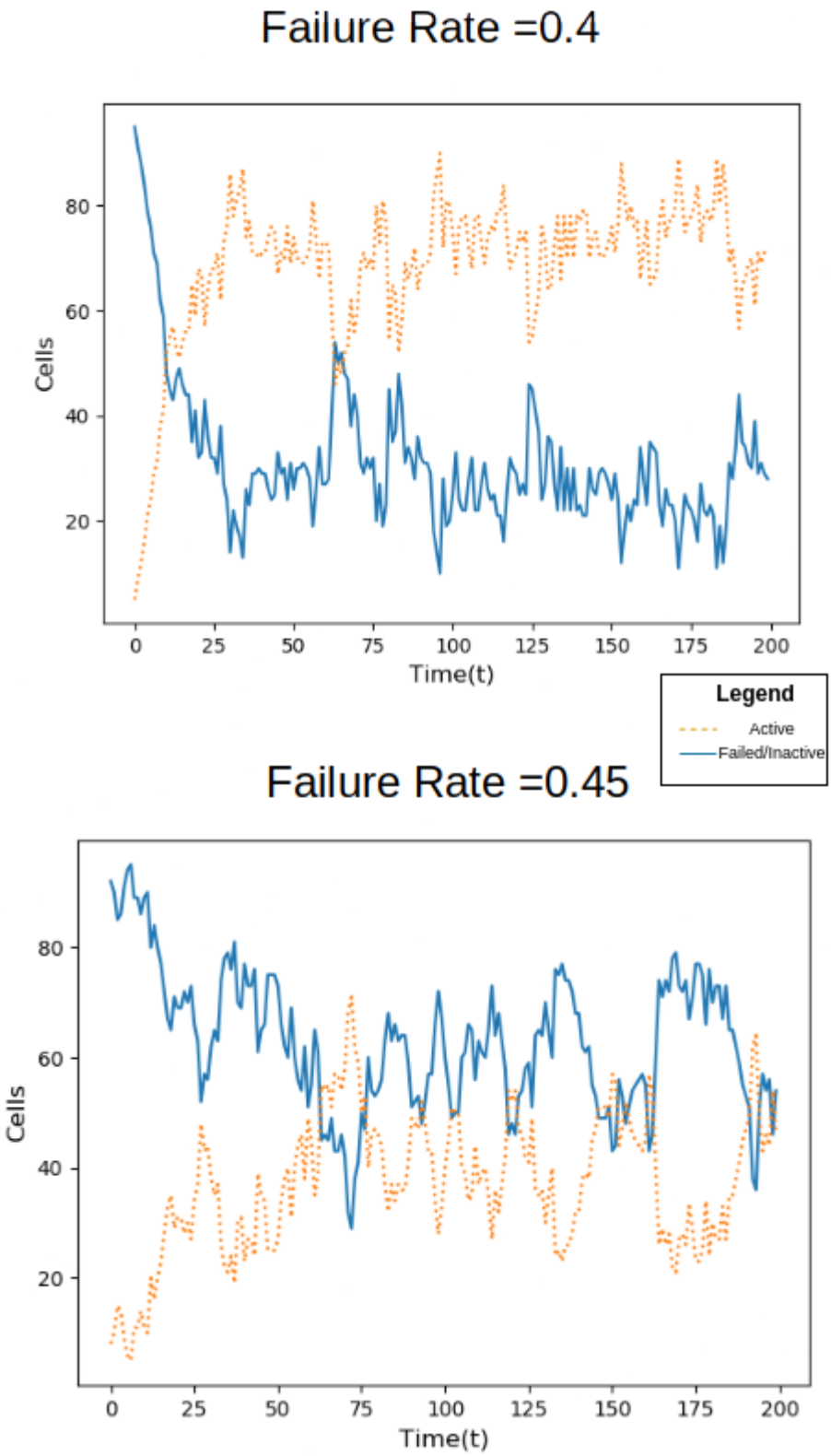


Figure 4.8: A validation run for 200 steps for 1 function where FR = 0.4 and 0.45. The periodic cycling between high-levels of nodes is an elementary representation of the system states. With high numbers of cells it is resilient, with low numbers of cells it is not resilient. Moving from high to low failed nodes illustrates the networks ability to heal after losing nodes.

Variables	Functions	NeighbourhoodStart	Connected0	Connected1
Connected0	-0.954	-0.049	1.000	1.000
Connected1	-0.896	0.096	1.000	1.000
Quantity of Node 0	0.012	-0.523	0.003	-0.086
Quantity of Node 1	-0.977	0.032	0.920	0.864
Quantity of Node 2	-0.125	-0.024	0.166	0.25
Quantity of Node 3	0.419	0.010	-0.350	-0.288
Quantity of Node 4	0.713	0.000 1	-0.697	-0.649
Quantity of Node 5	0.786	-0.006	-0.760	-0.737
Quantity of Node 6	0.651	-0.004	-0.642	-0.654

Table 4.2: Correlation Coefficients for validation via an extreme condition. FR == 0 & Spawnrate == 1

4.4.1.3 Internal Validity

The empirical experimentation consisted of 27000 simulation tests. These consisted of 10 runs of all independent variable permutations. The averages were then taken and due to the low variability and large amount of consistency (presented in the next section), these results seek to validate the simulation model.

4.4.2 Results and Analysis

The purpose of the simulation was to verify that the constrained system with no real routing could remain resilient i.e. continue to deliver services under varying network changes. As the services (applications) will successfully execute when all functions of each application are fully connected, the goal is to examine the quantity of connected applications at each time step. This connectedness of each network is then compared against the independent variables (application and complexity characteristics) mentioned in the previous section.

4.4.2.1 Stochastic Model of Best-Case Function Distribution

Prior to the CA results, this section presents a theoretical analysis to examine the effect of cell distribution on the connectedness of the applications where functions are evenly distributed. This provides a baseline ideal result with which to compare the CA results.

In an ideal situation where the cells would differentiate in a manner which left the functions evenly distributed the following applies. If $q <= 4$ then there is $> 100\%$ chance of a neighbour node being in a required state. As q increases, this value decays at an exponential rate (figure 4.9). Therefore in a less than ideal situation where nodes will fail and there is a chance that one or more cells are non-functional then this will decrease further by the FR. Figure 4.10 illustrates the probability for

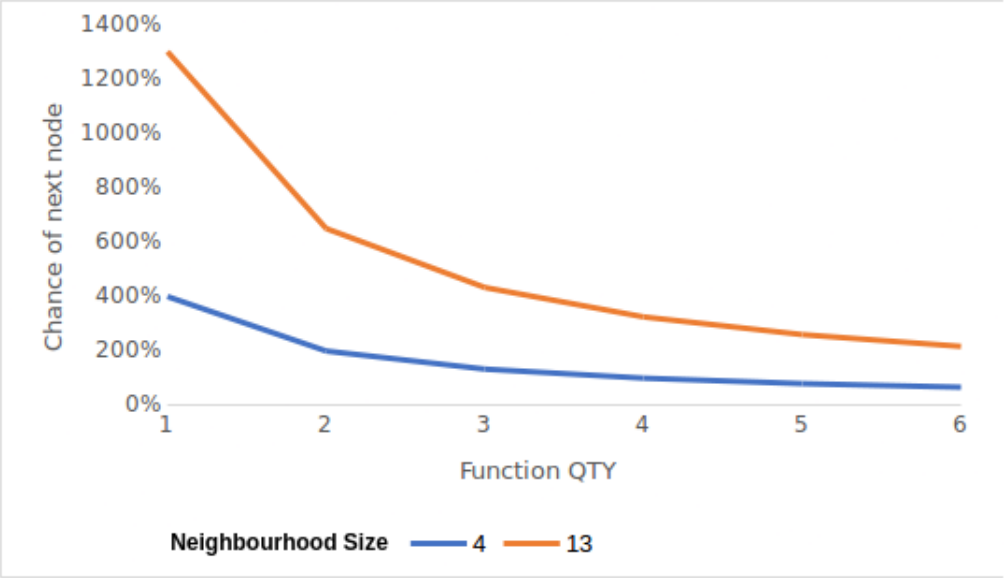


Figure 4.9: This graph illustrates the probability of finding the next node for neighbourhood sizes of 5 and 13. The decline indicates that connection issues relating to function quantity can be mitigated through increasing the neighbourhood size.

finding a next node where $q = 5$ and $q = 6$. Whilst figure 4.11 illustrates this with FRs included. The central node in all diagrams must reach the next function (consecutively numbered node) in order for the application to remain connected and execute.

A primary method of optimisation is therefore the ability to differentiate in a way which causes the needed functions to be located in the next hop, particularly where $q > n$. Following on from the previous example where all function types are evenly distributed across the map through intelligent differentiation. The following section presents a model that assumes this scenario.

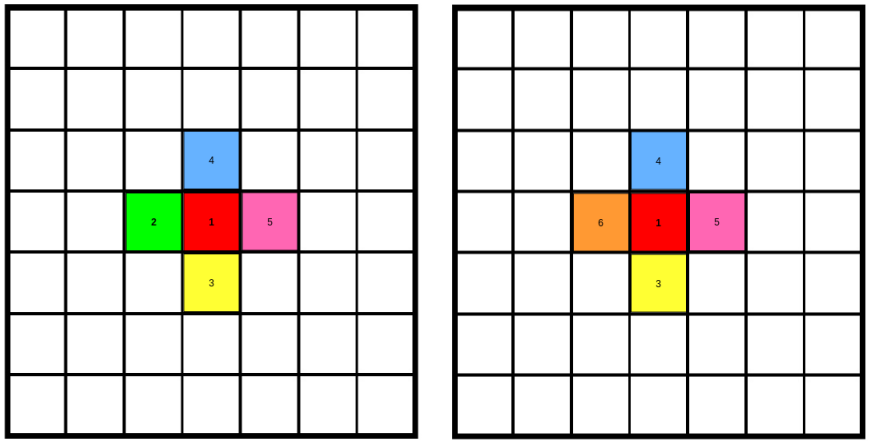


Figure 4.10: Illustrated probability of finding the next cell where r=0 and q=5 or q=6. The numbers on the nodes indicate the function type. The central node must reach the next consecutively numbered node in order for the application to connect. In the diagram on the left, n=q, so the probability of finding that node is 1. Where 2 can be seen adjacent to to the central node. However in the diagram on the right, as $q > n$ there is only an 80% chance of the next node being adjacent. In this diagram the next node cannot be found.

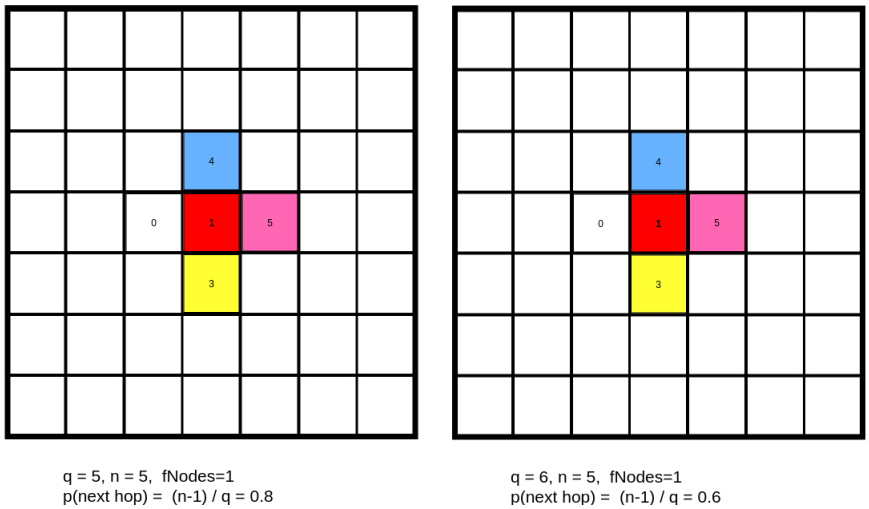


Figure 4.11: Illustrated probability of finding the next cell where r=0 and q=5 or q=6. The numbers on the nodes indicate the function type. The central node must reach the next consecutively numbered node in order for the application to connect. In both the left and right diagrams, the central node is not able to find the next consecutive node as failures have decreased the probability.

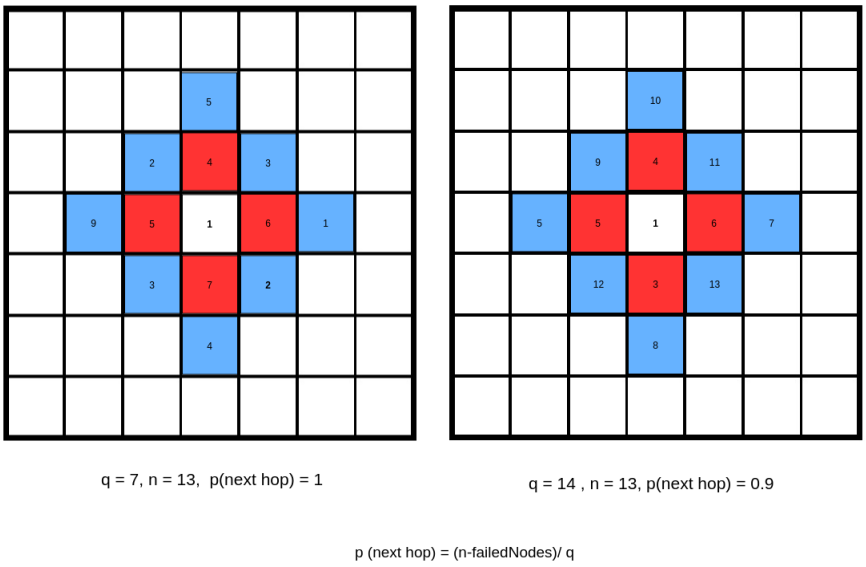


Figure 4.12: Illustrated probability of finding the next cell where r=1 and and q=7 or q=13. The inner layer of nodes (red) illustrate the first hop in the neighbourhood whilst the outer layer (blue) illustrates the next hop. All nodes can be reached by the central node. In the digram on the left, the next consecutive node can be reached as $q < n$. However in the digram on the right $q > n$, decreasing the probability of finding the correct node, causing the next consecutive node to not be found.

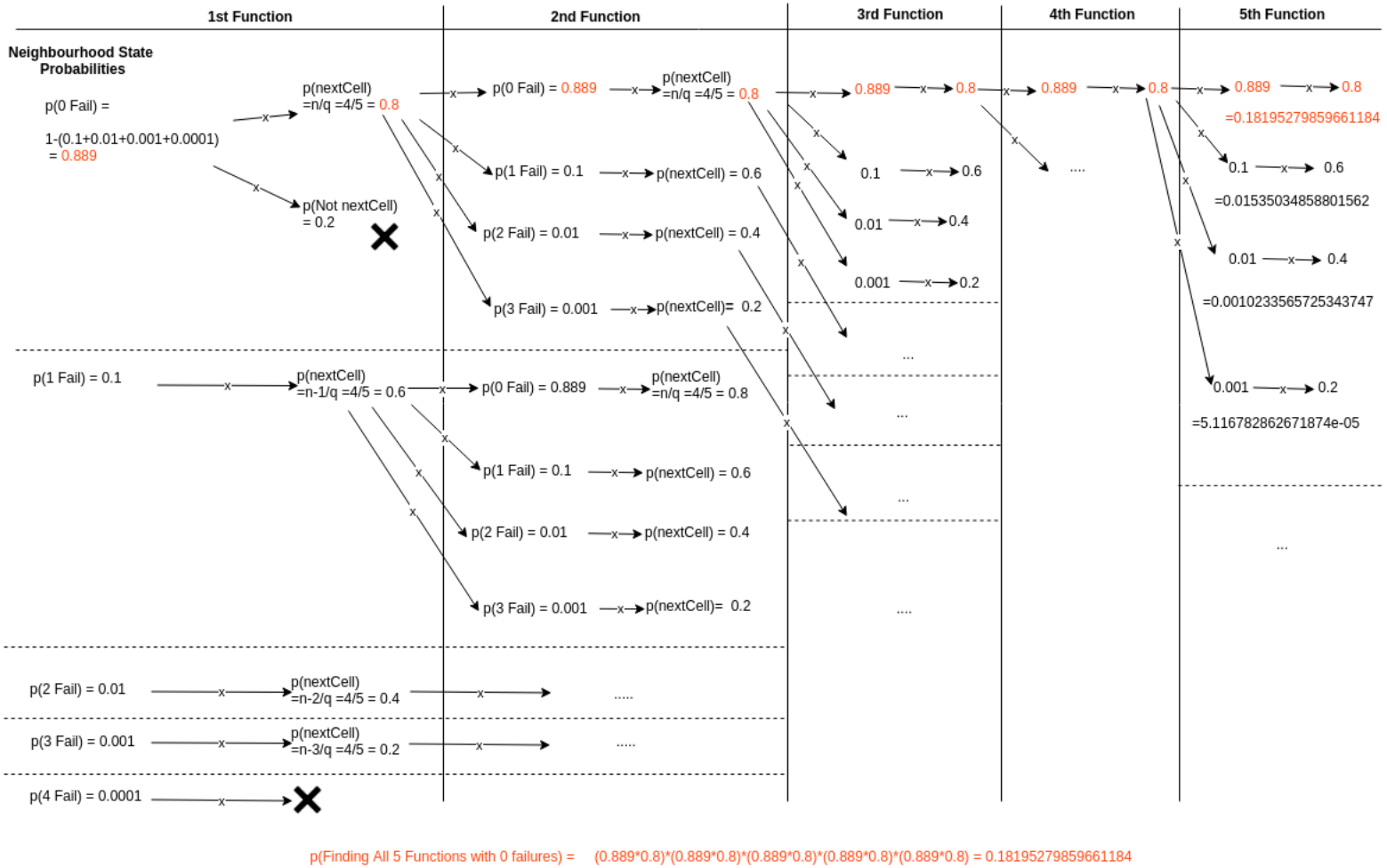


Figure 4.13: This diagram illustrates an example probability tree used to calculate the connectedness of a service where F=0.1, Q=5 and N=4. Only a number of examples are given due to the real scale of the tree being unsuitable for diagrammatic scale. Following the left most branch at all times will give the probability of finding all 5 functions with 0 failures. Highlighted in orange with the formula at the bottom of the diagram.

For values of N neighbourhood size, Q quantity of functions and F FR, the model will determine the probability of finding a fully connected application through the sum of all possible fully connected outcomes. Using a probability tree, the product of the probabilities of all occurrences where an application will remain fully connected (i.e. communication from the 1st node can reach the last) will determine the applications connectedness under varying scenario characteristics. The probability tree can be divided into distinct stages denoting whether the next function has been found.

At each stage of the tree there is the possibility between 0-N nodes will fail. As the only interesting results are those where a node finds the next function, the only branches in the tree followed are those with 1 or more active nodes. Figure 4.13 illustrates an example walk through. At each stage, all probabilities of node failures (excluding where all fail) are calculated. Each one is then multiplied by the probability of finding the next cell as discussed previously (n/q). Each one of these results spawns N new branches and the process is iterated $Q-1$ times. The results of the final stage are multiplied together to determine the overall probability.

To calculate this model the following formula can be used. Firstly we sum the probabilities of all nodes not failing (formula 4.1).

$$p(notfail) = 1 - (\sum_{n=1}^{N}(f^n)/Q \tag{4.1}$$

Then we determine the probability of the next node being found (formula 4.2).

$$p(notfail) * (N/Q) \tag{4.2}$$

This branches the tree and keeps iterating $N-1$ times where the resulting probabilities at N are multiplied. Next, all the possibilities where nodes have failed 1 to $(N-1)$ and the next node is found. As before the results where the next node is found iterate $N-1$ times where the results are multiplied (formula 4.3).

$$\prod_{n=1}^{N}(F^n) * ((N-n)/Q) \tag{4.3}$$

In order to examine the results, a recursive function was written in Python to iterate through all paths of the tree and determine the product of the probabilities. Figures 4.14 and 4.15 show the results of simulations of connected0 and connected1 simulations.

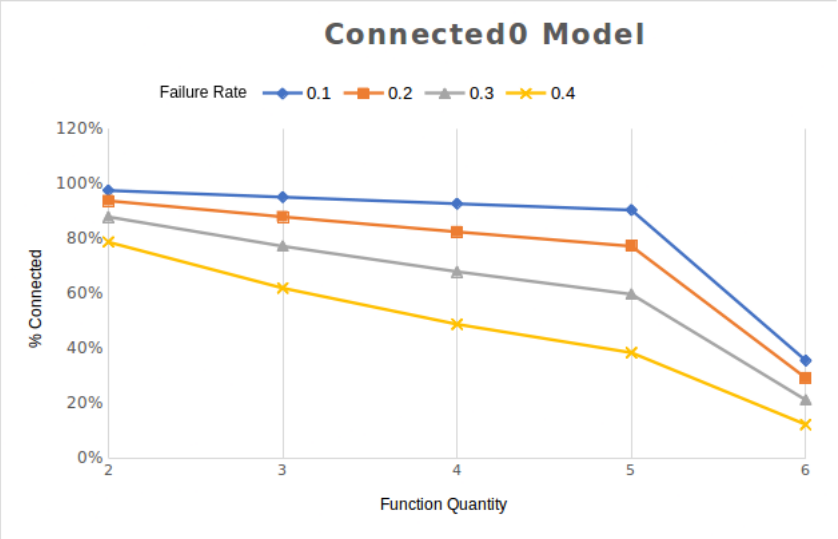


Figure 4.14: Results of the model comparing different FRs, simulating average connectedness of networks with 0 hop communication allowed where functions are evenly distributed.

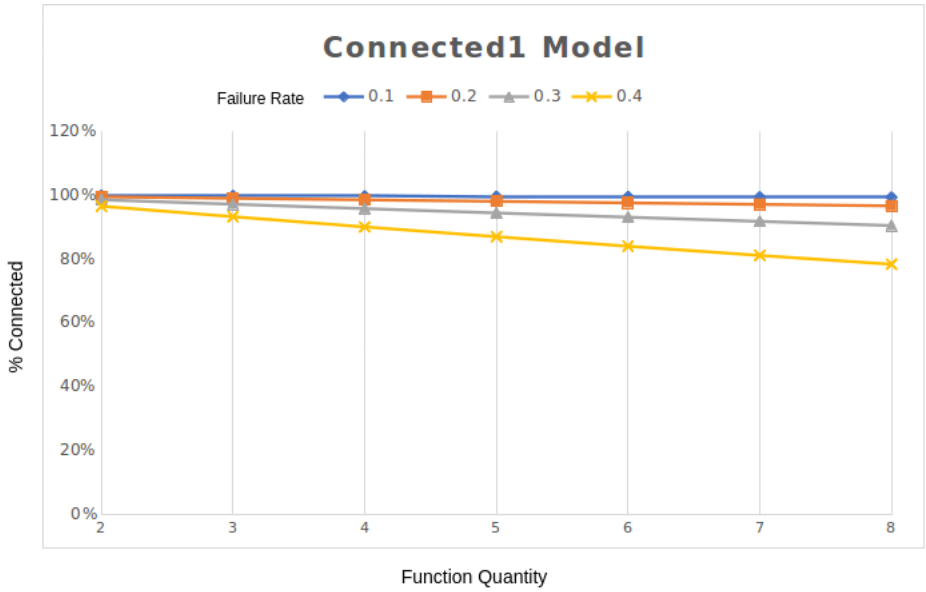


Figure 4.15: Results of the model comparing different FRs, simulating average connectedness of networks with 1 hop communication allowed where functions are evenly distributed.

The results of the connected0 model (figure 4.14) clearly illustrates the problem concerning the reduction in probability of finding the next required function due to the neighbourhood size. This is shown in the steep drop at 6 service size. Whilst the results of the connected1 tests (figure 4.15) illustrate how this problem is mitigated greatly through increasing the neighbourhood size and search space.

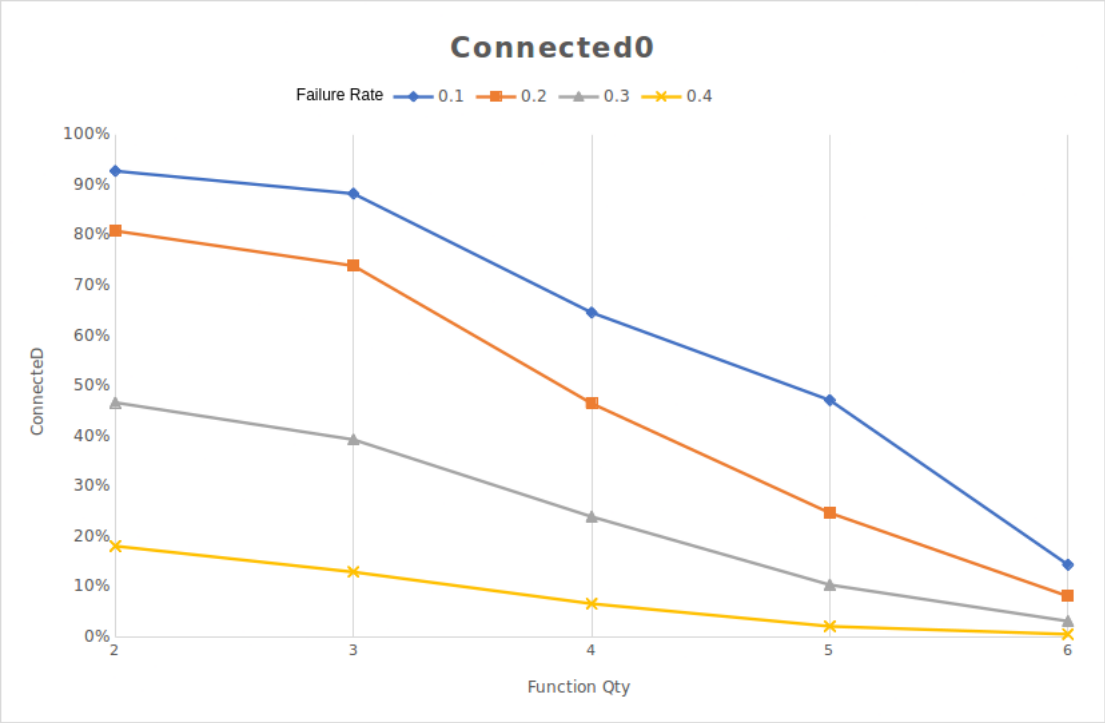


Figure 4.16: Results of average connectedness of applications with 0 hop communication allowed.

4.4.2.2 CA Results

The groups of tests examined in this section are simulated for a MC under stress, where $FR > 0$. We are seeking to determine how well the applications can still communicate using local only-interaction when under varying levels of network stress (FR) and application complexity (function types 2-6). Variables such as spawnrate and the cell differentiation process were examined in order to optimise the performance of the MC.

4.4.2.3 SpawnRate

During an initial analysis, it was shown that through aggressive spawning a performance improvement could be made. Conversely, reducing the spawnrate always caused negative performance. Figures 4.16 and 4.17 show the connected0 tests where $SR ==$ All and $SR == 1$, respectfully. The smoother declines indicates that this small optimisation can increase the ability for the applications to be connected, where higher FRs benefit more over lower FRs. This is due to the aggressive spawning increasing the likelihood of nodes filling gaps where previous nodes had failed (state 0) i.e there is a greater number of functional nodes and a reduced number of failed nodes. At lower FRs this has less of an effect due to lower quantity of failed nodes.

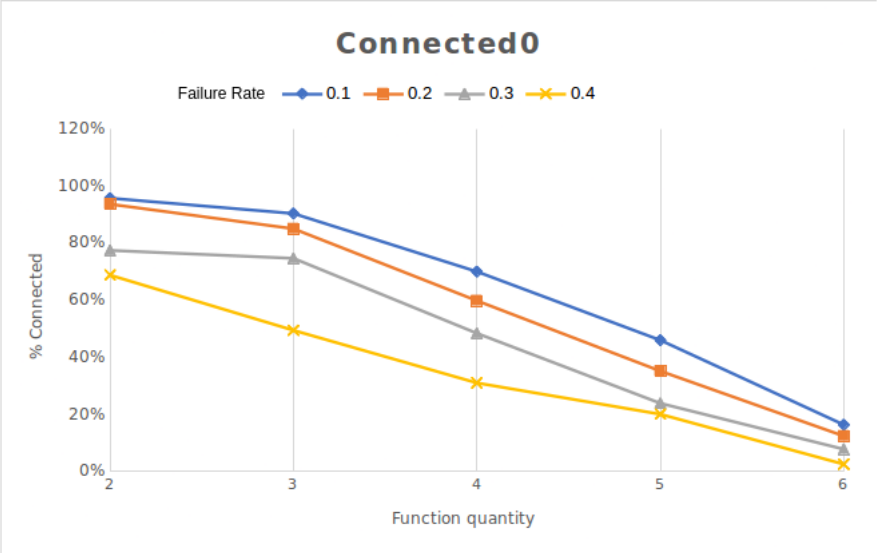


Figure 4.17: Results of average connectedness of applications with 0 hop communication allowed where spawnrate==1

4.4.2.4 Cell Update Function

Another method of performance optimisation made within the CA is the update function, which may choose to differentiate cells in order to optimise the MC. Therefore this section presents the results of 2 different groups of tests, each with 2 additional update functions. The update functions are as follows:

- **No Differentiation Optimisation (NoDif)** - Cells do not attempt to optimise the distribution of cells except to remove redundancy. The cell firstly examines the states of all adjacent cells. If its state is the same as an adjacent cell and the quantity of functions is greater than the neighbourhood size, it will attempt to differentiate to to an unseen state. The cell will still divide to a random function when there is adjacent space (Algorithm 1).
- **Differentiation upon duplication (Dif1)** - If the cell shares the same function type as a neighbour cell, it will differentiate to a function which is not represented (Algorithm 2).
- **Differentiate according to weighting (Dif2)** - As above the cell first examines the adjacent node states and determines the least represented functions out of the global set of functions. It will then differentiate to the least represented function or if there are many of the same weighting, a random function from this subset (Algorithm 3).

Algorithm 2: Cell Update - Differentiation upon Duplication

```
Data: NeighbourhoodState, CellState, Functions
Result: CellState, DivideCellState
Check Neighbourhood States;
if Cell is alive then
    if Cell should not die then
        if Cell should divide then
            Check Functions in NeighbourhoodState;
            for Function in Functions do
                if Function not in NeighbourhoodState then
                    DivideCellState is Function;
                else
                    DivideCellState is random from Functions
                end
            end
            Divide with DivideCellState;
        end
    else
        CellState is dead;
    end
end
```

Algorithm 3: Cell Update - Weighted Differentiation

```
Data: NeighbourhoodState, CellState, Functions
Result: CellState, DivideCellState
Check Neighbourhood States;
if Cell is alive then
    if Cell does not die then
        for Function in Functions do
            Count Function in NeighbourhoodState;
        end
        minFunctions = Calculate min(Function in NeighbourhoodSate);
        CellState = random(minFunctions)
        if Cell should divide then
            Check Functions in NeighbourhoodState;
            for Function in Functions do
                if Function not in NeighbourhoodState then
                    DivideCellState = Function;
                else
                    DivideCellState = random from Functions;
                end
            end
            Divide with DivideCellState;
        end
    else
        CellState = dead;
    end
end
```

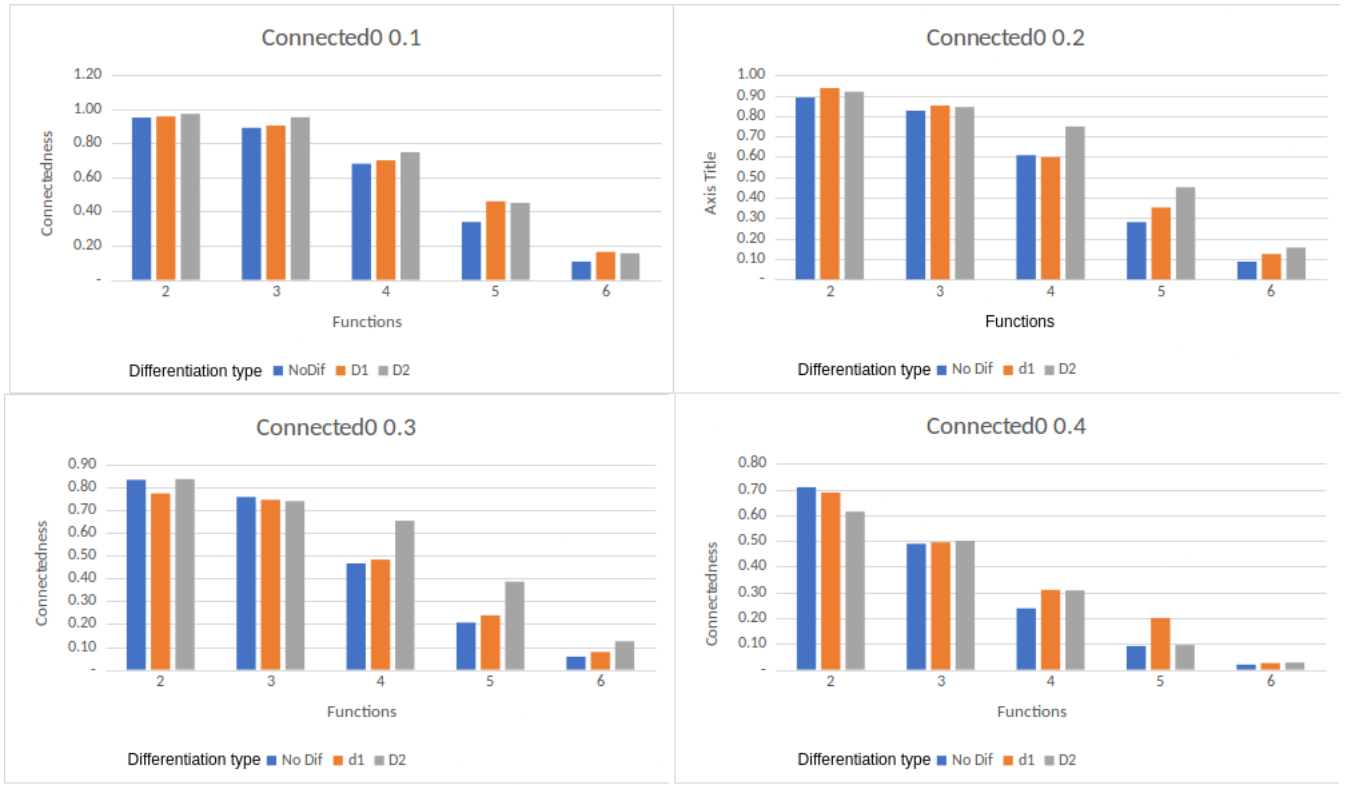


Figure 4.18: Average connectedness comparing differentiation methods for connected0 tests

Figure 4.18 presents the results of network connectedness for the connected0 tests. A clear trend can be seen that as the quantity of functions increases the ability for the applications to remain connected decreases. This is caused by the complexity of the application ensuring the likelihood of one function being adjacent to another to reduce. In the currently measured von-neumann neighbourhood where r=1, a cell will be able to communicate with 4 other cells. Therefore if $Q > 4$ the chance of finding the next node is < 100. This explains the considerable performance drop at functions 4 or greater. Regarding the difference between NoDif, Dif1 and Dif2, the trend illustrates dif2 providing increased connectivity over NoDif and Dif1 where $Q >= N$. This performance increase appears to be strong at medium FR of 0.2 and 0.3 but less favourable at 0.4. This evidences that attempting to optimise the distribution of cells does increase performance but only to a certain FR.

Figure 4.19 presents the results of network connectedness for the connected1 tests. The same trend can be seen as in connected0, however this time the decay is slower and the decline is a lot smoother, there are no sudden jumps (such as when $Q > N$). This would suggest that the issue of reduced probability is mitigated as the size of the search space is now increased. If it is assumed that the self-organisation of the MC will force differentiation of under-represented cells within the same time scale as the communication to next node, then the search space at each step will be $N * N$, else it will be $N * (N - 1)$. A considerable increase over simply N. The function distribution algorithm dif2 follows a similar performance improvement trend as with connected0 and therefore highlights the efficacy of this approach. As with the connected0 tests, high FRs and tests where $Q > N$ still have poor performance.

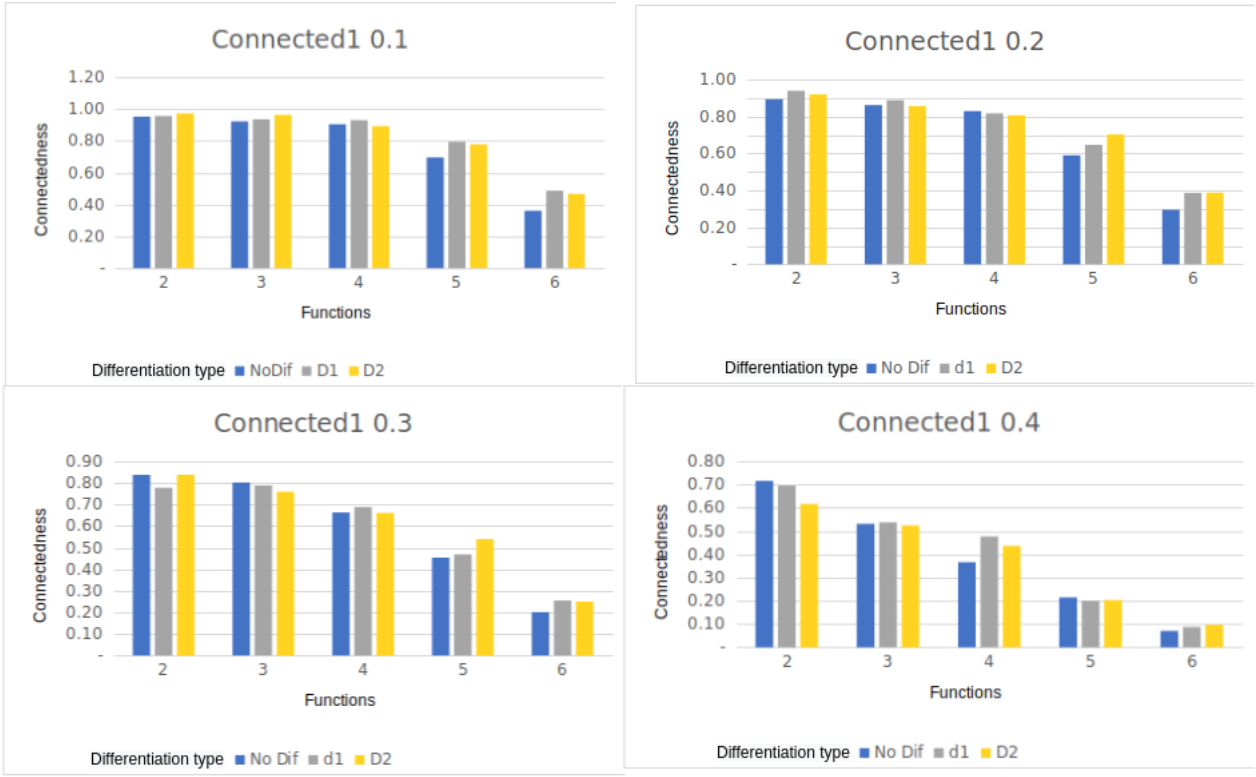


Figure 4.19: Average connectedness comparing differentiation methods for connected1 tests

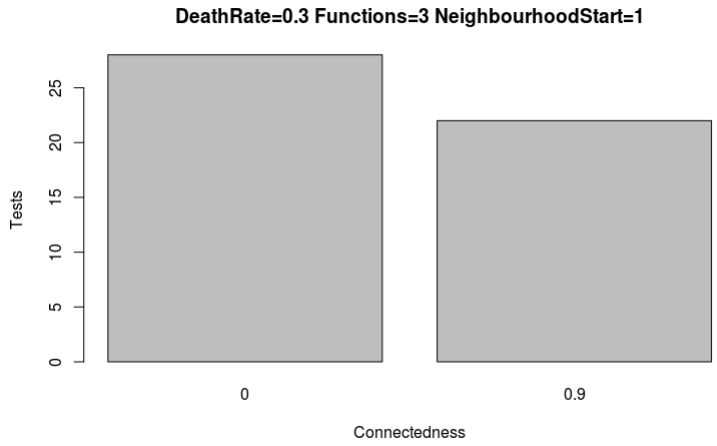


Figure 4.20: 50 test runs with FR == 0.3, functions 3 and Neighbourhood startsize 1. Showing just under 50% of tests were successful whilst the rest were unsuccessful.

4.4.2.5 Organism Start Size and Location

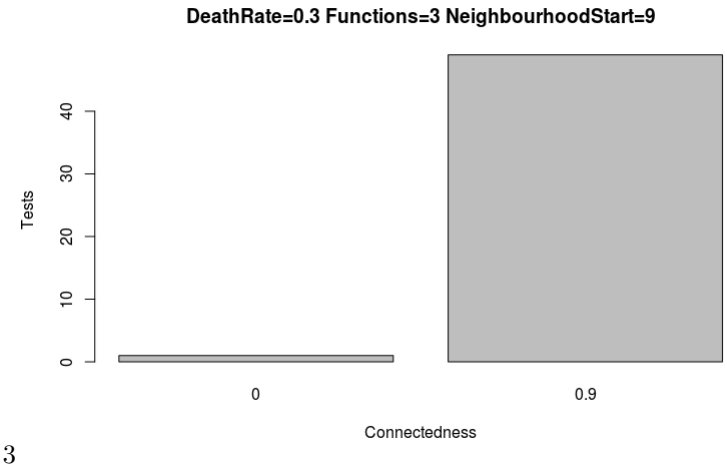
Starting location was examined for its relationship to complex systems, in case it had an effect upon the resilience of the MC. However due to a correlation of < 0.009 it was determined to have no effect.

A characteristic of the MC which is internally adjustable and is also concerned with complex systems is its starting size, i.e. the number of starting cells. Test groups were run with starting cells of 1, 5 and 9. For all tests, as start size increases, as does the connectedness, which is an intuitive finding. At high FRs, this can be quite a considerable performance increase of 50%, where $Q < N$.

Figures 4.20 and 4.21 show the distribution of tests which ended up with either an early failure or successfully survived, with the corresponding connectedness. In figure 4.20, the results for a starting MC of 1 show that a greater number of tests would fail than succeed. However figure 4.21 shows the same distribution but for a starting neighbourhood of 9 where only a small number of tests failed. This illustrates the effect of starting size upon success.

These graphs highlight some interesting aspects relating to the efficacy of different intelligent dynamic differentiation algorithms. At the higher start sizes the algorithms appear to be almost on par. However where start size == 1, the results don't fit a clear trend. Closer examination of these groups of tests indicates something interesting. That the test results were polarised where they either tend to be highly connected, ($> 0.9\%$) or not at all which explains the poor performance where $FR = 0.3$ in Dif1. This is a trend that is persistent throughout the tests where those that did not fail had consistent connectedness where the performance improvement could be gained by increasing the start size.

Examining this at the highest FR of 0.4 illustrates still good performance, however this only holds true where $N > 4$. This strongly illustrates that the initial stages of an MC are vital to its ongoing survival.



3

Figure 4.21: 50 test runs with FR 0.3, functions 3 and Neighbourhood startsize 9. Where the majority of tests were highly successful.

4.5 Discussion

This section presents a discussion of the results provided in the previous section. The results of all tests have indicated a number of points concerning factors which affect the connectedness of applications, within the embryonic model. All of which highlight methods to optimise the resilience of the MC dependent upon its characteristics. The independent factors (those variables which can be internally adjustable by the MC) are: the application complexity (quantity of functions per application), the level of division aggressiveness, starting MC size and the algorithm to determine differentiation. All of which are discussed in further detail below.

4.5.1 Application Complexity

The quantity of functions within an application undoubtedly has a direct effect upon its connectedness. This is by the inherent nature of needing to connect more cells within an application, leaving more points for failure. However, dividing application functionality is a prime characteristic of the MC architecture and a requirement of resilience, in order to distribute risk. Therefore any technique which increases connectivity under increased application complexity is a positive benefit to the resilience of the embryonic architecture. A clear point is that adjusting the complexity of an application can enable it to survive during periods of particular stress although at an obvious reduction of functionality. For example, functionality could be aggregated or divided as appropriate. This can pave the way for networking algorithms that downgrade according to perceived performance.

If it is not feasible to manage complexity through reduction of functionality, applications could be divided into sub sections or "organs" following the biological model. This would cause the overall likelihood of the application to still execute successfully albeit with a largely increased resource usage.

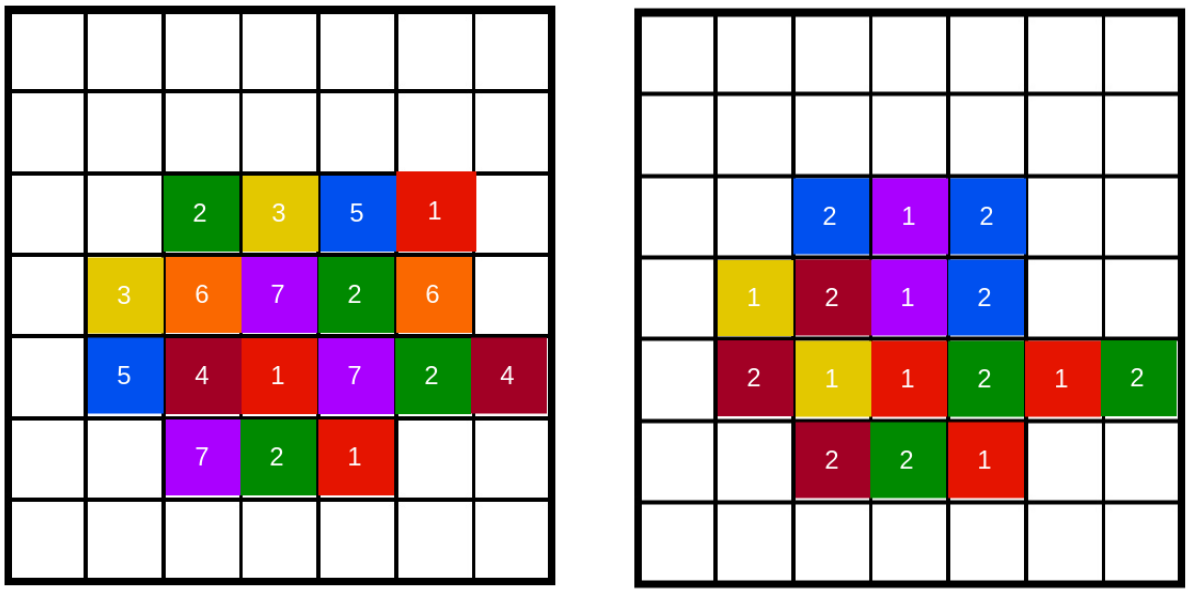


Figure 4.22: On the left is the original model where all messages are passed between cells indiscriminately. The complexity of the applications (the larger number of functions) causes the probability of finding the next function needed low. The model on the right attempts to mitigate this through grouping the functions into sub-applications, where messages will be passed between sub functions.

Whilst the application complexity would be reduced, communication protocols and network management complexity would increase due to the additional network overlay. Therefore finding an appropriate balance would be necessary.

4.5.2 Aggressive Spawning

In the previous experiments it was quickly determined that given the option, dividing to create a new cell increased the connectedness within the MC. The performance increase was considerable at higher FRs but also decreased proportionally to application complexity. This is intuitive as a greater number of cells permits a higher chance of finding the next required function. This is directly through the required function being adjacent to the cell and indirectly through enhanced communication. However, during practical testing, the time period for spawning will need to be determined appropriately so as to not delay execution of other components with the cell, e.g. function execution.

4.5.3 Complex System Characteristics

Two characteristics of complex systems were also investigated for their effect upon connectedness, the cell's initial starting size and location. The start location showed little benefit as varying it made no difference to the connectedness results with a almost zero correlation. However the starting size

correlated with an increase in connectedness, particularly at higher FRs. The reason for this is the same as the aggressive spawning point previously mentioned. This is a strong finding considering this could give networks starting in highly hostile conditions an increased chance of survival. As it was noted that networks either survived entirely or failed early on, this highlighted the necessity for ensuring strong survival in early stages, where increased MC start size is a strong enabling factor.

4.5.4 Increasing Search Space

Throughout all the results, a clear improvement can be seen with the connected1 tests over the connected0. As explained previously, this is due to increasing the potential search space through the increased neighbourhood size, which can also be understood as von-neumann networks where r=1 and r=2, respectfully. If the quantity of functions is greater than the neighbourhood size (which is decreased further by the FR) then the probability of connectedness decreases and therefore increasing this search space can mitigate this issue and thus improve the connectedness of the service. However due to the application of flooding based routing, larger sized neighbourhoods will have a detrimental effect upon MC performance. Through permitting 1 hop communication, the search space can be considerably increased and thus the likelihood of remaining connected increases. Optimisations will need be determined once more is known about the effect of communication upon the MC in later stages of the research.

Therefore, we can deduce from the previous analysis that the relationship between the neighbourhood size and function quantity (application complexity) has a direct effect upon the connectedness of that application. The probabilistic model presented in section 4.1 indicates that during a best case scenario, the most optimal selection for function quantity is any which is less than the neighbourhood size. This can be explained as follows:

If $Q > N$ then the chance of the application to remaining connected is $< 100\%$, which reduces considerably with larger service sizes. Therefore increasing the neighbourhood size, and in particular ensuring it is larger than the application size including FR, such that is $N > Q$ or $N > (Q * FR)$ will cause a considerably increased chance for the application to remain connected.

4.5.5 Function spread through differentiation algorithms

Another point of optimisation is the method used by cells to differentiate according to the needed functions. The probabilistic model illustrates a best case scenario where function types are evenly distributed. However, as a result of node failures and possible differentiation methods the CA is incredibly dynamic, causing cell states to be in a constant state of flux, potentially changing at each time step. Therefore through deriving intelligent mechanisms which differentiate to force a given spread of functions, the connectedness can be improved. This can be seen with the performance improvement between the two differentiation algorithms, particularly where $Q > N$ and at higher

FRs. However this performance does not reach that of the probabilistic model although it does help to confirm its findings. This is largely due to the highly dynamic nature of the CA which also lacks many features making it less representative of the real-world system. The next stage of this research will develop a prototype implementation to permit investigation of the model with the currently lacking communication and processing performance attributes.

4.5.6 System states

As discussed during the system validation, it is possible at this point to observe the change in system states in an elementary manner through the quantity of cell types. At this point it can be seen when the system is converge at the initial spawn and the cycle between degraded and acceptable performance, with the transitions between the two. This is useful as it provides direction for examining the state of the system externally, without the use of processing applications. An important point here is that the the converging state is distinct from the degraded state. Specifically that once cells reach a converged state they tend to persist and cycle between degraded and converged. However those that don't reach the converged state may simply die.

4.6 Considerations for Practical Implementation

In the previous chapter, modelling of the embryonic characteristics against the required functionality highlighted a disparity between goals. Whilst the networking functionality decreed an array of features such as routing tables; the resilience and complexity characteristics were at odds, due to the increase in attack surface. Therefore this chapter provided models and corresponding simulations to understand the efficacy of the embryonic model yet with the negative networking aspects removed. This enabled an investigation into the architectural structure of the model without skewing the results with complexity of the communication.

The simulations examined the effect of the independent factors (those variables which can be internally adjustable by the MC) which are: the application complexity (quantity of functions per application), the level of division aggressiveness, starting MC size and the algorithm to determine differentiation. These were varied to determine the ability for applications to remain connected under varying levels of failure. The results of simulations indicated a number of theoretical findings which benefit the development of the proposed resilient embryonic platform. Primarily they confirm that through replacing routing with flooding based, local-only communication, the proposed model can still effectively communicate in a p2p manner.

However the connectedness of an application will decrease as its complexity increases. Particularly when the quantity of functions is greater than the size of the local communication neighbourhood although increasing the size of the neighbourhood will cause increased communication overhead. There-

fore this highlights an area for focus during the practical implementation to determine the trade-off between neighbourhood size and performance. A similar point relates to the starting size of the MC. It was found that the larger the number of starting cells, the greater the chance of a MC to survive, particularly under high levels of stress.

Another optimisation method is through applying differing levels of intelligence during the cell's dynamic differentiation. This investigation indicated that it is important to distribute the function type so that the likelihood of successful communication increases, particularly for smaller neighbourhood sizes. Within the context of the practical development this creates an area of focus relating to the self-organisation of the MC platform. In the next stages this will be managed according to the user interaction and changes in the system.

The next chapter concerns further practical implementation of the proposed architecture where the findings from the simulations within this work help to drive the focus of the next investigation.

As the practical implementation will mostly increase complexity at the communication layer, a key focus of these investigations will examine performance degradation as a result of increased communication. Any optimisation methods which involved widening the scope of communication, such as the neighbourhood size and application complexity, will need to be heavily examined for their effect upon performance in tandem with the communication. Communication should increase exponentially in line with the number of communicating cells, therefore performance degradation is expected. Therefore any optimisation methods which will not involve the creation of increased communication scope will be essential to managing this. Methods such as determining differentiation decision or novel methods involving communication optimisation.

Overall the results of the CA simulations have highlighted a number of key points relevant to the next (practical) stage of the investigation:

- Structural characteristics can be varied in order to increase resilience under varying levels of stress. These will provide the baseline categories for the next stages of tests.
- Increasing structural characteristics tend to be resource intensive in all manners, so their usefulness in a practical context will be quantified when using real-world systems.
- An elementary measurement of state-space can be seen through the quantity of cell types. This may prove useful for deriving metrics in later investigations.
- Further methods of intelligent decision making relating to optimising the function spread when choosing to differentiate can be investigated in a greater manner, as the discrete nature of the CA simulation prevented this but did highlight its effectiveness.

The practical implementation will be in some ways less constrained than the CA simulation as the CA operated in discrete time steps and without communication. This ensures that decision making

can be operated on a more precise scale. However this may also mean that synchronisation issues could occur. Overall the CA simulations were focused on examining structure and architecture, whilst the practical implementation will examine communication performance and effect.

4.7 Summary

This chapter provided models and corresponding simulations to understand the efficacy of the embryonic model. The results of the CA simulations have highlighted a number of key points relevant to the next stage of the investigation: structural characteristics can be varied in order to increase resilience under varying levels of stress. Increasing structural characteristics tend to be resource intensive in all manners, so their usefulness in a practical context will be quantified when using real-world systems. Further methods of intelligent decision making relating to optimising the function spread when choosing to differentiate will be investigated in a greater manner, as the discrete nature of the CA simulation prevented this but did highlight its effectiveness.

Chapter 5

Proof-of-Concept Implementation

5.1 Introduction

In the previous chapters, embryonic development characteristics were modelled and then combined with decentralised cloud platform characteristics in an attempt to provide an architecture that delivers services in a highly resilient manner. A Cellular Automata model was then simulated to evaluate a number of the resilience characteristics. The CA simulation highlighted that the local-only communication driven architecture could deliver the proposed applications in a resilient manner, within the constraints of a number of parameters. This chapter presents the next stage of the research, in which a proof-of-concept implementation (employing full networking and software functionality) of the previously simulated architecture will enable further analysis of its resilience characteristics, driving more avenues for investigation. Two slightly different variants of the architecture are presented. The first is a user-driven variant, where the embryonic platform develops, organises and heals according to user-requests. The second is an autonomic variant, in which the embryonic platform develops and self-organises entirely autonomously depending upon starting conditions.

To achieve this goal, this chapter takes a software-engineering approach. The specification of the software is defined with a set of formal functional and non-functional requirements. Designs are then engineered from the high-level user interaction to lower-level sequence flow and data structures. Finally the solution is validated against the given requirements. The development process is a strongly test-driven iterative approach, in which individual components were built then integrated. Due to needing to adhere to all requirements two different variants are derived and validated.

5.2 Software Specification

The goal of the developed application discussed in this section is to realise a practical implementation of the embryonic model defined in the previous chapter. Data processing applications designed for

decentralised cloud disciplines (e.g. edge/fog/mobile cloud) will be executed upon the platform in a resilient manner (i.e. delivered in a trusted manner despite the presence of internal and external threats to the network). The P2P overlay network that the platform operates upon will contain self-healing and self-organising properties, driven by the characteristics of cellular differentiation and division. Finally the platform will communicate in a message-oriented manner using local-only communication as much as possible in order to maintain high-level of node churn. The proof-of-concept will be leveraged as an investigation platform to provide further empirical investigation of the embryonic architecture.

5.2.1 Software Requirements

This section presents the formal software requirements for the platform and will be used to evaluate the success of the implementation and drive testing plans. These are split into functional and non-functional requirements, and prioritised using the MoSCoW method (The classification of whether a requirement Must, Should, Could or Won't feature in the solution) shown in table 5.1. The first point of note is that there are very few functional requirements compared to non-functional requirements. Intuitively this is by nature of the platform being autonomous whereby the user (which dictates functional requirements) drives the platform which should execute applications requested of it and provide simple billing information. All other characteristics of the platform should be abstracted away from the user such that it operates without any user-intervention. This polarisation of requirements illustrates the autonomic nature of the design and therefore the adherence to the self-organising and complex nature of the system.

5.3 High-level Software Design

This section describes the high-level functional processes of the architecture using a series of UML diagrams with descriptions. There are two architectural variants. The first is a user-driven variant which was developed closely in line with the functional cloud user requirements. The second diverges away from the traditional cloud requirements and aligns with the characteristics of a complex system due its emergent behaviour. It investigates the findings from chapter 4 concerning the differentiation optimisation.

5.3.1 Message Registry

The first important design component of the application is the messages sent and received. The previously defined application logic can operate with differing messaging types. Initially, (Appendix B Paper 2), the communication protocol was extensive, consisting of multi-hop communications, directed messages and request/response pairs. This was conceived in order to write a robust protocol which

.	**Description**	**MoSCoW**	**Rationale**
F1	Receive a sufficiently defined application and produce the correct output	M	Cloud
F2	Divide and differentiate according to external processing requests	M	Embryonic
F3	Present metering usage information per user.	S	Cloud
NF1	Scale as required to meet processing requests or fail gracefully	M	Cloud
NF2	to self-organise to meet varying internal and external conditions	M	Embryonic, Resilience
NF3	to self-heal through division when nodes are lost	M	Embryonic, Resilience
NF4	Employ cellular signalling based communication methods	M	Embryonic
NF5	Provide a specified level of QoS	C	Cloud
NF6	Provide portability to diverse hardware/architecture/platforms	S	Resilience
NF7	Provide data integrity and confidentiality whenever needed	S	Resilience
NF8	Enable user modification of architecture	W	Embryonic, Resilience

F=Functional, NF=Non-Functional, MoSCoW prioritisation is one of Must, Should, Could or Won't

Table 5.1: Functional and Non-functional Software Requirements using the MoSCoW classification system

provided the functionality available to process full applications in the likeness of the embryogenesis process. To more strongly adhere to the requirements of complex systems and the decentralised environment, this protocol steadily became reduced. Firstly due to the introduction of local-only communication, any multi-hop, directed messages and request/response pairs were removed. Finally any redundant messages were condensed in order to reduce bandwidth consumption due to the flooding network. The majority of functionality was reduced into the single keep alive message - OKA. This is the fundamental complex systems concept of nodes communicating locally where they will continuously gossip to their neighbours about their known information. Table 5.2 presents the finalised messages implemented in this proof-of-concept. These messages were not employed in the CA simulation as it did not employ communication characteristics.

Type	**Code**	**Function**
Organism	OKA	Keep Alive – sends known state of local nodes
Organism	OCRQ	Capacity request for a particular function
Organism	OPAR	Advertise all known peers
Function	FDPR	Application function traffic in
Function	FAOT	Application function traffic out
API	PFRQ	Request to load function
API	PAPP	Push application to MC

Table 5.2: Register of messages

5.3.2 User-driven Variant

Figure 5.1 shows the use-case diagram for the multicellular architecture. Surrounding the system boundary are 3 actors:

- **Developer** - is in charge of creating the app, compiling the data and pushing the application onto the network for execution. The developer is responsible understanding the platform application specification and validating their code before pushing it to the network. May be a human operator or autonomous device.
- **Cloud Management** - who operates and manages the underlying supporting cloud infrastructure. Providing and scaling the supporting compute, storage and networking resources as appropriate. Monitors resource usage for metering purposes. There can be one or more cloud management entities working collaboratively or in isolation.
- **End-User** - receives the data from the platform for processing / presentation. As with the developer this actor could be a human operator or an an autonomous device. The end-user and developer could also be the same entity.

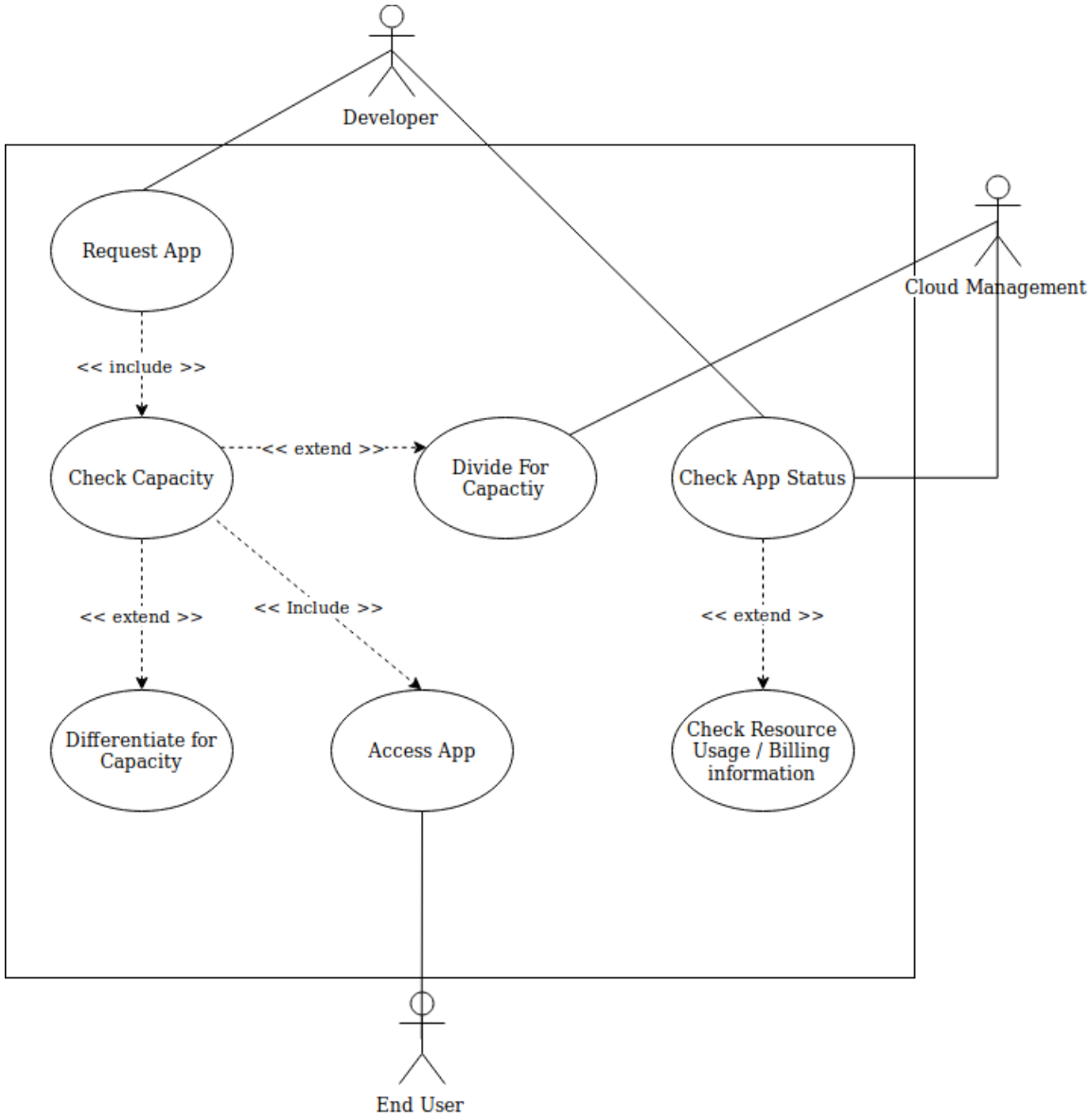


Figure 5.1: Usecase for MC architecture. It consists of 3 distinct users, the developer requests and uploads the application to the system. The Cloud management actor monitors the app status for billing purposes and also scales resources as necessary. Finally the end-user will simply access the application to push or pull data.

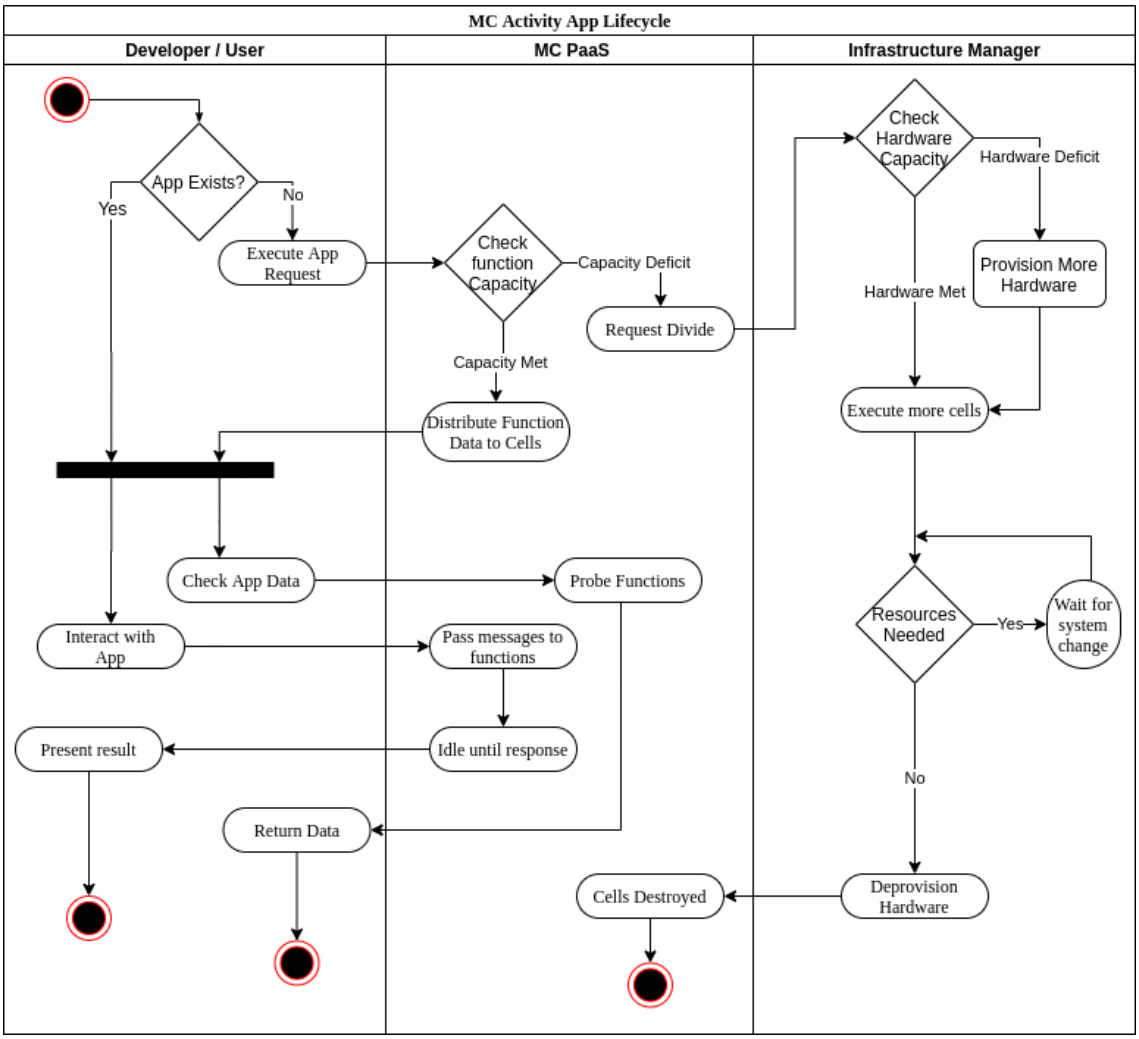


Figure 5.2: Activity Diagram for MC

Figure 5.2 illustrates the execution flow for 3 distinct actors culminating in an entire activity lifecycle diagram for an application. The external developer or user (left), the MC architecture (centre) and the Cloud Infrastructure manager (right). The diagram highlights the user-driven nature of the platform in that the user-request is what causes the MC platform to drive the resource to scale according to need. The application is executed across each function until completed at which point the result is pushed out to the user for presentation.

Figure 5.3 shows the activity diagram for the lifecycle of the cell. The leftmost entity is the superset object, the cell, which executes the two daughter entities the node (which handles network communication) and the genome (which executes function data). After initialisation, the genome differentiates to the given function and then begins a loop involving polling a socket for any appropriate messages to process. Concurrently the node will poll sockets for both the client side API and for any locally connected cells and pass the messages to the appropriate location for processing, if appropriate. Finally the cell loop will maintain an up to date list of local cells, and gossip any necessary information

to its peers.

Figure 5.4 shows the sequence diagram including function calls of the application lifecycle. It starts from the developer/user who, using the MC API, first pushes a request to the MC consisting of an application template. This data structure will illustrate the functional requirements of the application where each function is individually checked for availability, prompting differentiation/division as needed. Finally the function calls with corresponding data are transmitted to the MC either in one block or distinct units with the output data presented back to the user.

Figure 5.5 finally illustrates the sequence diagram of the cell, which complements figure 5.3. Note that after the preliminary variable settings, (thresholds, differentiation etc.) that the execution occurs concurrently in 3 distinct places. The subscriber thread continuously polls local nodes for new messages and processes them if appropriate, the genome continuously polls for new messages and executes them if appropriate, whilst the cell provides a continuous loop which gossips about local knowledge and prunes nodes as appropriate (providing self-healing functionality).

5.3.3 Self-organised Variant

Following on from the analysis of the user-driven test-case, this test-case takes a different approach. In contrast to the user-driven variant, which required a user to send an application template which drove the development of the MC platform, in this instance the cells receive information form their mother (DNA) when they are first created. This information instructs them to divide according to the range of functions that it could differentiate to. Nodes will then self-organise according to a global requirement and their local information.

The difference in functionality is as follows:

- In the first architectural design the user requested function capacity and the local cell organised the division and differentiation of this capacity local to the user. The required application types are now built into the cell upon spawn by it's mother cell.
- Having the functionality built in allows cells to differentiate according to local information to ensure an even and dynamic distribution of functionality.
- Cells now divide to fill out their capacity and then self-heal to do so. This is encoded into the cell by the "DNA" given from the mother cell. This not only permits entirely autonomous behaviour but also allows the entire platform to be re-started with different parameters by the mother cell.

The lack of continuous user prompts reducing the user-requirements can be seen in an updated use-case diagram figure 5.6. The lower user-facing functionality highlights this architecture's stronger lean towards autonomic behaviour.

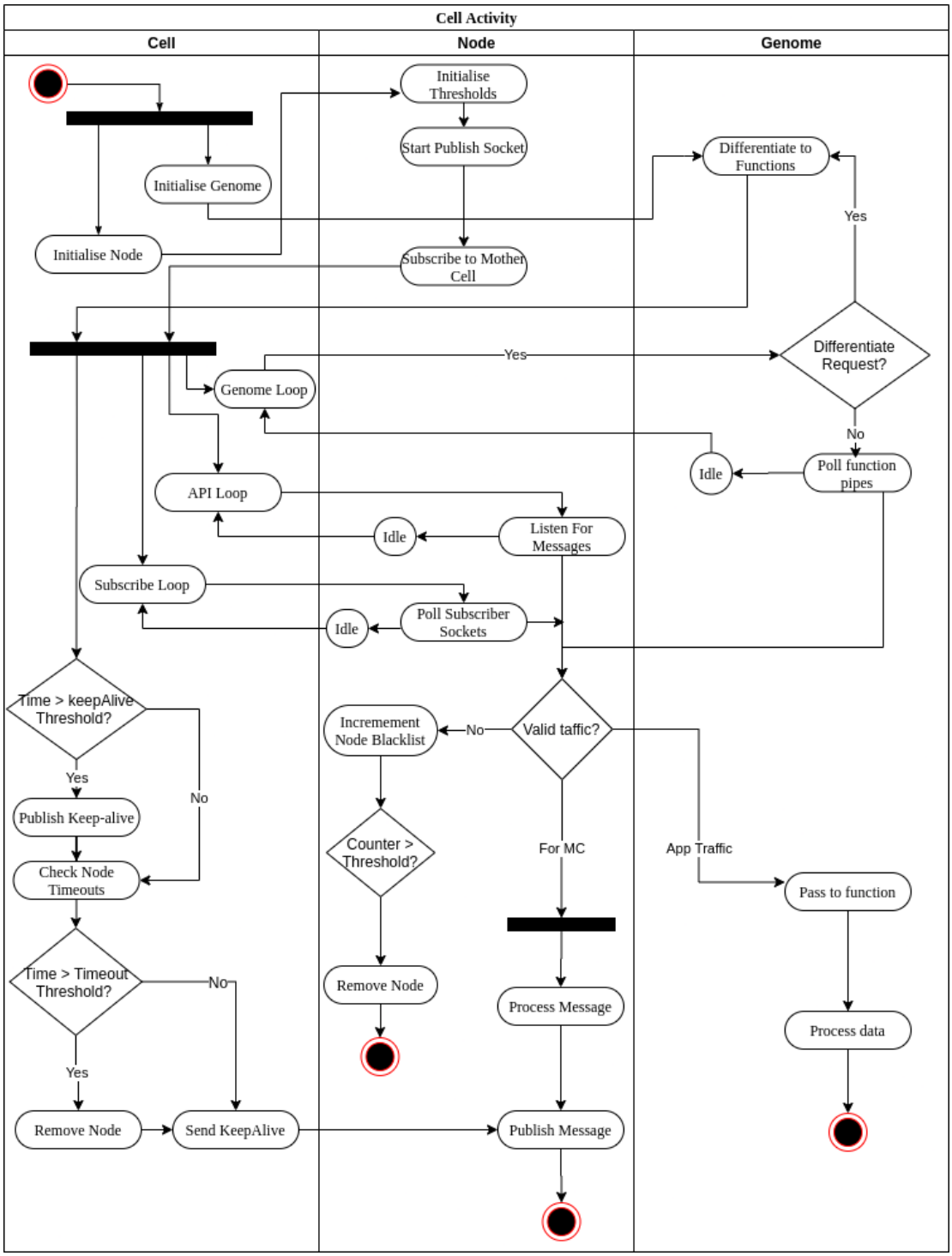


Figure 5.3: Activity Diagram for Atomic Cell

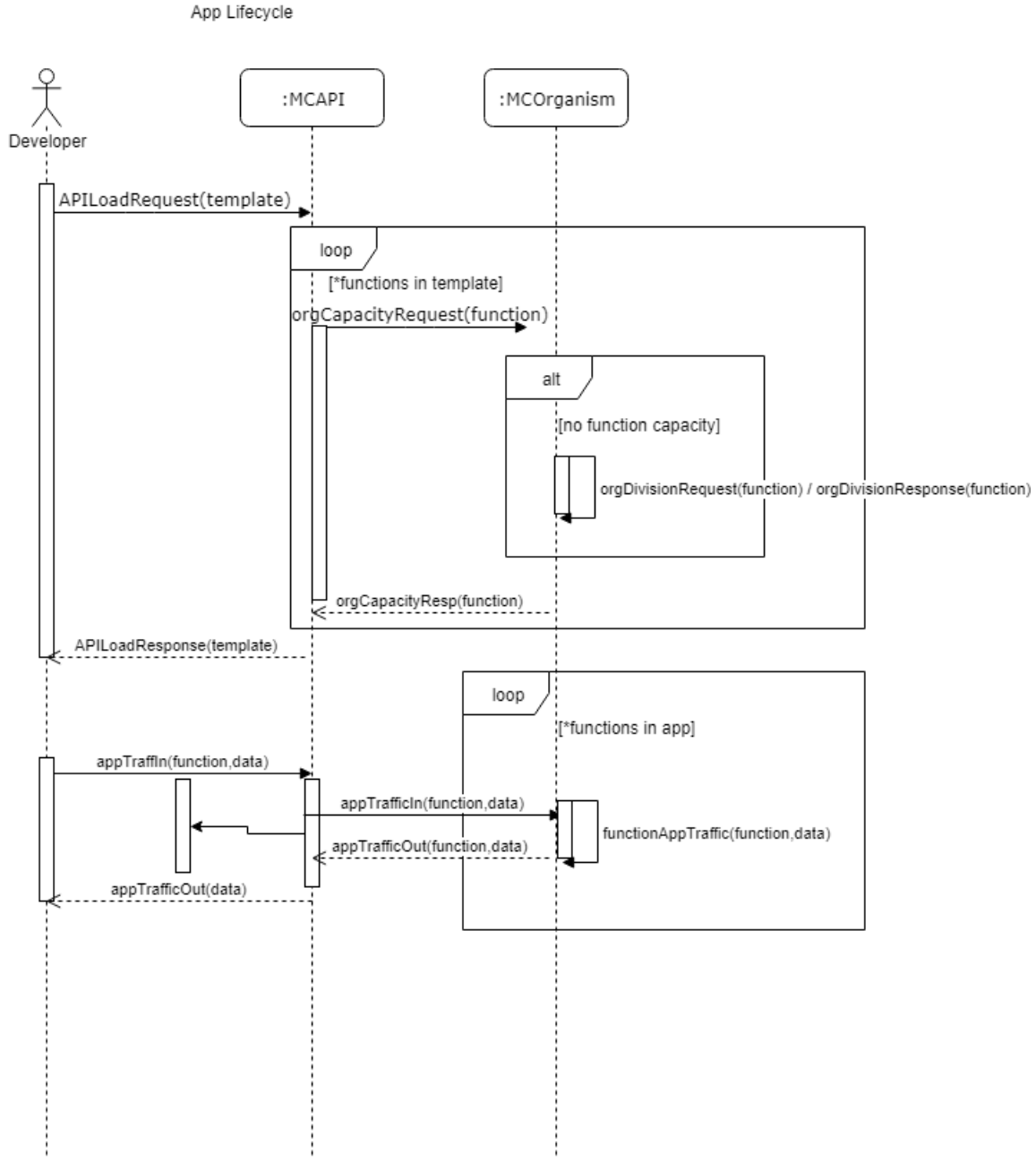


Figure 5.4: Sequence diagram for application Lifecycle

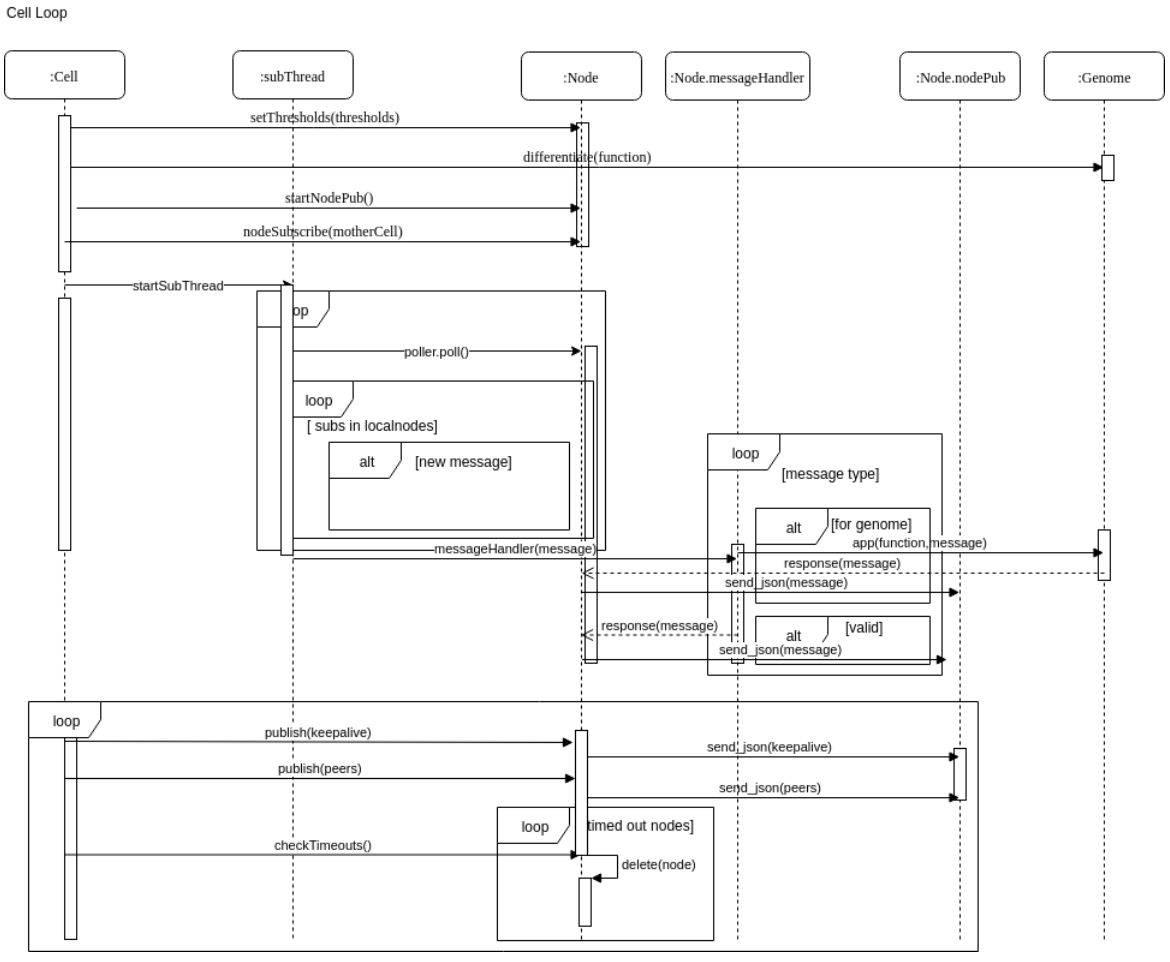


Figure 5.5: Sequence diagram for the cell

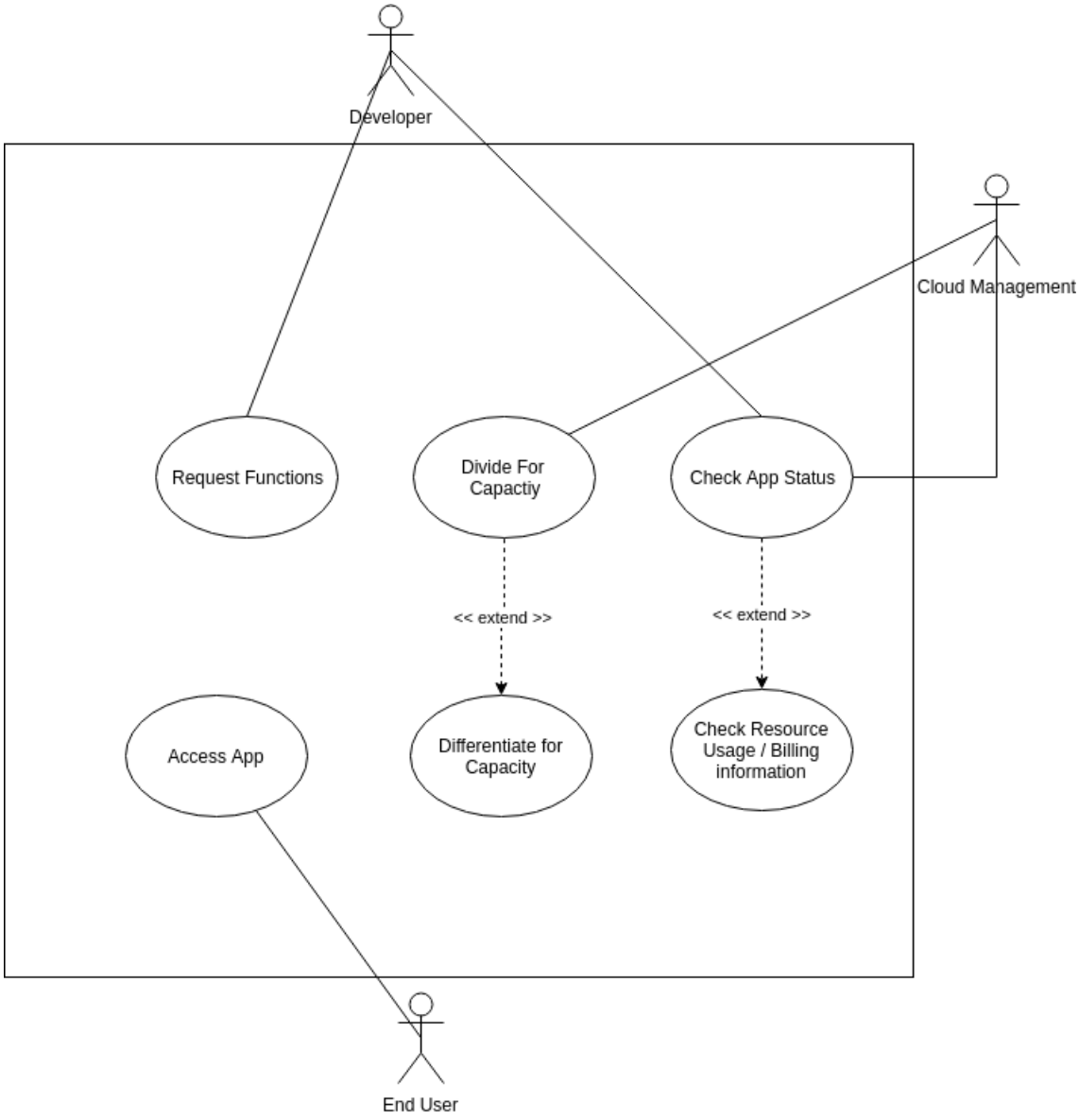


Figure 5.6: MC Use-Case Diagram Self-Organising

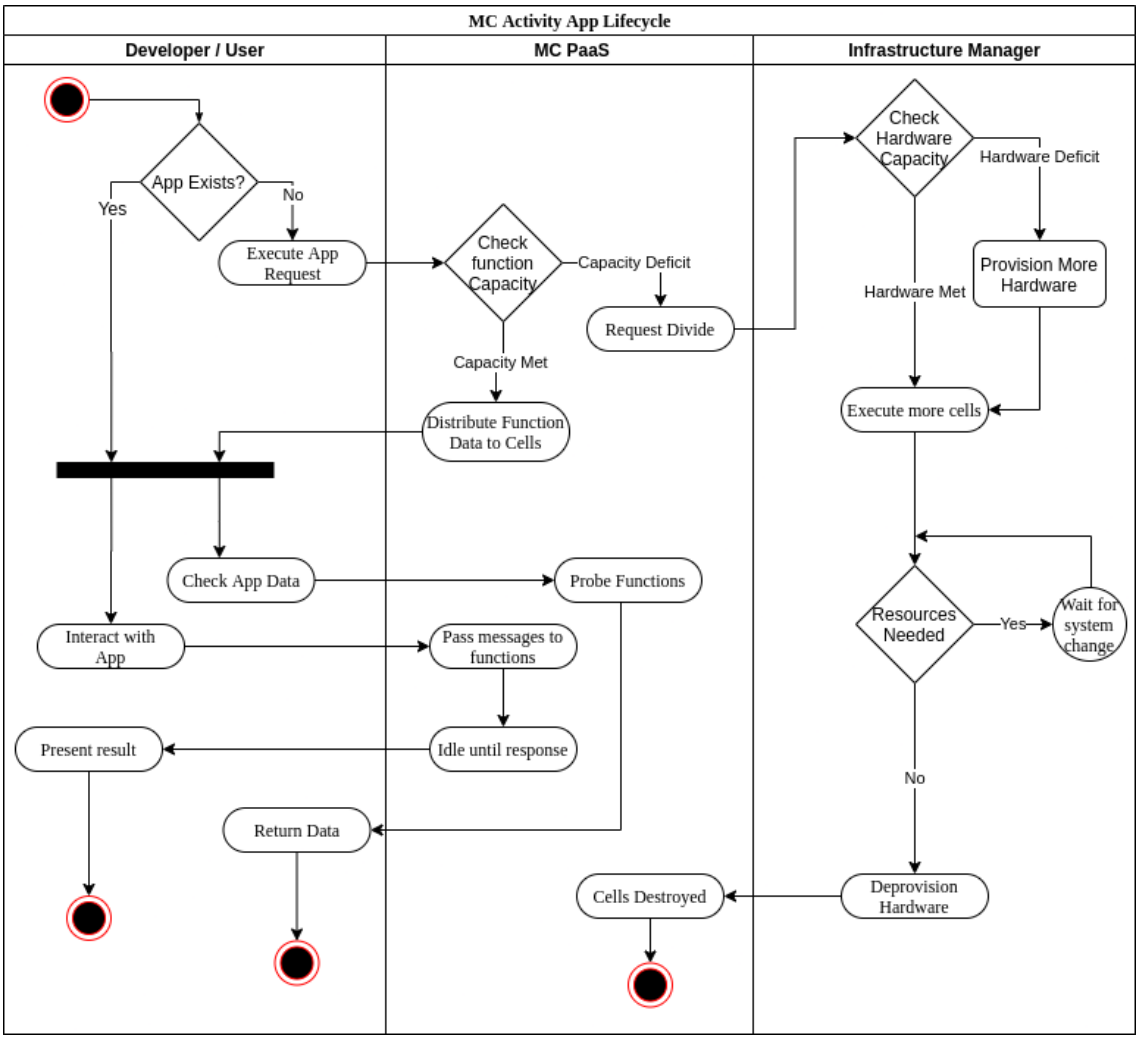


Figure 5.7: MC Activity Diagram Self-Organising

Figure 5.7 shows the minor difference between the activity life cycle, in that the developer and user instigates the cell with the template as opposed to making a request and waiting for division. Whilst figure 5.8 shows the changes for the atomic cell. Firstly after initiation it divides. Then at the end of the cell column there is an additional division if the timed out cell is a child which occurs to ensure the system does not divide past its initial constraints. Finally the differentiate request will now come via a check on the local neighbourhood as opposed to externally.

Figure 5.9 shows this autonomic functionality mostly in the cell sequence diagram. An additional loop is added at inception which divides until capacity is reached. When a timeout occurs for a cell, in this new architecture the cell will divide again if it was a child node to provide self-healing capacity. Whilst finally an additional check is performed in the main cell loop which continuously tests the local neighbourhood for even function distribution and differentiates if needed. In this validation case the optimisation uses Algorithm 2 - Cell Update Differentiation upon duplication algorithm, defined in section 4.4.2.4.

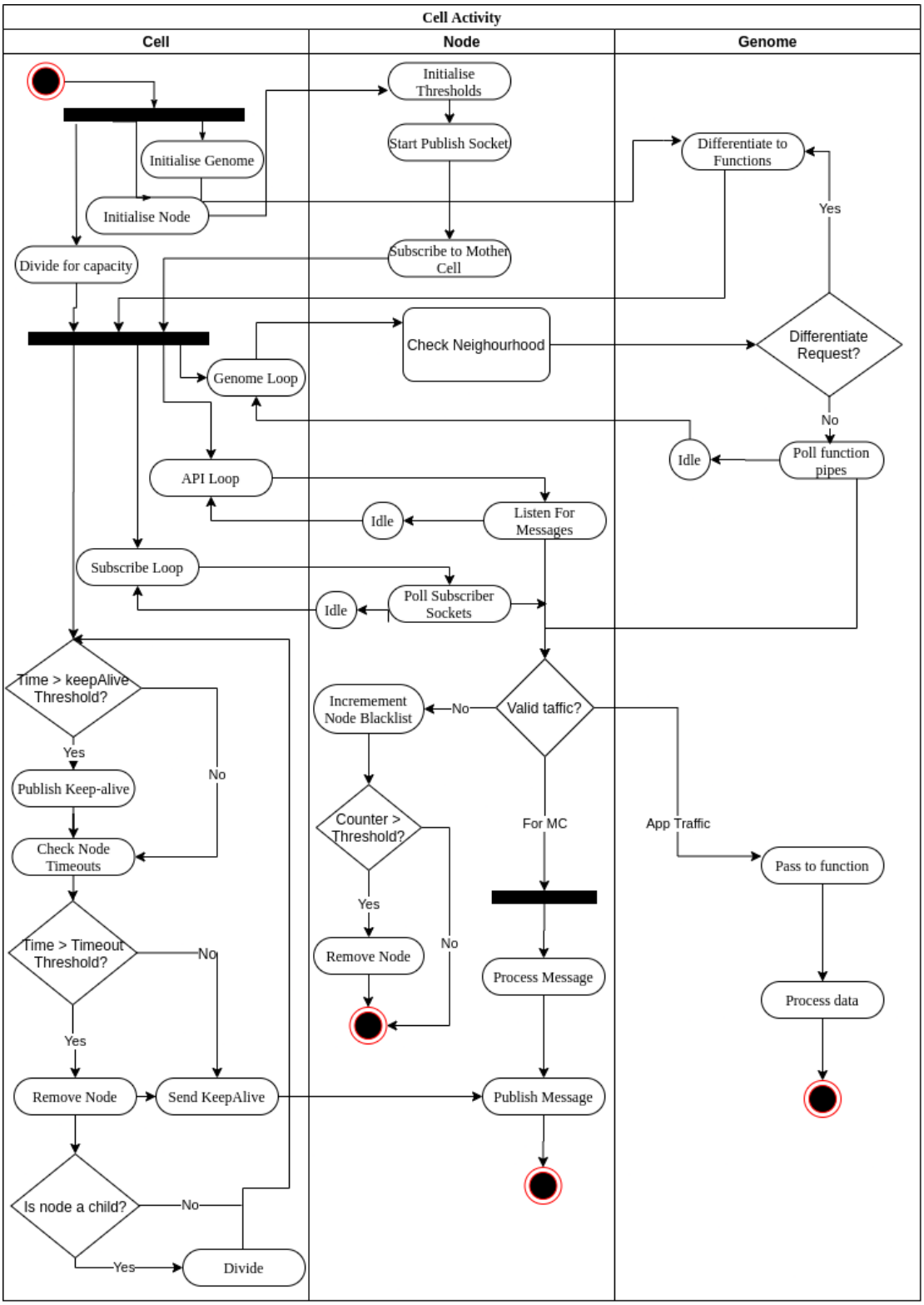


Figure 5.8: Cell Activity Diagram Self-Organising

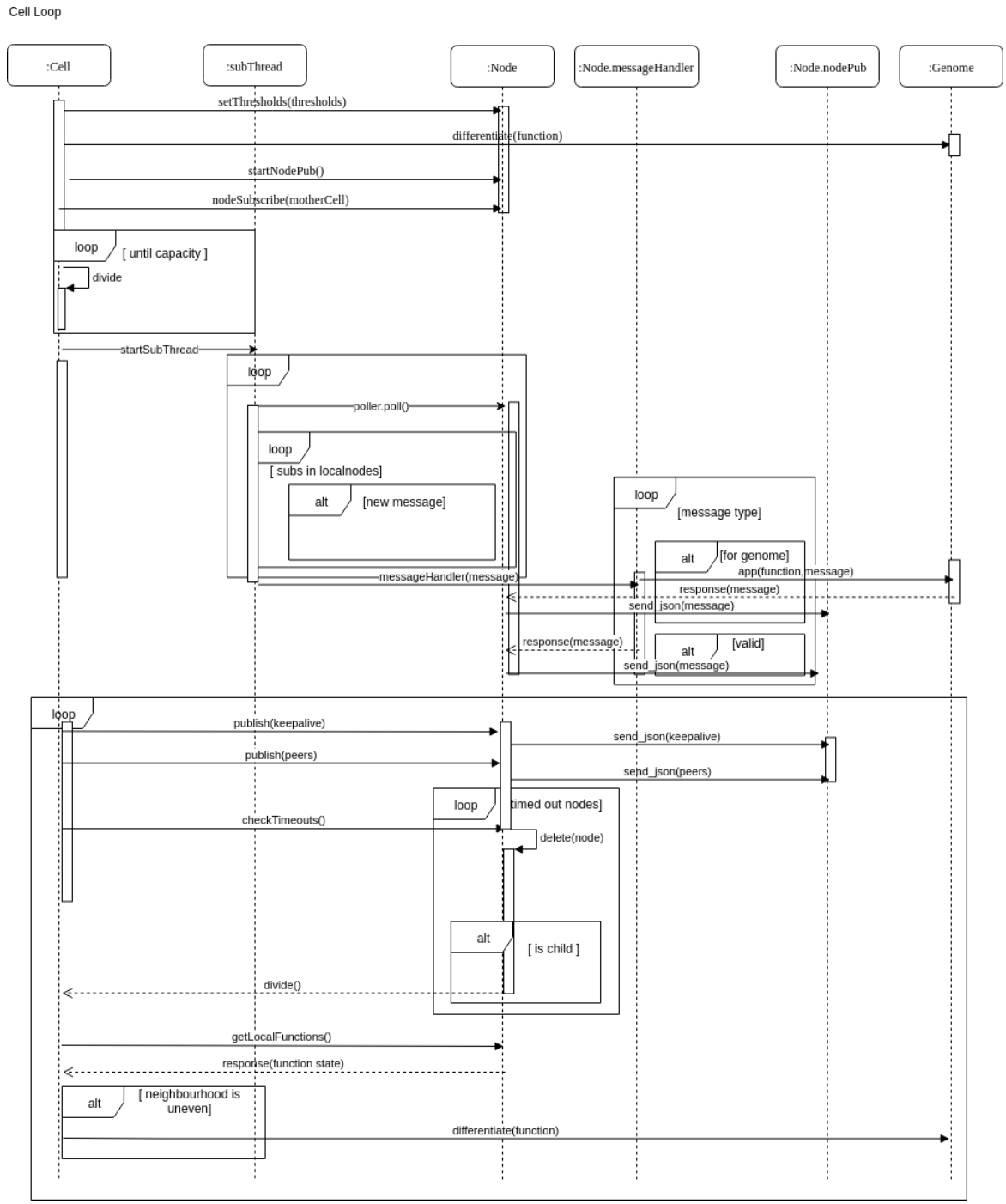


Figure 5.9: Cell Sequence Diagram Self-Organising

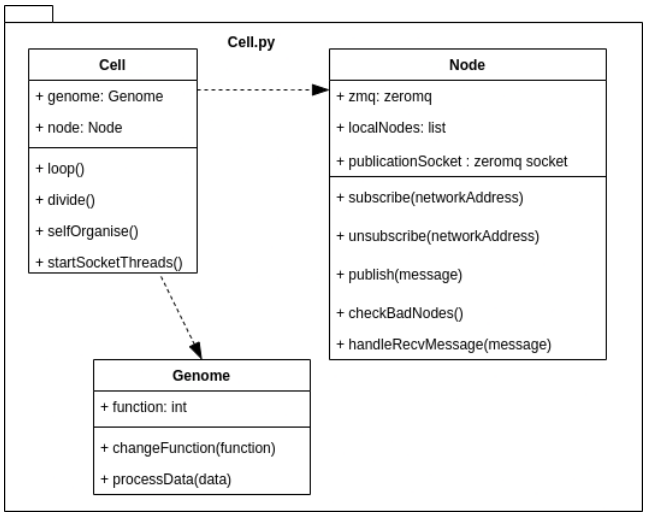


Figure 5.10: Simplified Class Diagram illustrating the relationship between the components within the python implementation of the cell. This diagram is reduced for clarity purposes.

5.4 Implementation Details

The high-level designs in the previous section enabled the practical implementation of the cell with a number of supporting applications. The embryonic software contains only one component, the cell. As such there is a singular python file named *cell.py*. This file contains 3 classes which correspond to the cell's top level components: cell, genome, and node. Figure 5.10 illustrates a simplified class diagram of the relationship between these components.

The code is heavily supported by the ZeroMQ (which stands for zero message queue) library (Hintjens 2013) which is an extremely fast and efficient high speed message passing library written in C. ZeroMQ has many advantages over coding the networking application from scratch using an operating system primitive such as Berkeley sockets (Stevens et al. 2004) (Vessey & Skinner 1990). These advantages include:

1. Messages are transmitted message-oriented not node-oriented and thus easily fits the communication paradigm modelled on cellular communication.

2. The performance in terms of message transmit speed and resource footprint enables any application to scale easily.

3. The brokerless queuing system easily permits the development of P2P architectures through the removal of a central broker system.

4. The opensource and multi-platform/language nature of the library allows the code to execute on a diverse array of hardware and software environments meeting the diversity resilience requirement.

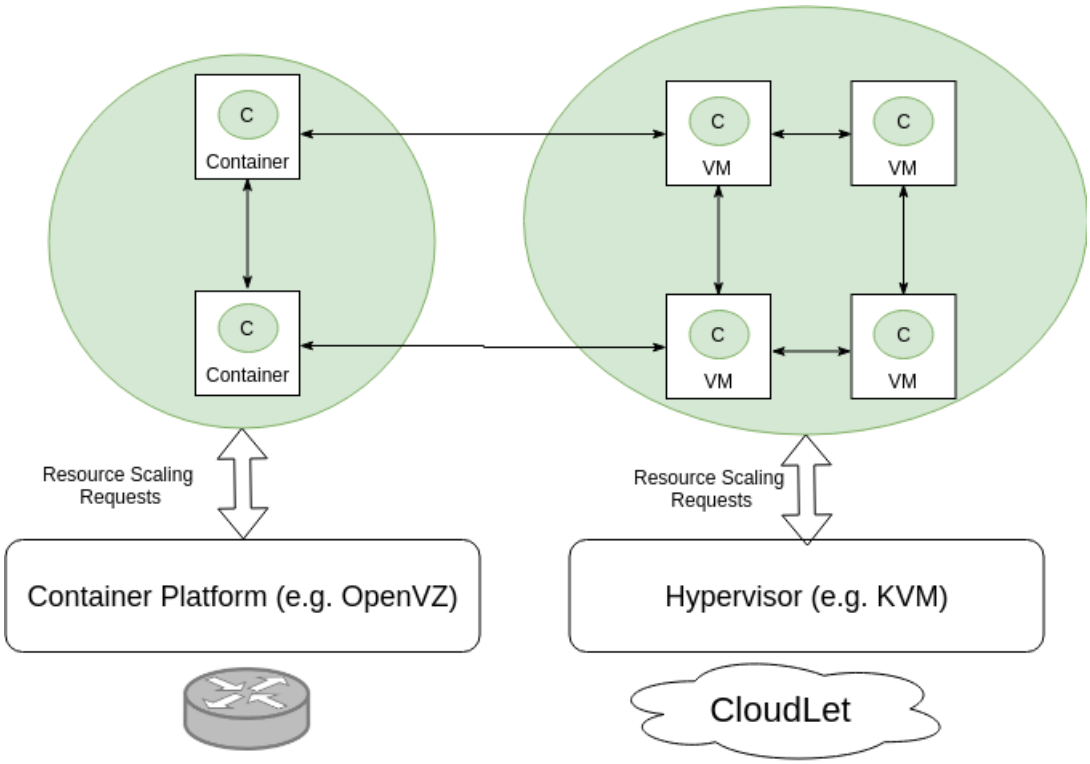


Figure 5.11: Embyronic Platform operating upon multiple different virtualisation platforms.

5. The publish/subscribe communication model permits the self-healing, apoptosis model of cellular death whereby nodes will cease communication with misbehaving nodes through simply unsubscribing.

Collectively, the ZeroMQ enabled cells will sit upon differing virtualisation platforms dependent upon the underlying node. As node diversity is wide the the cells may operate upon highly constrained embedded hardware or less constrained data-centre hardware. The virtualisation platform will be provided by the host system. Cells will communicate with the underlying platform to make requests to scale only. Figure 5.11 shows cells within corresponding virtualisation mediums operating upon diverse virtualisation platforms and devices.

5.5 Validation

After the development has been completed the application is validated to determine if it meets the requirements derived previously. This involves pushing an application to the architecture, testing for its successfully execution and instigating self-healing activities. The validation involves a visual analysis of the system at each stage in tandem with analysing messages sent and received by the cells to determine the activity occurring.

5.5.1 Experiment Test Bed

To permit the development and investigation of an arbitrary number of cells it is important to have a robust and accurate test bed environment. As cells within a network were headless and employed very fast network communication, recording all messages occurring within the system is essential. The experimentation environment consists of a full stack application which was custom developed in Python. A front-end GUI provides a dashboard of current network state. As the application was developed in a test-driven iterative method, this test bed was used to validate each individual functionality of the software up until all requirements were met. Figure 5.12 illustrates the process flow of the tools employed in this environment. As the architecture consists of singular components and a linear data flow process, it is liable to numerous single points of failure/subversion vulnerabilities, therefore it is intended for experimentation purposes and not for production use.

The environment is shown in figure 5.13 and consists of the following custom developed tools:

- **Debug Server** - Cells send all messages they send and receive from other cells, to a ZMQ socket which parses them and stores them in a MYSQL server database for later processing and analysis.
- **Events Database** - using a MYSQL server all messages from each test are stored for later retrieval. MYSQL was eventually chosen after testing multiple database formats as the quantity and rate of messages sent by nodes scaled quickly and was unmanageable by other services.
- **Web Front End** - The user then interacts with a HTML/Javascript front end. It employs Bootstrap.css for presentation, jquery for data retriveal and presentation and alchemy.js to dynamically produce graph images of networks, At the high-level, the user can view all nodes known within the database and current network structure. The user can then drill down into individual nodes and view information about each node such as subscriptions, last seen time and the updates it is transmitting to other nodes. A low-level view allows examination of individual node messages for debugging purposes, with the ability to sort by message type, or receiver, sender node.
- **RESTFul API** - is used to retrieve messages logged in the MYSQL database and present them to a web front end or any other application which may want to perform analysis on the tests.

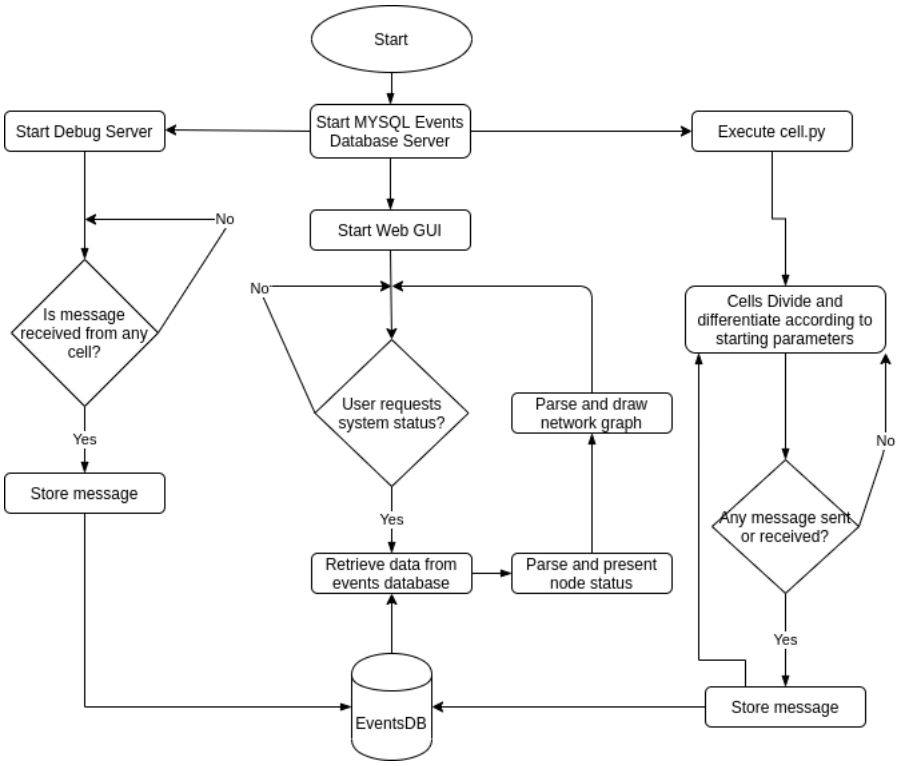


Figure 5.12: The execution flow of the tools used for experimentation and validation of the embryonic proof-of-concept implementation

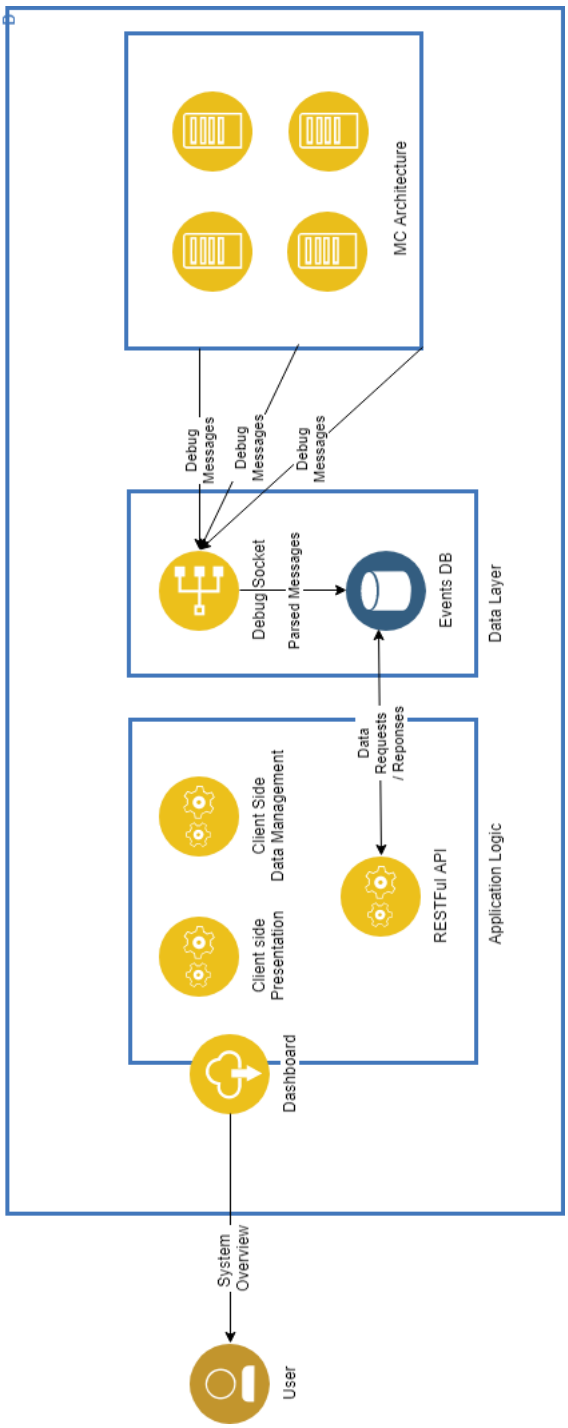
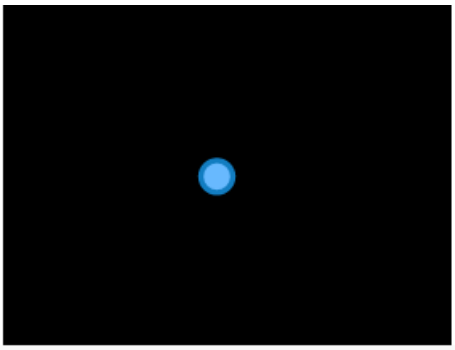


Figure 5.13: Dashboard Architecture



.4849365	localhost:19377	Output	Bound to localhost:19377
.489464	localhost:19377	pub	{"OKA": "[[], [], []]"}

Figure 5.14: User Driven Test Case Stage 1 - Initial Cell Spawn

5.6 User driven test-case

The user driven management of this platform provides an ease of requirement validation. Differing user requests are presented to the platform with the appropriate output tested against the specification. This may take a number of forms: the change in structure of the network, the ability to process the data presented and the type / quantity of messages passed. Table 5.3 illustrates an example test-case to illustrate the execution of the platform meeting a number of the requirements. Corresponding figures 5.14- 5.20 related to each step show the output from the experimentation platform network image at that stage and corresponding messages reported to the debug server which illustrate that functionality.

The steps involve a walk through of a user-driven test case of the proof-of-concept. It illustrates the ability for one cell to differentiate, divide and request further divisions in order to meet the capacity to process an application as requested by an external user. It then illustrates the application being processed across the nodes. Further to this, after destruction of nodes causing link failure, a secondary request will cause the network to self-heal and create enough capacity to process the application again.

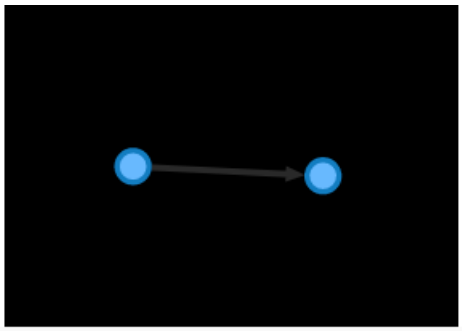
5.7 Self-organising Test-Case

The self-organising nature of this variant is illustrated through its reduction in user facing functionality and its higher decree autonomy. With only a slight increase in code complexity it is able to quickly adapt and self-organise at the expense of having a functionality that is static, not dynamic.

Table 5.4 illustrates the multi-step process for the self-healing variant for converging the network. Upon cell spawn the network will automatically divide to maximum capacity, as defined by its starting conditions, and then self-organise to distribute the function quantity correctly. If a node is to timeout

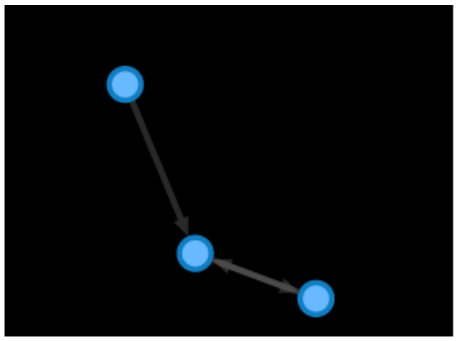
Stage	Description	Figure
1 – Cell spawn	The initial cell is spawned and waits until it receives a user request. An output message can be seen illustrating the cell has bound to an address and port. OKA's can be seen but are currently blank as the node has no local information. No OPARs can be seen as the node has no peers to advertise or advertise to	5.14
2 – API App Request	The user submits an application request [[1,3,5,7],2]] to cell localhost:16983. This cell notices that function 1 does not exist in its local info. It has spare capacity so chooses to differentiate. It then notices that the next function also does not exist but due to having no capacity it divides to meet this requirement. It prompts the daughter cell to differentiate to function 3.	5.15
3 – Cell Division	The initial cell continues to spawn until it hits the limit of it's local division capacity. All nodes become aware of each other through OPARs and the network is fully connected.	5.16
4 – Capacity Requests	Having now used all available capacity the initial cell needs to find capacity for an additional function. It therefore sends an OCRQ,7 to all subscribers. Any subscriber with available divisible capacity will request a peer advertisements and then wait a random amount of time before dividing and instructing the cell to differentiate to that function. In this situation the timeouts cause an additional cell to be created which in this instance can be used for redundancy.	5.17
5 – Network convergence	The network has developed capacity for all the functionality for the application and all nodes are able to communicate.	5.18
6 – App processing	The initial cell responds to he user informing them that all functionality can now be reached. The user then pushes the application. In this simplistic example the data is merely multiplied by the value of each function as it reaches each cell. The result can be seen in FAOT	5.18
7 – Node Failure	Two cells are killed in the network which is represented by the loss of connections in the network. The other nodes stop subscribing after a predefined timeout	5.20
8 – Self-heal	The user pushes the application again. The cell notices that capacity is lost and requests further divisions. The additional cells with bi-directional connections in the network are dead. The FAOT illustrates the networks ability to process the data after healing. This self-healing is a foundational resilience process.	5.20

Table 5.3: User driven test case stages



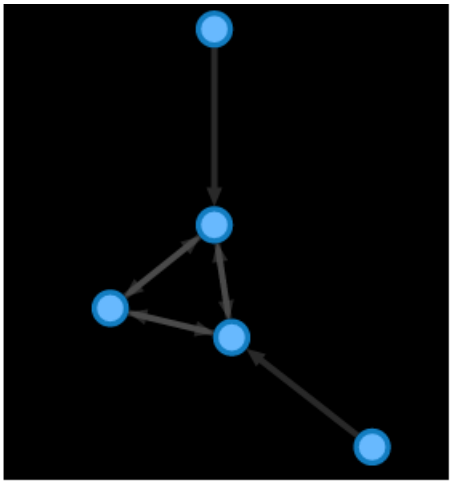
.4016185	localhost:18598	Output	Bound to localhost:18598
.4078302	localhost:18598	out	{"typ": "sub", "node": "localhost:19377"}
.4281454	localhost:18598	pub	{"OKA": "[[3], [], []]"}
.4424105	localhost:18598	pub	{"OPAR": ["localhost:19377"]}

Figure 5.15: User Test Case Stage 2 - API Application Request



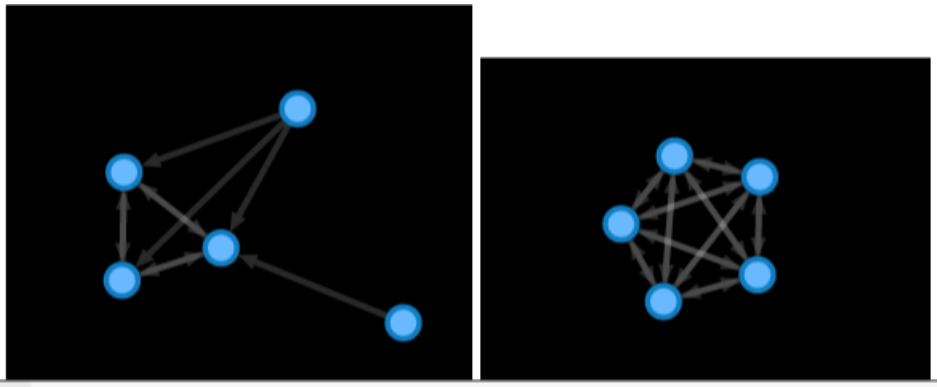
.4911084	localhost:19377	recv	{"id": "localhost:19925", "typ": "OPAR", "pl": ["localhost:19377", "localhost:18598"], "ttl": 0}
.4983287	localhost:16423	recv	{"id": "localhost:19925", "typ": "OPAR", "pl": ["localhost:19377", "localhost:18598"], "ttl": 0}
.3648415	localhost:19925	pub	{"OPAR": ["localhost:19377", "localhost:18598", "localhost:16423"]}
.3822336	localhost:19925	recv	{"id": "localhost:19377", "typ": "OPAR", "pl": ["localhost:18598", "localhost:19925"], "ttl": 0}

Figure 5.16: User Test Case Stage 3 - Cell division



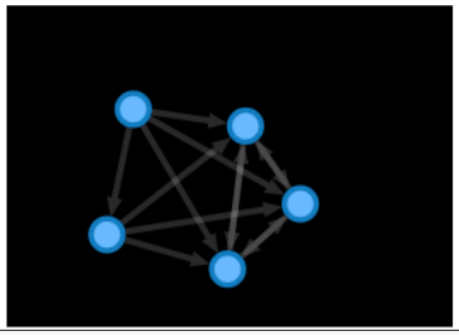
.7272208	localhost:19377	pub	{"OCRQ": "7"}
.7336874	localhost:18598	recv	{"id": "localhost:19377", "typ": "OCRQ", "pl": "7", "ttl": 0}
.7369978	localhost:19925	recv	{"id": "localhost:19377", "typ": "OCRQ", "pl": "7", "ttl": 0}

Figure 5.17: User Test Case Stage 4 - Capacity requests



.112699	localhost:19377	pub	{"FDPR": [[1, 3, 5, 7], 2]}
.1215367	localhost:19377	recv	{"id": "localhost:16423", "typ": "FDPR", "pl": [[1, 3, 5], 14], "ttl": 0}
.1295443	localhost:19377	recv	{"id": "localhost:19925", "typ": "FDPR", "pl": [[1, 3], 70], "ttl": 0}
.1375098	localhost:19377	recv	{"id": "localhost:15924", "typ": "FDPR", "pl": [[1, 3, 5], 14], "ttl": 0}
.1450138	localhost:19377	recv	{"id": "localhost:18598", "typ": "FDPR", "pl": [[1], 210], "ttl": 0}
.1519082	localhost:19377	pub	{"FAOT": 210}

Figure 5.18: User Test Case - Stage 5 - The network becomes converged and the application processes successfully

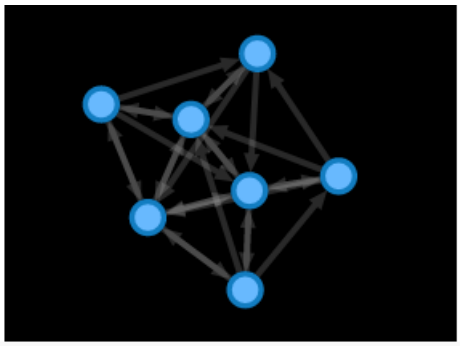


.832397	localhost:19925	out	{"typ": "seen timeout", "node": "localhost:15924"}
.7634516	localhost:16423	out	{"typ": "seen timeout", "node": "localhost:15924"}
.915199	localhost:19377	out	{"typ": "seen timeout", "node": "localhost:15924"}
.7607853	localhost:18598	out	{"typ": "seen timeout", "node": "localhost:15924"}
.1576452	localhost:18598	out	{"typ": "seen timeout", "node": "localhost:16423"}
.2136884	localhost:19925	out	{"typ": "seen timeout", "node": "localhost:16423"}
.2937992	localhost:19377	out	{"typ": "seen timeout", "node": "localhost:16423"}

Figure 5.19: User Test Case - Stage 6 - timeout

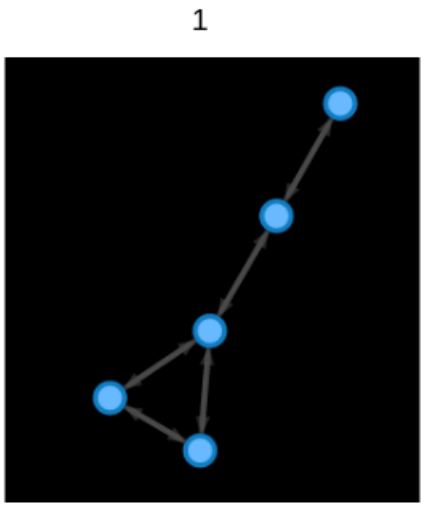
Stage	Description	Figure
Initial spawn	The initial cell is spawned with a function list and division depth. In this case it is 4 functions and divide depth of 2. We can see that the 5 node structure is again formed. With OKAs and OPARs causing the networking to form quickly	5.21
Self-organisation	The network self organises to provide the required structure. The OKAs illustrate the nodes function as the first element and the list of functions visible in its neighbourhood as the second element (array). There is a repeated function 4 seen by all nodes as there is a redundant cell.	5.21
App processing	The user submits an application which is processed very quickly with minimal processing time. The FDPR messages are slowly iterated upon until the result FAOT is presented to the user.	5.22
Self-heal	A cell is deleted causing timeouts to occur. The mother node to the cell recognises that it is a daughter cell and self-heals.	5.23

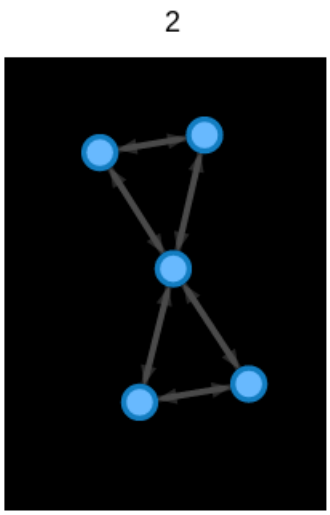
Table 5.4: Self-Organising Variant Test-case Steps

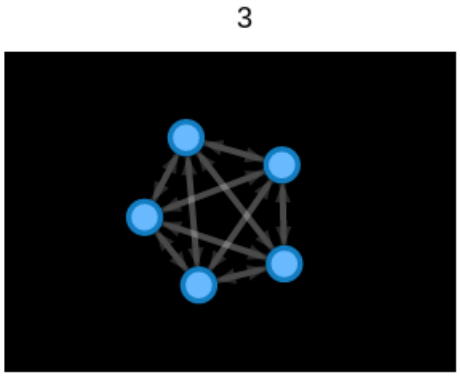


.8563113	localhost:15423	Output	Bound to localhost:15423
.8624673	localhost:15423	out	{"typ": "sub", "node": "localhost:18598"}
.8770545	localhost:17289	Output	Bound to localhost:17289
.8789332	localhost:17289	out	{"typ": "sub", "node": "localhost:19925"}
.2915447	localhost:17289	out	{"typ": "sub", "node": "localhost:19377"}
.301415	localhost:17289	out	{"typ": "sub", "node": "localhost:18598"}
.7387936	localhost:19925	out	{"typ": "sub", "node": "localhost:16423"}
.74626	localhost:19925	out	{"typ": "sub", "node": "localhost:17289"}
.7624285	localhost:18598	out	{"typ": "sub", "node": "localhost:15924"}
.7701836	localhost:17289	out	{"typ": "sub", "node": "localhost:16423"}
.7758203	localhost:19377	out	{"typ": "sub", "node": "localhost:17289"}
.7843719	localhost:18598	out	{"typ": "sub", "node": "localhost:15423"}
.7911336	localhost:15423	out	{"typ": "sub", "node": "localhost:19377"}
.813986	localhost:15423	out	{"typ": "sub", "node": "localhost:19925"}
.822885	localhost:15423	out	{"typ": "sub", "node": "localhost:15924"}
.831322	localhost:19377	out	{"typ": "sub", "node": "localhost:15423"}

Figure 5.20: User Test Case - Self-heal upon request





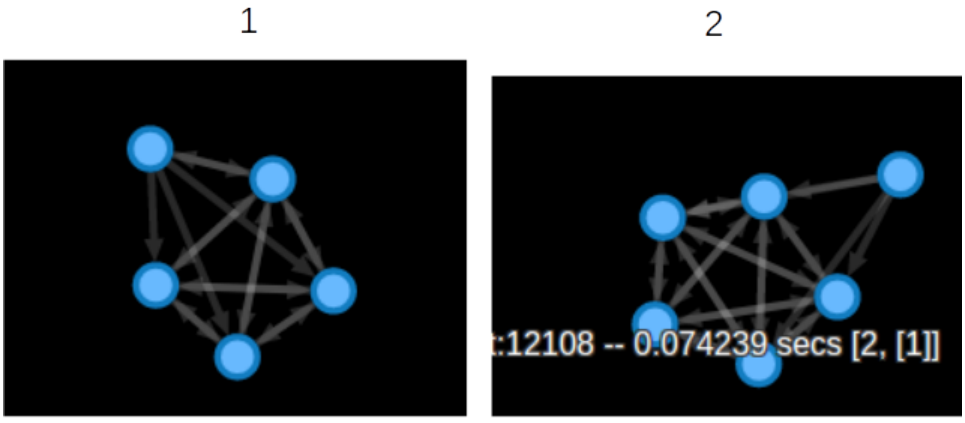


.871364	localhost:19037	Output	Bound to localhost:19037
.0139697	localhost:16770	Output	Bound to localhost:16770
.0214038	localhost:16770	out	{"typ": "sub", "node": "localhost:19037"}
.162643	localhost:18646	Output	Bound to localhost:18646
.1731503	localhost:18646	out	{"typ": "sub", "node": "localhost:16770"}
.187884	localhost:18646	pub	{"OKA": "[1, [0]]"}
.1983442	localhost:18646	pub	{"OPAR": ["localhost:16770"]}
.8866034	localhost:19037	out	{"typ": "sub", "node": "localhost:16770"}
.023684	localhost:16770	out	{"typ": "sub", "node": "localhost:18646"}
.034954	localhost:16770	pub	{"OKA": "[2, [0, 0]]"}
.046963	localhost:18646	recv	{"id": "localhost:16770", "typ": "OKA", "pl": "[2, [0, 0]]", "ttl": 0}
.0624037	localhost:10527	Output	Bound to localhost:10527
.0665514	localhost:10527	out	{"typ": "sub", "node": "localhost:19037"}
.1749537	localhost:18646	pub	{"OKA": "[4, [2]]"}
.186847	localhost:16770	recv	{"id": "localhost:18646", "typ": "OKA", "pl": "[4, [2]]", "ttl": 0}
.2092917	localhost:11734	Output	Bound to localhost:11734
.2143772	localhost:11734	out	{"typ": "sub", "node": "localhost:10527"}
.2207909	localhost:11734	pub	{"OKA": "[1, [0]]"}

Figure 5.21: Self Organising Test Case Step 1 - Network convergence

.967798	localhost:19037	pub	{"OKA": "[4, [1, 3, 2, 4]]"}
.1021693	localhost:16770	pub	{"OKA": "[1, [4, 2, 3, 4]]"}
.1447468	localhost:10527	pub	{"OKA": "[3, [4, 4, 1, 2]]"}
.2501004	localhost:18646	pub	{"OKA": "[2, [1, 4, 3, 4]]"}
.3110518	localhost:11734	pub	{"OKA": "[4, [3, 4, 1, 2]]"}
.299443	localhost:19037	pub	{"FDPR": [[1, 2, 3], 8]}
.35691	localhost:10527	pub	{"FDPR": [[1, 2], 24]}
.3649595	localhost:16770	pub	{"FAOT": 48}
.3728185	localhost:18646	pub	{"FDPR": [[1], 48]}

Figure 5.22: Self Organising Test Case Step 2 - self-organisation



.0912669	localhost:19037	out	{"typ": "seen timeout", "node": "localhost:11734"}
.2330203	localhost:16770	out	{"typ": "seen timeout", "node": "localhost:11734"}
.3921592	localhost:18646	out	{"typ": "seen timeout", "node": "localhost:11734"}
.4631588	localhost:12108	Output	Bound to localhost:12108
.4660118	localhost:12108	out	{"typ": "sub", "node": "localhost:10527"}
.2819781	localhost:10527	out	{"typ": "sub", "node": "localhost:11734"}
.2865357	localhost:10527	out	{"typ": "sub", "node": "localhost:12108"}
.2902765	localhost:10527	out	{"typ": "seen timeout", "node": "localhost:11734"}

Figure 5.23: Self Organising Test Case Step 3 - self-heal

then it's mother cell will quickly spawn another. Finally a user can push their application to the MC system at any point to have it processed.

5.8 Analysis and Comparison

Characteristic	**User-driven**	**Self-organising**
Feature library	dynamic and adaptable	static and stable
Network Structure	unstructured	Semi-structured
Organisation	externally-prompted	autonomic
Performance	timeout-reliant	autonomic

Table 5.5: Comparison of architecture characteristics

Two similar yet slightly different architectural variants of the embryonic cloud architecture have been presented, with examples for each which illustrate their functional capability. The first architecture is user-driven. It has a strong alignment with the functional cloud computing requirements as it can dynamically support different applications and scale according to demand. However, whilst meeting the functional requirements this implementation suffers from a number of short-comings. Firstly, its organisational behaviour is entirely dependent upon user input - an external prompt. Whilst this is functional it ensures the system lacks the true autonomic and complex system attribute of being self-organising. When cells wait for an external request they become dependent upon this external input to make decisions. A truly self-organising system will organise according to internal stimuli. Therefore to mitigate this lack of autonomic behaviour and further align with the thesis' hypothesis, a second variant was developed which included self-organising behaviour to provide the required embryonic functionality. An overall comparison of the two architectures is shown in table 5.5. Neither one of these variants is incorrect and has can be seen from this discussion each have advantages and disadvantages which will be suitable for different use-cases. For example if the environment has only one application or function distribution requirement then the autonomic variant will likely be chosen. A system designed for general public use or in the most constrained of resource requirements might opt for the user-driven variant.

Functions The second architecture attempts to mitigate this short-coming of autonomic functionality through providing self-organising functionality. It mimics the architecture simulated by CA in section 4.4.2.4 where cells will aggressively divide until their environment is at full capacity. In this manner they produce a semi-structured network. This heavy resource reliance provides them the ability to more easily and rapidly respond to node failure without user-prompt or lengthy timeouts. However this feature comes at the expense of all cells organising according to a desired and static function distribution and thus complicates dynamic adjustment according to external need. Adding

different functionality requires all nodes to be destroyed and recreated with new start-up parameters or "DNA". However it should be recalled that the results of the previous simulation highlighted that the greater the initial starting network size the greater the chance of survival.

Network Structure A key difference between the two variants is that the autonomic uses a structured network and the user-driven does not. A structured network has disadvantages in that it exposes predictable points of weaknesses. In contrast the unstructured network can produce more resilient, random and emergent structures which provide resilience in some areas but suffer from management and bandwidth issues. The autonomic / structured network can ultimately scale in a more stable manner due to not suffering from node communication and subscription issues. A key way in which this network structure can be varied in terms of both performance and resilience is through adjusting the number of children nodes that a cell can spawn. Due to the flooding-based routing employed, even one additional node can have drastic effects on bandwidth contention.

Performance The autonomic variant reduces communication overhead and improves temporal performance by allowing each node to organise according to local information i.e. decisions do not need to be made through discussion or cooperation. To compare the two examples above, the user-driven variant took 785 seconds in total against 300 seconds for the autonomic variant. Whilst the reader should be aware that the performance could be tuned, timeout operations and waiting for user input slow the process considerably. Regardless the proof-of-concept implementation requires performance tuning in both variants prior to an accurate determination of it's performance.

States As a complex system, one of the defining features which are observable are its states and the transitions between them. If the goal of the system is to provide trusted service delivery then 3 states can be observed when considering the requirements at this point of the experimentation which are shown as as a Finite State Automata (FSA) in figure 5.24 and are defined below:

1. **Convergence** - is the process in which the network is moving towards the state in which it can deliver the service. It can be seen as the network is first initialised up until it is stable.

2. **Acceptable Service Delivery** - is when the network has reached an acceptable state and can process the application.

3. **Degraded Service Delivery** - is where the system is between stages. It may have lost nodes to failures or been self-organising to achieve convergence. At this point service delivery may be sporadic.

4. **Failed** - the system which does not successfully converge or which failures happen quicker than divides will be in a state of failure and must be restarted.

In the context of the user-driven and autonomic variant, these states are the same. However the autonomic variant will change states without any user prompt. This distinguishing characteristic

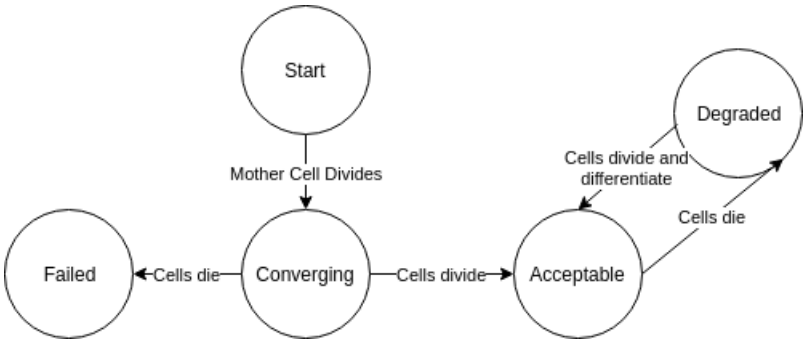


Figure 5.24: Finite state automata for the embryonic implementation

highlights the greater alignment with emergence for the autonomic variant. A key point to note from this FSA is that the change between states is driven by the cellular aspects of differentiation and division.

5.9 Summary

This chapter presented the design and implementation for the embryonic cloud architecture. Through attempting to meet the requirements, two different variants were derived. One was user-driven and the other autonomic. Although both met the basic cloud functionality initially defined, the second variant provides strong autonomic behaviour as the first is driven by user input. Due to this it is concluded that although combined to meet all requirements successfully, the autonomic variant is largely the more resilient as it contains the emergent property of self-organisation.

Chapter 6

Proof-of-Concept Testing and Evaluation

6.1 Introduction

In the previous chapter, a proof-of-concept for the previously defined embyronic architecture was built and evaluated. It was shown that in order to meet all the requirements strongly, two different variants arose. The first was user-driven and the second autonomic. In order to determine the resilience capability of an embyronic-inspired cloud platform, this chapter evaluates the performance of the autonomic variant against application processing and resilience criteria.

Firstly, a methodology is defined to provide empirical consistency and validate the proposed tests. It leverages a consistent experimentation environment which automatically collects and then pre-processes a timeline of events for each test run. Tests are initially conducted for networks with 0 failure rates, to provide a lower bound for application processing and understand more about the independent variables effect upon the network characteristics. Next, tests are undertaken upon networks with induced failure rates to understand the resilience capacity of this network. Finally an analysis of these results is given to provide insight in to methods of measuring the resilience of the platform such that it may assist in answering this thesis' hypothesis.

6.2 Methodology

All tests were run on a 64bit i7 quad core Laptop running Debian Linux where all cells used local-host addresses and randomised port numbers. All communication was conducted over the operating system's internal network stack and therefore was minimally affected by any external conditions. Additionally, all activity on the computer including background services was kept to the bare minimum

to provide fair and consistent test results.

Tests were autonomously executed in batches using a suite of scripts to provide empirical consistency. Messages both sent and received by all cells were pushed to the debug server as per the setup in the previous chapter where they were stored in a MYSQL database. However this time there was no front-end web GUI. The mother cell was spawned with the configuration options for that test. A loop then iterates every second up until the upper time limit (5 minutes in the first instance but up to 20 for specific investigations). The events which occur during the loops typically consists of pushing applications to be processed and instigating artificial faults by killing nodes according to a pseudo-random threshold. Post-test run, all of the messages would be retrieved from the back end database and parsed to create a timeline of events. When ever a change in the network structure occurs, a graph of the network is constructed using the networkx python library. A history of all graphs is then stored (with a corresponding PNG image) for further analysis. This consists of a number of different algorithms which examine the resilience of the network at each stage. Statistics about the application processing time and messages sent and received are also recorded. Note: that graphs at this stage, particularly for those tests with node failures, were pruned of isolates. Therefore the graphs do not show all nodes that may be active, only those that are connected to other nodes. This is to ensure more accurate reporting of performance and resilience statistics. Figure 6.1 illustrates this process.

6.2.1 Dependent Variables

The motivations for this evaluation are two fold: firstly the performance and resilience of the architecture must be evaluated to ensure it meets the fundamental requirements of providing a resilient service. Secondly the data derived must be in a state that it can be used to compare with disparate cloud architectures. As defined previously, resilience is the ability for the network to deliver the service in a trusted manner, within the face of changes. Therefore two groups of quantifiable measures are taken for each test. Firstly the success of the application executed at each point, and the quantity of messages it took to accomplish this. Secondly, the structure of the network at each point as described by a number of selected graph algorithms.

6.2.1.1 Application Performance measures

The first group of metrics are employed to evaluate the performance of application processing upon the architecture. These seek to determine whether the embryonic architecture can successfully process the application in the state it is currently in. How long does it take to process the application and the number of messages sent and received during the course of this process? For the purposes of this testing, the applications used are only representative of what is considered to be a real-world data processing task as discussed previously. This is accomplished largely as it permits the timeline of activity within each test to be examined easily from an external viewpoint. Determining if the

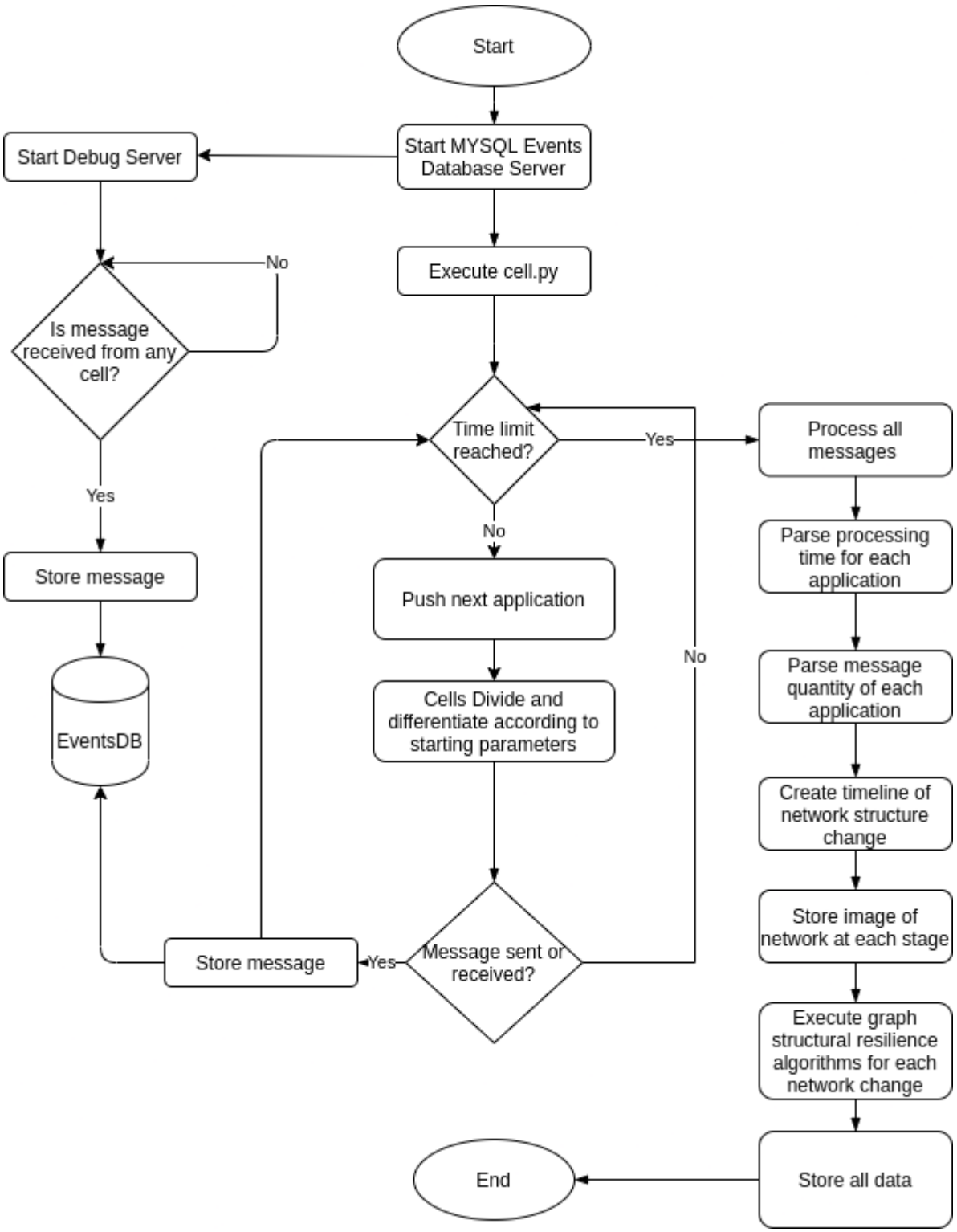


Figure 6.1: The execution flow for the methodology for batch experimentation of embryonic system performance and resilience tests

application has been processed correctly requires the same logic as processing it upon the collection of cells. Moreover a simplistic data processing application still permits investigation and evaluation of the success of the platform as the focus is in connecting the distinct function types together to form a complete application. Therefore the example services consist of a simplistic function which merely multiplies the data it receives by its function number and then pushes it to the next function until the data cannot be processed anymore due to all functions being exhausted. The application values are generated according to a maximum value and cells will differentiate according to their local information driven self-organisation as detailed in section 4.4.2.4. Table 6.1 presents an example data processing application for 5 functions and input data of 2. As the data is received by each function it is multiplied by that value and then pushed to the next. Although simplistic, this method is chosen as it minimises any effects upon the system performance that may result from more complex applications. Additionally the ability for software environments to execute arbitrary code is not in question and therefore does not need to be evaluated. This test focuses upon the networking architecture and self-organisation which permits the passing of messages between arbitrary software functions.

Parameter	**Data**
Functions	5
Data	2
Stage 0	[[1,2,3,4,5], 2]
Stage 1	[[1,2,3,4], 10]
Stage 2	[[1,2,3], 40]
Stage 3	[[1,2], 120]
Stage 4	[[1],240]
Stage 5	[240]

Table 6.1: Example data processing application messages for performance testing

Applications are pushed to the architecture from the API every 10 seconds to allow time for the other messages to clear for more accurate reporting. The data is incremented each time to ensure different results to allow for ease of measurement. As the timeline is generated from processing the messages after the experiment has concluded, performance related values are recorded next to each application which are presented in table 6.2.

6.2.1.2 Graph-based metrics

The structural resilient characteristics were measured through a number of graph-based algorithms which have been used throughout literature to examine the resilience of network structures (section 2.4.6). The purpose of these are to investigate the resilience of this system for comparison to others, in addition to determining the most effective resilience metric for the embryonic system. They have

Variable	**Description**	**Reason**
Application Data (*Data*)	The output of the processed data.	Used to determine which application the messages are for and what stage it is at.
Total Process Time (*TotalProc*)	The time in seconds from the first message to the last.	How long does this application exist for in total?
Shortest Process Time (*ShortProc*)	The time in seconds from the first message to the first output message.	How quickly can the data be processed?
Longest Receive Time (*LongRecv*)	The time in seconds from the first message to the last received message.	How long are output messages still cycling around the network?
Process Messages (*Proc*)	The total number of data processing messages	How many messages does it take to process the data?
Output Messages (*Out*)	The total number of data result messages.	Used to determine if the application was successful and also how many messages appear.
Receive Messages (*Recv*)	The total number of messages received by nodes.	Used to determine the overall effect upon network contention.

Table 6.2: Variables recorded per each test to record network performance

been chosen for their ease of comparison to other architecture types and the ability to summarise the network structure succinctly.

There are many possible metrics to choose from and whilst all may have some form of value, many were pruned from the list of possible metrics. For example those which did not summarise the information into one value instead providing lists of nodes or mappings between nodes and values due to their difficulty in providing comparison between differing architecture types. Moreover, many common metrics were not used due to their unsuitability for directional-graphs, which is a feature of this architecture and a prominent component of its resilience due to the publisher/subscribe method of node removal. All variables are listed in table 6.3 whilst the mathematical foundations are described below.

A graph is defined as a tuple: $G = (V, E)$ Where V is the set of vertices (nodes) and E is the set edges between them. Edge e_{ij} connects vertex v_i with vertex v_j The total number of nodes N is thus $|V|$.

Average node connectivity of a k-connected graph is defined in equation 6.1

$$\bar{k}(G) = \frac{\sum_{u,v} k_g(u,v)}{\binom{p}{k}} \tag{6.1}$$

Where

$$k(G) = min(k_G(u,v) : u, v \in V(G)) \tag{6.2}$$

Degree assortativity for a directed edge network as defined in equation 6.3 where "e_j, k is the joint probability distribution of the excess degrees of the two nodes at either end of a randomly chosen link....μ_q and ϵ_q are the expected value or mean and standard deviation of the excess degree distribution q_k" (Thedchanamoorthy et al. 2014).

$$r = \frac{1}{\epsilon_q^2}\left[(\sum_{jk} jke_j, k) - \mu_q^2\right] \tag{6.3}$$

Where E is the number of edges. $\overline{j^\alpha} = E^{-1}\sum_i j_i^\alpha$ and $\sigma^\alpha = \sqrt{E^{-1}\sum_i (j_i^\alpha - \overline{j^a})^2}$; $\overline{k^\beta}$ and $\sigma\beta$ are similarly defined.

Average shortest path length is defined in equation 6.4 (Kulig et al. 2015). Where V is the set of nodes in G, $d(s,t)$ is the shortest path from node Sub and t and n is the number of nodes.

$$\bar{d} = \sum_{s,t \in ''V} \frac{d(s,t)}{N(N1)} \tag{6.4}$$

Clustering coefficient is defined in equation 6.5 (Saramäki et al. 2007)

$$CC = \frac{1}{N}\sum_{V \in G} C_v, \tag{6.5}$$

Network criticality is defined in equation 6.6 where $Trace(L^+$ is the Moore-Penrose inverse of the Laplacian matrix (Alenazi & Sterbenz 2015*a*).

$$\hat{\tau} = \frac{2}{N-1} Trace(L^+) \tag{6.6}$$

Finally, **effective graph resistance** (for undirected networks) is defined in equation 6.2.1.2 (Alenazi & Sterbenz 2015*a*).

$$R_G = N \sum_{i=2}^{N} \frac{1}{\lambda} \tag{6.7}$$

Variable	**Description**
N	The total number of nodes
Average Node Connectivity ($\bar{k}(G)$)	The average number of nodes each node is connected to. It gives "the expected number of vertices that must fail in order to disconnect an arbitrary pair of non adjacent vertices" (Beineke et al. 2002).
Degree assortativity (r)	The similarity of node connections. Chosen as the graph is directed and therefore should provide insight as to this effect upon the resilience. (Alenazi & Sterbenz 2015*a*)
Average Shortest Path Length ($\bar{d}$)	The average number of hops between nodes (Alenazi & Sterbenz 2015*a*). Chosen as it describes network structure. A lower value should mean a greater chance of application execution.
Clustering Coefficient (CC)	The degree to which nodes cluster together. Chosen because it highlights structural issues in a simplistic way (Alenazi & Sterbenz 2015*a*).
Network Criticality ($\bar{\tau}$)	Used to measure the resilience of a network against structural changes. A lower value equates to a more structurally resilient graph (Alenazi & Sterbenz 2015*a*).
Effective Graph Resistance (R_G)	Commonly used to measure the resilience of a network against structural changes. A higher value equates to a more structurally resilient graph (Alenazi & Sterbenz 2015*a*).

Table 6.3: Variables recorded per each test to record network structure statistics

6.2.2 Independent Variables

In order to evaluate the previously defined dependent variables whilst the system is under different states a number of parameters were varied (table 6.4). These are related to those which were previously

investigated within chapter 4 using CA although variations exist due to the continuous nature of these tests contrasting with the discrete nature of the CA simulations.

Variable	**Description**	**Range**
Functions ($Func$)	The quantity of possible function types.	2 - 7
Division (Div)	The extent to which nodes will divide. This value is halved at each division	2 – 6
Subscriptions (Sub)	The maximum number of other cells each cell can subscribe to	2 – 8

Table 6.4: Independent variables chosen for their ability to affect service resilience

6.2.2.1 Functions

The first variable is the function quantity ($Func$). It is the same parameter as that used in the CA and also described in the previous section. It is known from the simulated investigation (chapter 4) that the number of distinct functions used to compose an application has a direct effect upon the ability to resiliently deliver the service in local-only communication environments. This can be mitigated through self-organisation techniques although a goal of this experimentation is to determine the upper-bound of function quantity under different characteristics.

6.2.2.2 Division Rate

The first variable which differs from that in the CA-based simulations is the division rate (Div). Due to the change in architecture type each cell is now initiated with a value given to it by its mother cell which defines how many times it should divide, in order to control the number of divisions and the size of the network this value is halved at each cellular division. This value therefore permits the control of the size of the network.

As per the results from the CA simulations, the larger the network (at the onset) the greater the chances of the organisms overall survival, resilience and ability to process applications. In this experiment the size is tested to determine the network performance, i.e. the success and performance of the application under these division rates. The trade-off is that a larger network causes considerably greater resource cost.

6.2.2.3 Subscriptions

The second parameter is the maximum subscriptions allowed per each node (Sub). A greater quantity of subscriptions permits greater node connectivity which enhances resilience and the chance of

application execution being successful. However this comes at the expense of an increased number of redundant messages being transmitted and received putting strain on the network. Moreover, causing a node's neighbourhood to increase varies the effectiveness of the self-organisation as there is now a greater number of cell states to process. This value was adjusted to evaluate how much the network performance was reduced given the increase in connectivity.

Note: this value was swapped for the variation in packet TTL (or connected 0, connected 1 during the CA simulations) as it accomplishes the same increase in application execution success increase with the added advantage of increasing network resilience through increased connectivity. Initial tests using TTL values highlighted that messages would get lost when the network became hierarchical.

6.3 Results and Analysis

This section presents the results of the proof-of-concept test cases for test with 0 failure rates and artificially instigated node failures. Note: during this analysis test variables are often shortened to the initial of the variable. For example Div=4 Func=1 Sub=6 would be abbreviated as D4 F1 S6 for clarity purposes.

6.3.1 0 Failure Rates

The first group of tests do not induce any artificial failures within the nodes, therefore the failure rate of any node was 0%. Despite this they still permit self-organisation and may self-heal if a timeout occurs. These tests were conducted for two reasons: the first is to identify the lower and upper bounds for application execution given different network parameters. The second is to understand how the independent variables affect the structure of the network with no failure rates. This therefore provides baseline values for network size and structure according to the different independent variables. These tests examine the effect of the independent variables upon the structure and size of the network. Insights are gained from examining the quantitative network data and visually inspecting graphs.

6.3.1.1 Division effect upon network structure

Firstly, Div is analysed to determine the network sizes. Note: correlations between Div and the graph resilience metrics were not included as no significant trends were identified. Plotting the curves of the network convergence provides insight into the state change from converging to converged.

Figure 6.2 shows example network structures for each Div at 300 seconds where $S = 4$, whilst table 6.5 lists the corresponding quantity of cells (N). $S = 4$ was chosen as a middle value and it permits a clear visual representation of the number of cells due to a low amount of edges. These results indicate the trend towards increasing cells as the division rate increases.

Figures 6.3, 6.4, and 6.5 plot the network convergence curves for groups of tests where $Div = 4$, 5

Div	*N*
3	7
4	25
5	47
6	85

Table 6.5: N at 300 seconds per division value in figure 6.2

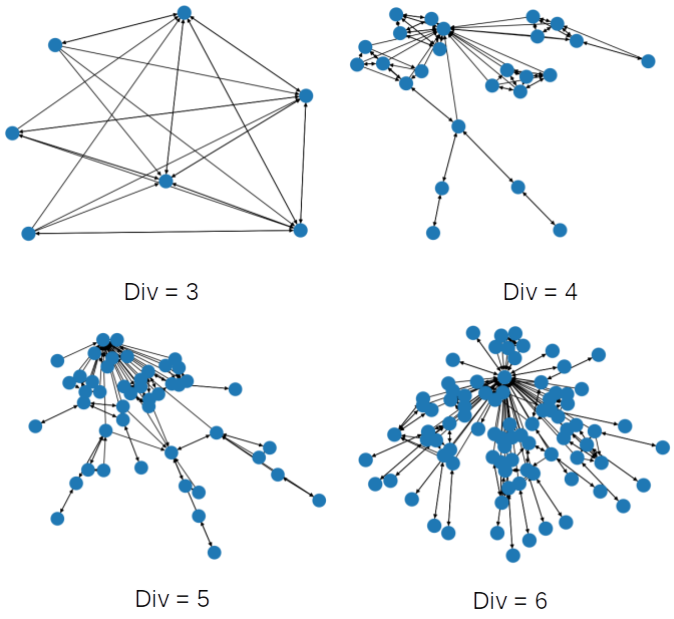


Figure 6.2: Example networks at full convergence with differing division rates. All are where $Sub = 4$

and 6 respectfully. Note that $Div = 3$ was removed from these tests as the low quantity of cells meant that different subscriptions had little variation. The figures illustrate that although N increases as Div increases, overall N does not increase in a linear manner with Sub. For example, in $Div = 4$ and $Div = 6$, the network which converged the first, to the lowest amount of N was $Sub = 7$. $Div = 5$ appears to be anomalous here as the higher values of $Sub = 7$ performs better and converges higher than $Div = 6$. This is due to the odd network structure permitting clusters of nodes to form, unlike in $Div = 6$ where the number of nodes in a cluster is equal to Sub and therefore clusters become isolated (figure 6.6).

When $Sub > Div$ it clearly permits the network to converge within the 300 second timescale. Timeouts appear to occur commonly at lower values of Sub for child cells which cause cells at these stages to continue to divide until they fill their capacity. Table 6.6 shows the timeouts for the test runs and illustrates a general trend towards less timeouts as Sub increase whilst the lowest is always where $Sub = Div + 1$. This is likely due to the constrained Subscriptions added to the cell's division requirements being less than permitted. This illustrates the lower bounds for stable network configurations at 0 failure rates. This is likely to have less of an effect during network stages so the configuration will remain the same to permit stronger comparison.

Div	*Sub*				
-	**3**	**4**	**5**	**6**	**7**
4	22	22	8	16	14
5	34	26	28	5	7
6	45	46	40	27	7

Table 6.6: Quantity of timeouts for each test run

6.3.1.2 Subscriptions effect upon network structure

Next, the network structure is examined from the perspective of changing subscriptions as more edges fundamentally changes the network structure. Table 6.7 presents correlation coefficients between subscription tests and some relevant graph metrics for divisions where the network has converged. Network criticality can be seen affected by mid size networks (2-5) but again this tales off where $div = 6$. A negative value indicates the desired effect, as sub increase the network criticality decreases. Finally effective graph resistance generally increases with subscriptions across all groups of tests which indicates further benefits. Overall this preliminary analysis of these tests indicate that subscriptions are beneficial to the network resilience.

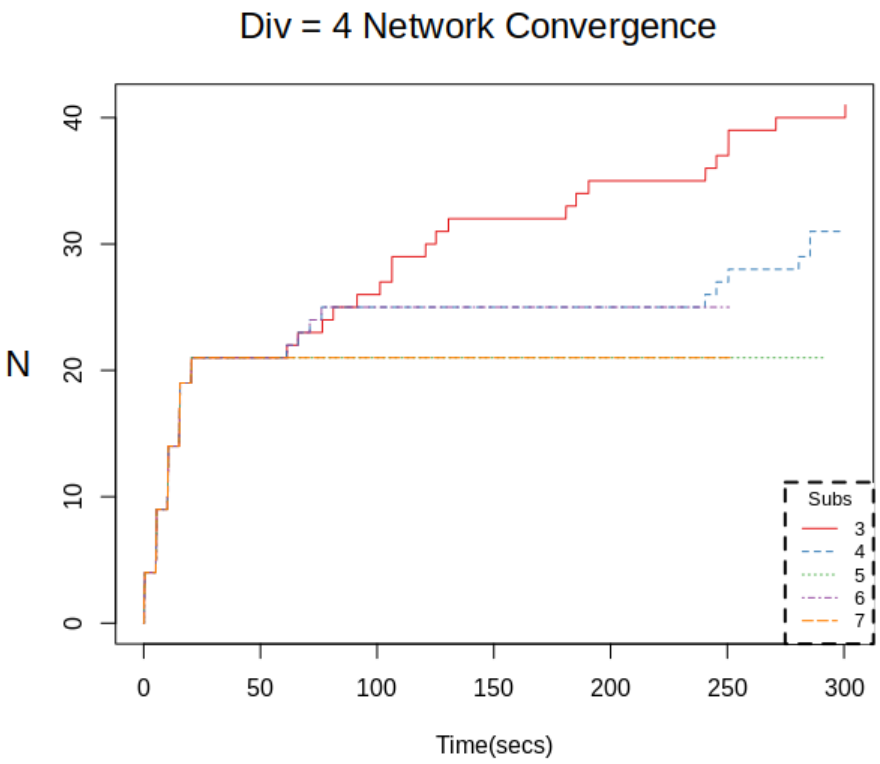


Figure 6.3: Network convergence curve where $Div = 4$

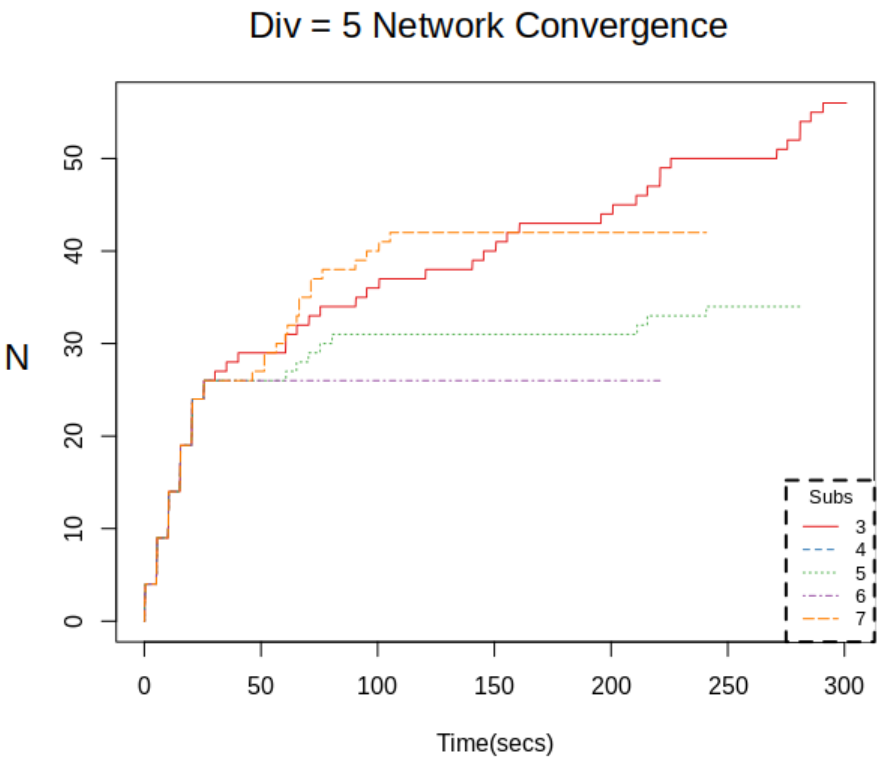


Figure 6.4: Network convergence curve where $Div = 5$

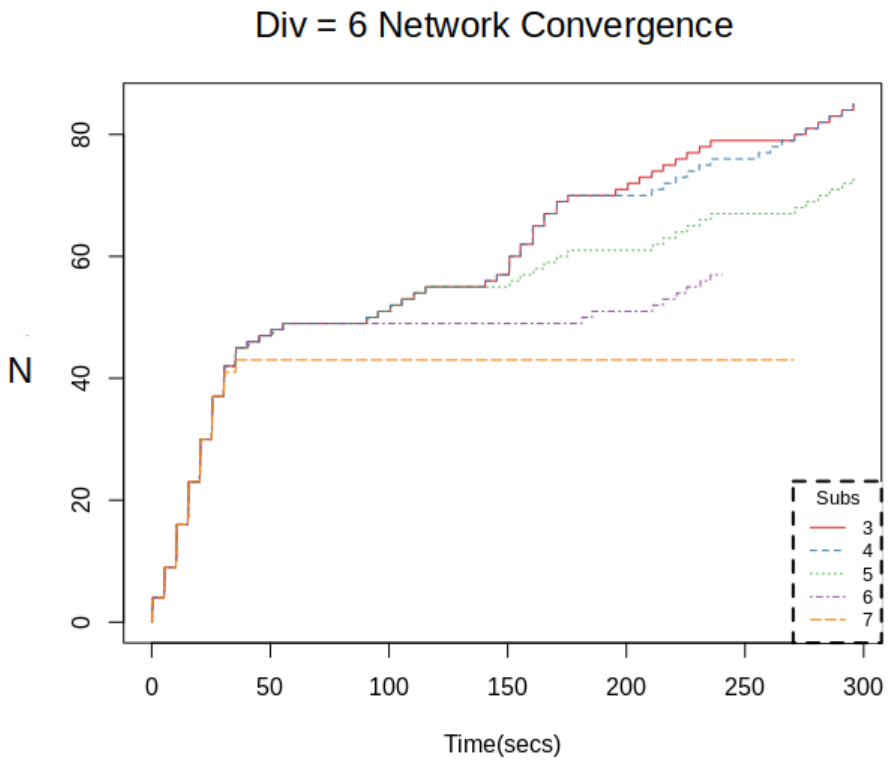


Figure 6.5: Network convergence curve where $Div = 6$

Div	$Func$	CC	$\bar{d}$	$\bar{\tau}$	R_G	r
All	0.411	0.294	0.307	0.0109	0.064	0.0609
3	0.39	0.36	0.09	-0.25	0.316	-0.01
4	0.43	0.143	-0.3927	-0.1292	0.2428	0.127
5	0.65	0.42	-0.55	-0.539	0.527	0.1447
6	0.82	0.95	0.39	-0.059	0.91	0.0524

Table 6.7: Correlation Coefficients For Subscriptions

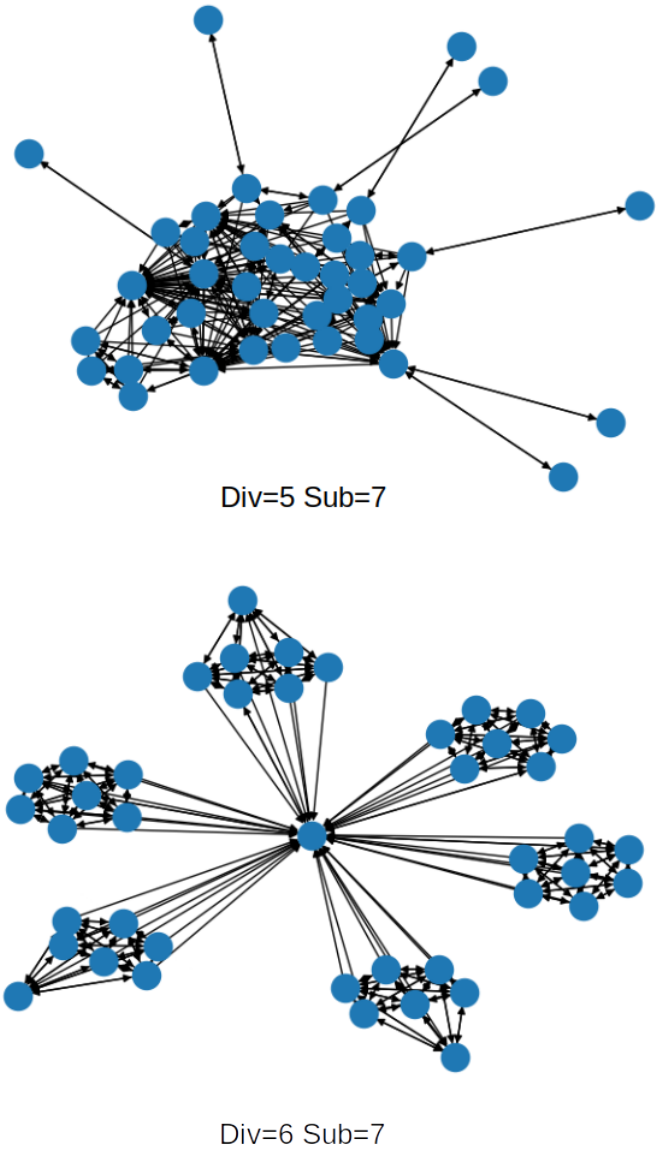


Figure 6.6: $D5$ $F6$ and $S7$. Even clusters are formed at specific division and Subscription quantities.

6.3.1.3 Application Performance

These tests evaluate the effectiveness of the platform to deliver the applications. To provide context for the performance, table 6.8 shows the correlation coefficients for application performance versus the independent variables. *Out* could be argued to be the most relevant as a greater number of output messages means more successful data processing. The strong negative correlation between *Func* and *Out* indicates that as application complexity increases the probability of a successful application decreases. Noting that this trend does not stay true for the *Proc* and *Recv* indicates that in many cases the applications are still being executed although just not to completion. An increase in both *Div* and *Sub* coincide with better application processing success overall.

Table 6.9 shows the correlation coefficients again but only for rows where $Out > 0$. These values show that for successfully executed applications, those tests at lower success rates were less strongly correlated with greater values of *Sub*. *Func* is strongly correlated with *Proc* but this is likely due to the increase in quantity of messages being sent rather than any performance increase. Finally *Div* is positively correlated with *Out*. This is both intuitive due to the increase in N but also informative as it ensures that despite some network structures being conventionally non-resilient (hierarchical) the data is still processed successfully. *Proc* values also increase with *Div*. With the shortest *Proc* increasing with *Func* but not the total, whilst *Proc* does not increase with *Sub*. This is a very informative point, as it means that greater values of *Sub* can be used to enable resilience yet without the extra overhead caused by increasing numbers of nodes as *Div* increases.

–	**Div**	**Func**	**Sub**
Proc	0.56	0.13	0.23
Out	0.30	-0.42	0.26
Recv	0.47	-0.10	0.37

Table 6.8: Correlation Coefficients for Application Performance

-	**Div**	**Func**	**Sub**
Proc	0.65	0.47	0.23
Out	0.43	-0.22	0.21
Recv	0.59	0.18	0.39
ShortProc	0.27	0.46	0.12
LongRecv	0.39	0.00	0.00
TotalProc	0.43	0.09	0.09

Table 6.9: Correlation Coefficients for Application Performance for Successful Applications

Application Execution Success - Tests from all divisions where the division was highest and

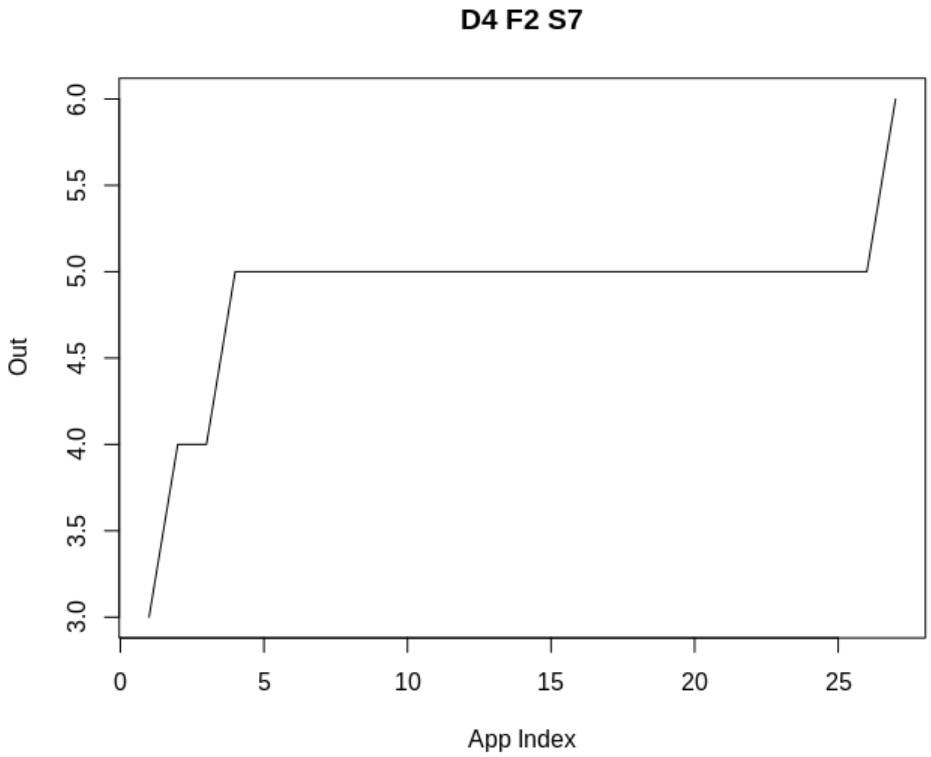


Figure 6.7: $D4\ F2\ S7$ - Stable States

$subs > funcs$ executed the most successfully. Figure 6.7 shows an example case of stable and successful execution output - a baseline. Figure 6.8 shows an example where applications are executed successfully but the performance cycles and presents a glimpse of the system states. Table A.1 -A.4 list the average of *Proc*, *Recv* and *Out* values for all tests. In terms of general performance, some large ratios can be seen between *Recv* messages mostly related to the greater Subscriptions but also due to the size of the network as *Div* increases.

Test configurations where the division was close or equal to the Subscription performed variably. The volatility in these tests provide insight into the network due to the variability in data which are discussed to provide insight into why the performance varied. Figure 6.9a shows the output message rate over time for a test where the system cycles between processing well and processing poorly. Neither of the other two test runs of this variant were as successful. This illustrates the importance of initial starting conditions. Manual investigation of the recorded graph statistics predominately showed nothing of merit other than a general sloping trend across all other metrics, which would indicate that the network became less resilient over time (if these metrics are believed to represent a resilient network in this instance). However figure 6.9b plots the assortativity for this test, which interestingly illustrates a similar cyclic pattern, whilst figure 6.9c shows the plot for clustering which, while not as apparent, still indicates the cycling pattern seen in the application output. Additionally

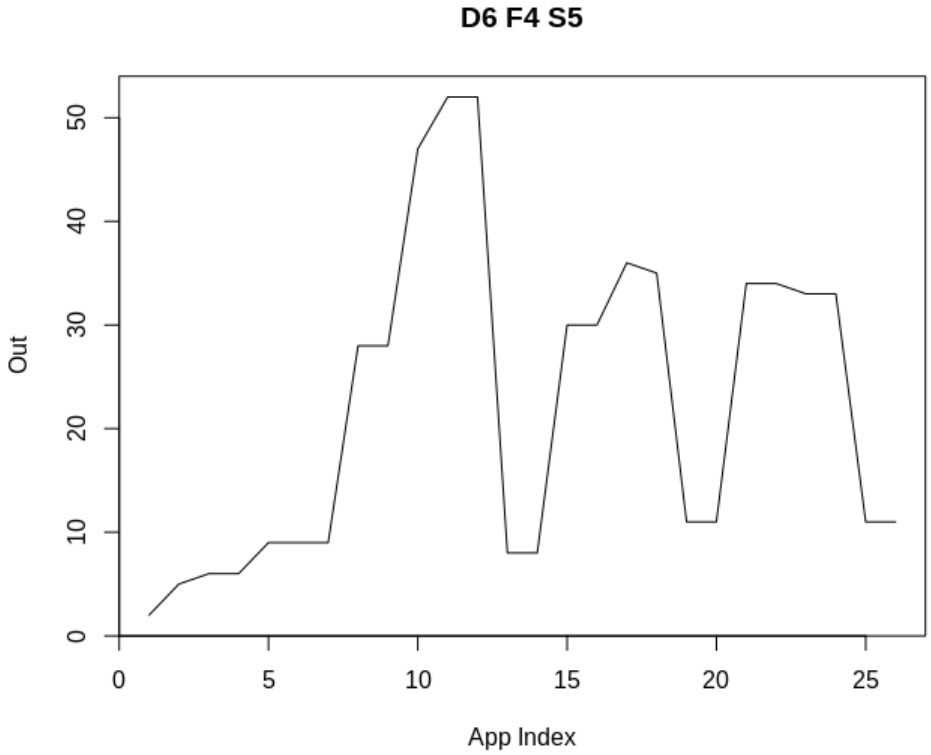


Figure 6.8: $D6$ $F4$ $S5$ Output - Cyclic States

the initial curve as the network converges indicates the convergence state. All 3 graphs indicate the same dip at approximately 210 seconds (note that the output graph is skewed by approximately 20-30 seconds) which highlights the possibility of these graph metrics being applied as a resilience metric for determining network states.

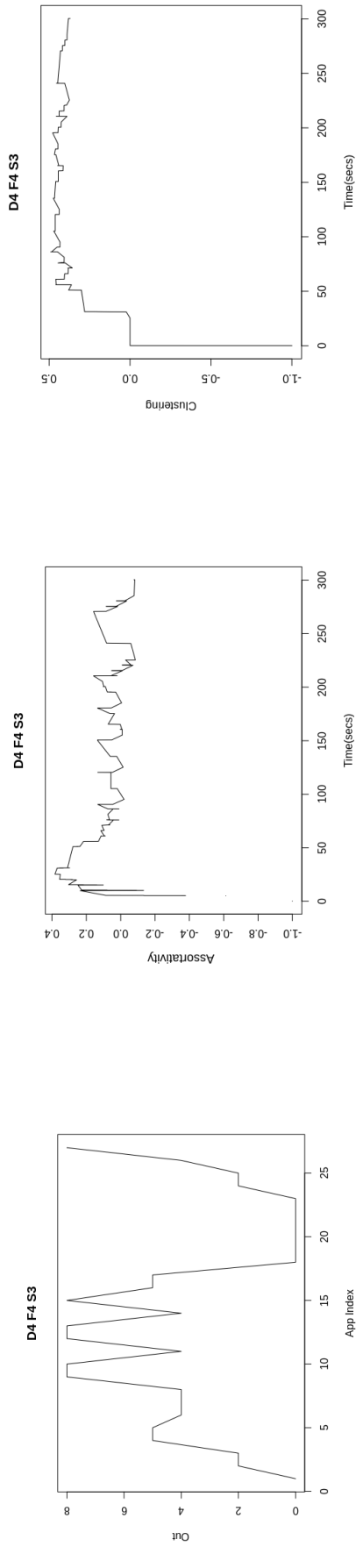


Figure 6.9: D4 F4 S3 graph metric comparison

The effect of starting conditions is again highlighted in figure 6.10 where 3 runs produced two converged networks which did not execute applications successfully and one which did. This indicates the *chaotic* nature of this system. Due to the lack of node failures, networks will remain in this structure. Figure 6.11 shows the output, assortativity and cluster curves for one successful and one unsuccessful network. Note that for run 1, the timeline ends at approximately 100seconds as the network does not change further and the graphs report changes only. Unlike in the previous test, there is little relationship between clustering and the application execution effectiveness. Assortativity clearly drops to 0 in the 1st test run as application execution stops. In the successful test run it remains high through successful execution with a minor upward trend. These results again lend support to assortativity being a strong candidate for measuring resilience in the context of this platform. Other graph metrics were not plotted (as before) due to following the same downward trend. Which is similar to elementary statistics such as node quantity and node connectivity.

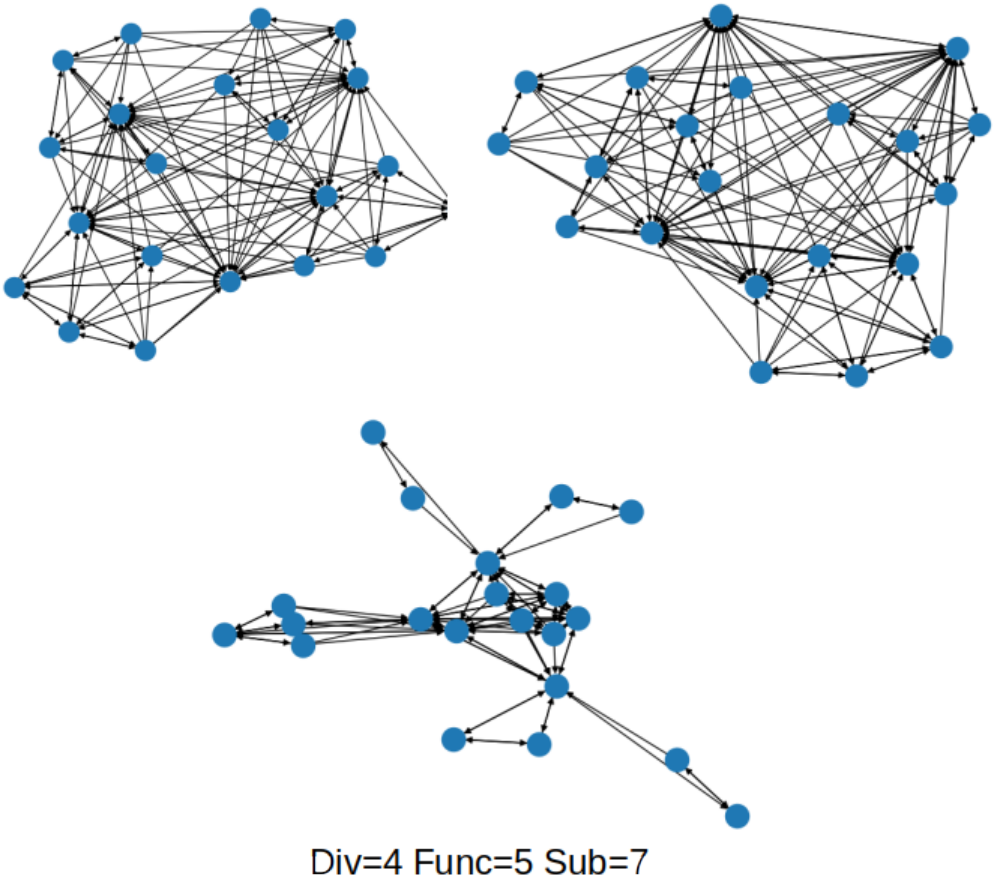


Figure 6.10: 3 runs of the same test parameters D4 F5 S7 illustrating varying network convergence and chaotic conditions.

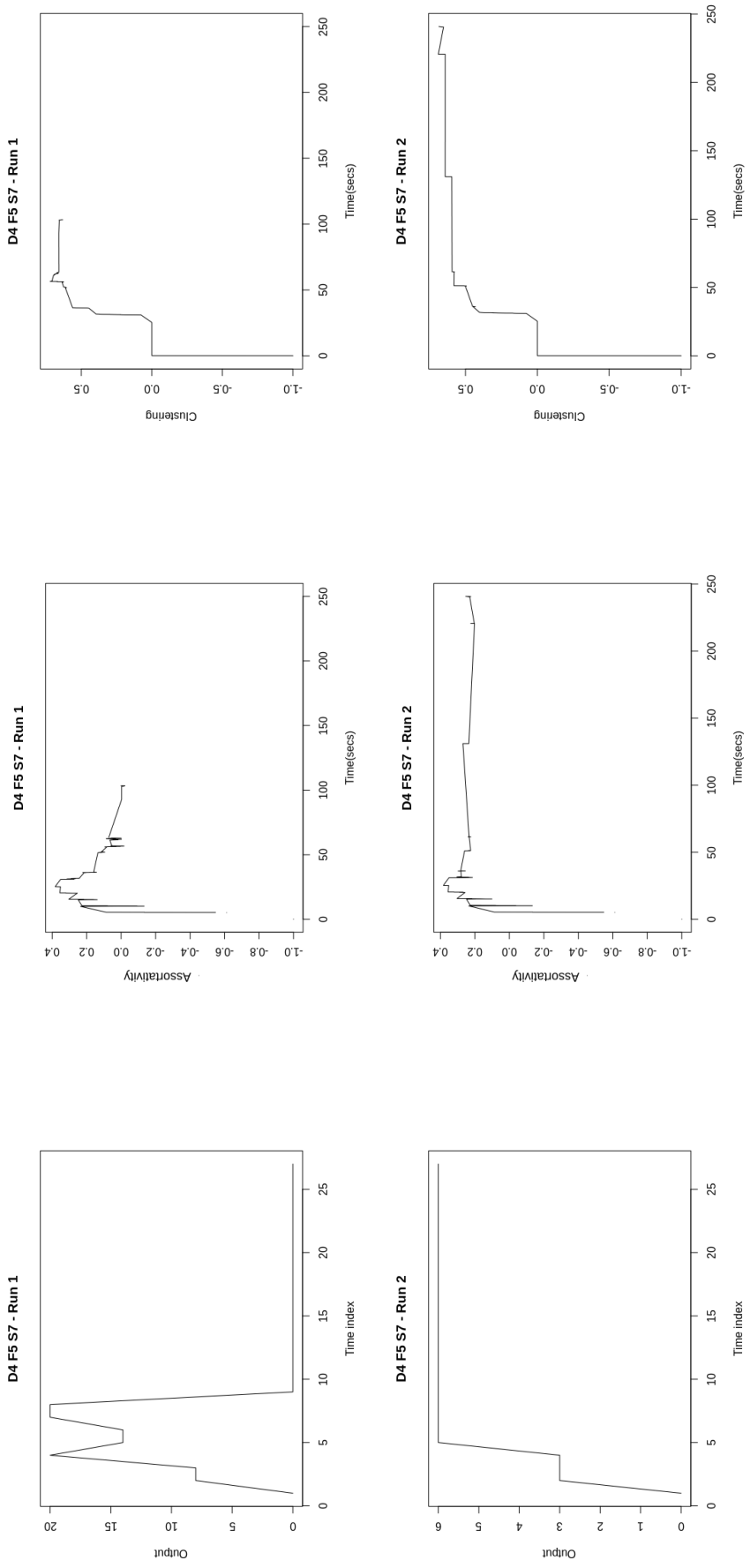


Figure 6.11: Output, assortativity and clustering for 2 different runs of the same parameters (D4 F5 S7)

Unsuccessful Applications - a number of parameter sets produced no outputs over all test-cases. Particularly where $sub = 3$ or 4, and $func = 5$ or 6. Figure 6.12 shows the assortativity and clustering for test $D4$, $F6$ and $S7$ and test $D6$, $F7$, $S3$. As there are no output messages in these tests the curves are plotted next to the process curve for each test to provide an analysis. Process messages have a greater chance than output messages of varying in quantity during each peak as they may be received by different nodes depending upon the network structure and cell self-organisation. However the cyclic nature of the peaks can still be seen in these graphs matching with the assortativity and somewhat with the clustering. Therefore this illustrates that assortativity and clustering are not suitable for determining resilience in terms of the successful application execution alone. The constraining factor here is the divisions and predominately the function size. A clear rule is that outputs will successfully occur when the subscription size is equal to or greater than the function size.

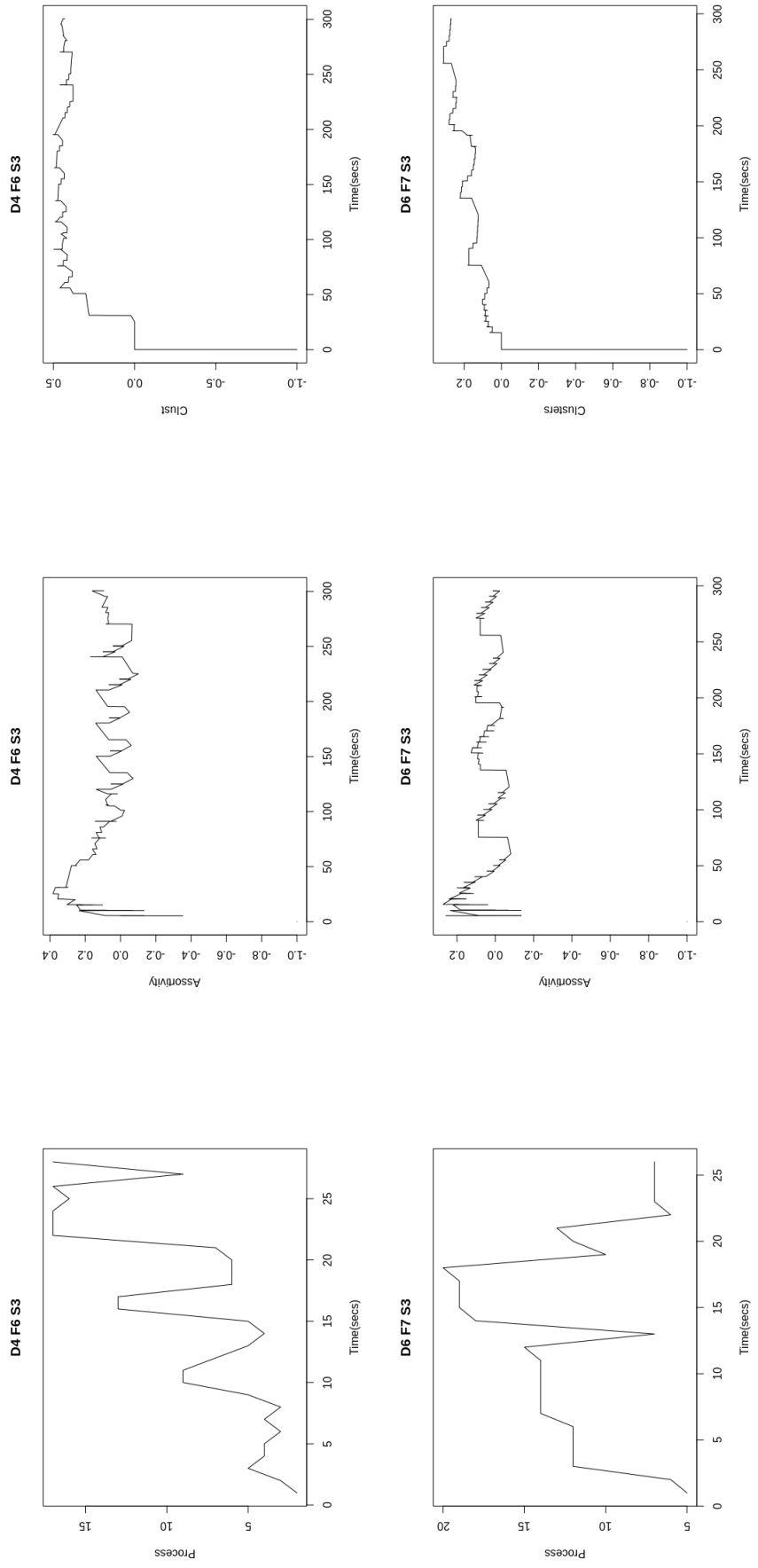


Figure 6.12: Process, assortativity and clustering for 2 different test groups (D4 F6 S3 and D6 F7 S3) where no application executed successfully.

6.3.1.4 Summary

In this section, results of tests with no failure rate were presented and discussed to understand the relationship between the independent variables, network structure, and performance. It was shown that two graph algorithms, assortativity and clusters can illustrate the likelihood of successful application execution under low stress for the short length tests examined here. However this assumes the network parameters to be optimal for the application type. i.e. the subscriptions are greater than the divisions and the division rate must be higher with higher functions.

6.3.2 Failure Rates

These tests follow from those conducted previously although with a fundamental addition of an artificial failure rate. At every 2 seconds, for every currently running child cell a pseudorandom variable is produced. If the variable is less than the given failure rate then the cell will be killed. The purpose of the failure rates is to evaluate the performance of the system under a high rate of churn. Therefore, these tests will evaluate the self-healing capabilities of the system and the application execution success alongside it.

From the previous section a number of points were discovered about networks with no failure rates. The first analysis was concerned with the independent variable effect upon the structure of the network. Analysing the network from the perspective of it being converged is not suitable in this instance as the failure rates cause the network to be dynamic. The failure rate was set at a specific level to further understand the limits of the system under different parameters as a higher failure rate causes a greater perturbation of the system. In this instance, 0.01 was determined as a suitable value after manual experimentation. During preliminary testing, a slightly lower value of 0.001 causes the network to remain stable and high, evidence of which is provided later in this section.

6.3.2.1 Application Performance

To evaluate the application performance whilst the system is self-healing, tables 6.10 and 6.11 present the correlation values for the independent variables against application performance variables for all tests and tests where the application was successfully executed, respectfully. In the values of all tests, the effect upon the quantity of messages processed performs less well in across all values compared to the 0 failure rate tests. In this instance as functions increase the quantity of messages processed actually decrease slightly, whereas in the 0 failure rate tests they increased ever so slightly. This illustrates the stronger relationship for tests which include failures between a high application complexity and the ability of the services to remain connected. For both *out* and *recv*, the performance is slightly worse than the 0 failure rate tests. These results are intuitive overall as the network is under stress in comparison to the 0 failure rates.

In terms of the successful applications, the relationships between the independent variables and the application performance on the whole follow the same trend as 0 failure rate tests yet with worse performance. Some times the performance degradation is to such a degree that the relationship is insignificant such as with $TotalProc$ and $LongRecv$ with $func$. This is interesting as it indicates that in this instance the time taken to complete applications is unrelated to their complexity.

.	**Div**	**Func**	**Sub**
Proc	0.43	-0.11	0.04
Out	0.23	-0.54	0.10
Recv	0.41	-0.25	0.14

Table 6.10: Correlation Coefficients for Application Performance in the Failure Tests

.	**Div**	**Func**	**Sub**
Proc	0.47	0.48	0.09
Out	0.39	-0.27	0.09
Recv	0.48	0.22	0.21
ShortProc	0.07	0.21	-0.02
TotalProc	0.25	0.00	-0.08
LongRecv	0.23	0.00	-0.07

Table 6.11: Correlation Coefficients for Application Performance in the Failure Tests for Successful Applications

Tables A.4 and A.5 list the averages for performance variables for all tests, sorted by *out* descending. Whilst a higher quantity of output messages generally means the application has been executed successfully, it is important to note that a network with a higher output relative to another does not necessarily equate to better performance overall. For example if the ratio of messages ($recv : output$) is skewed towards *Recv* then this may not be operating at optimal performance, considering resource cost. Additionally, the larger value of *div* which equates to a larger network will use considerably more resources.

These tests provide a comparison from the baseline presented for the 0 failure rate tests. Comparing the results there is a general trend across all values with worse performance than the 0 failure rates. This serves to validate the results as the failure rate is likely to worsen performance overall. Unlike in the 0 failure tests, *sub* appears to have no signification relationship with application performance. *Out* increases with *div*, which is again intuitive due to the increase in N. Finally the results of functions follow the same trend as in the previous tests, which is intuitive.

It can be seen that just under 50% of the test cases still successfully executed with an average of 1 or more output messages, albeit often with a considerable amount of messages received and processed.

Additionally a number of tests which had no output messages still had a considerable quantity of messages processed and received. These results mostly consisted of high application values which is intuitive and illustrates that lower application complexity strongly increases the chance of application survival.

In terms of temporal performance, some interesting performance variations can be seen. Particularly at the lower output performing tests and higher division rates such as D6 F5 S6 is 1.45 seconds faster than D6 F6 S5. Whilst D6 F4 S5 is a whole second slower again with 1 less function. This illustrates the potential non-linear speed increases through reduction of application complexity.

6.3.2.2 Metric Investigation

A sample of tests will be examined in further detail to investigate methods of measuring resilience according to the structural graph methods investigated. An initial analysis of tests which were largely unsuccessful (i.e. on average they have around 1 output message) indicates the same findings from the unsuccessful tests, in that where $func > sub$ or div is simply too low then the application has an extremely low or impossible chance to execute (in agreement with the findings from Chapter 4). Therefore the graph metrics in this instance do not wholly represent the resilience of the application in terms of successful application execution as they do not include information about the size of the service. Therefore this investigation focuses on test-cases which were successful.

D4 F2 S6 is selected as a sample test for analysis due to it being a high output performing test case. Plotting the output curve of one instance illustrates that performance is again variable (figure 6.13), the test was re-run for a longer duration (600 seconds) to illustrate the cyclic nature of the tests and that at these parameters the network can be seen to self-heal back to a resilient state although it is still cyclic in nature (figure 6.14). The rise and fall of the curves for both N and *connectivity* clearly illustrates the network structure losing nodes and self-healing as it does so. With the shorter test run it is also difficult to evaluate if the additional graph metrics indicate state change or not. While the frequency of the peaks in the output do coincide with the peaks of the assortativity and clustering graphs, it is difficult to determine at this larger scale. While the trend in incline does not relate, nor does the scale. While network criticality indicates little information after the initial network convergence. Network criticality, which is a known metric to describe network resilience, tales off and does not vary much in line with the output. This preliminary finding hints that traditional structural metrics may not be valuable to the resilience of this system as they do not include information relating to the application. Overall this illustrates that the graph metrics do follow similar trends to the output, such as timing and frequency of peaks, although with the additional networking perturbance it is difficult to view visually when compared to the networks in the 0 failure rate tests.

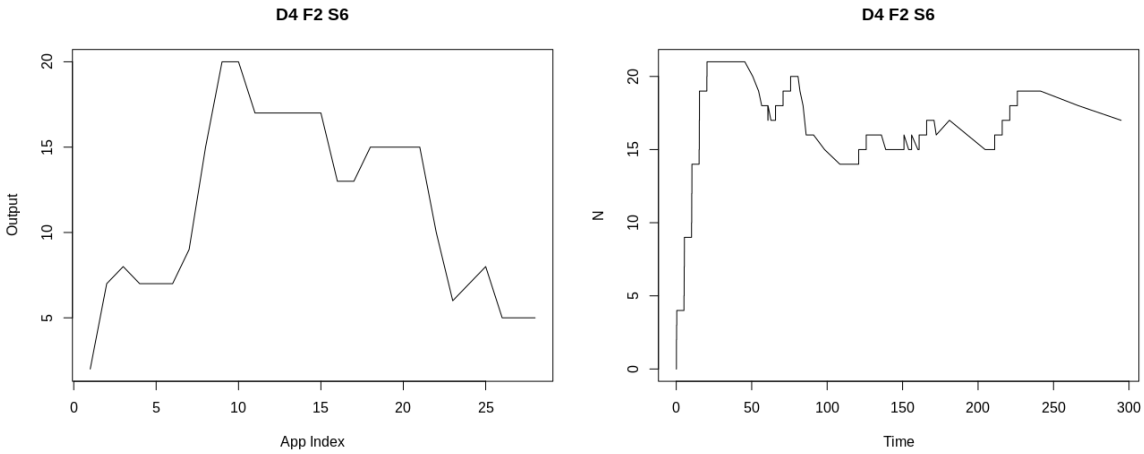


Figure 6.13: D4 F2 S6. The figure on the left indicates the application execution performance. The figure on the right illustrates the rate in change of nodes and therefore the network's ability to self-heal

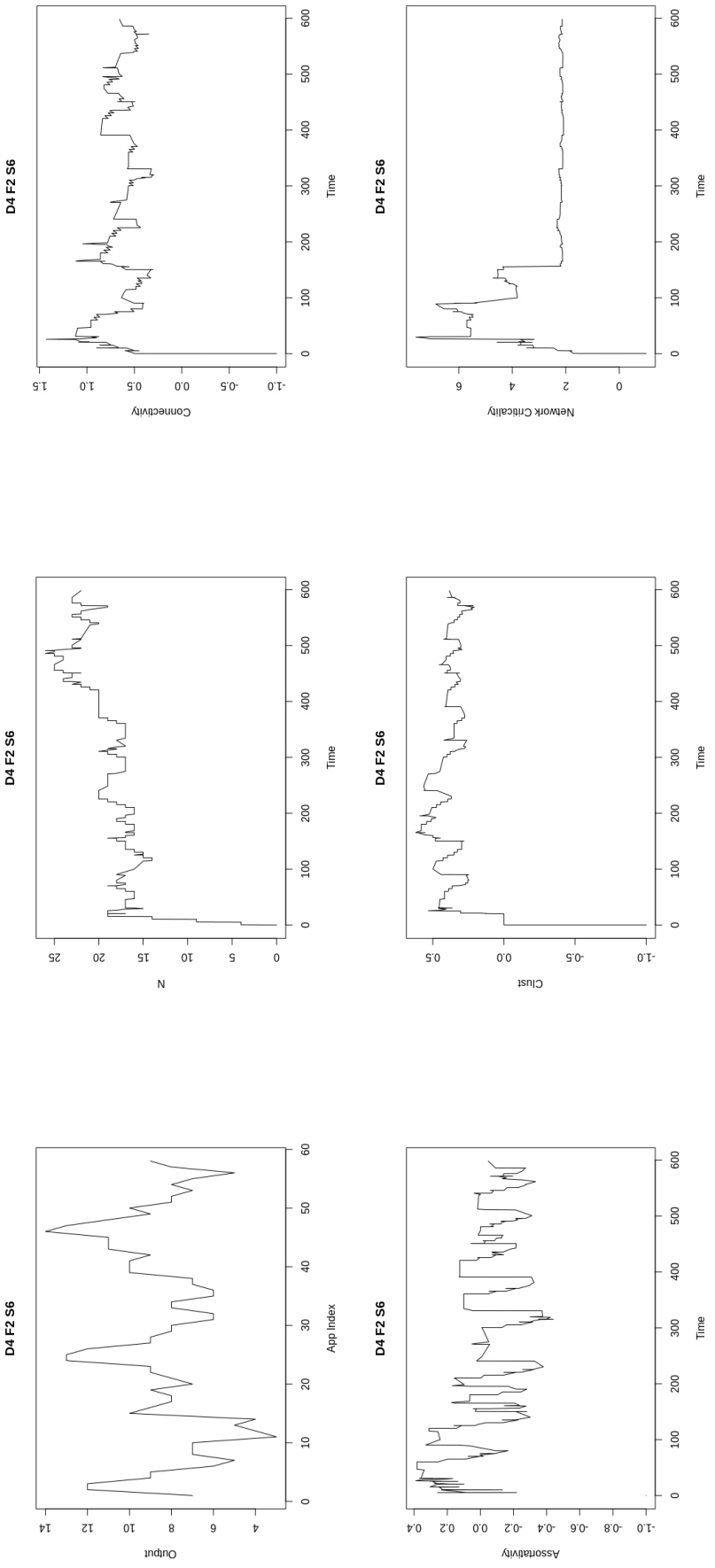


Figure 6.14: Output and relevant graph metrics for D4 F2 S6 for 600 seconds

D5 F2 S4 is evaluated as another seemingly successful test candidate. Having a lower ratio of process to receive messages with only 3 less inputs than D4 F2 S6 might indicate its efficiency. This time the initial test case length is doubled to provide further detail about the cycle. It illustrates that in a double length (600 seconds) only one cycle is observed. Figure 6.15 plots the test against a lower failure rate to illustrate the effectiveness of the self-healing at a lower failure rate. This illustrates the effect a higher failure rate has on the network's ability to create nodes and the stress the network is under.

Figure 6.16 therefore plots curves for output against relevant graph metrics for this test where the test run was executed for twice the previous length (20 minutes or 1200 seconds) to understand more about this variable system. Firstly, the output curve shows a reduction in output around application index 60 (approximately 600 seconds) which flat lines and then drops further instead of cycling back to an acceptable state. The system then drops to 0 output, before steadily returning. Examining the curve for N shows that just before this there was a drastic reduction in cells. This is likely due to a cascading of failure causing a number of cells to be removed. The connectivity of the network actually increases at around this period. While this might sound counter-intuitive of a network which is failing, understanding that many nodes may be weakly connected to one or two cells hierarchically means that a failure of a number of these cells would increase the connectivity of the network overall.

Assortativity, which has been noted as a prime candidate for accurately representing resilience in terms of application execution across different scales can be seen at this point (approximately 600 seconds) to be fundamentally changing from relatively stable oscillations to suddenly erratic oscillations, similarly (although less apparent) this can be seen in the clusters metric. This cross evaluation illustrates an interesting point in that the change in state could be measured by the uniformity of the oscillations which may be viewed from multiple metrics, where assortativity is the cleanest representation of this. This is confirmed through examining the periodicity of the spectrum and the variation in peak height. Where a low variability indicates a more uniform frequency and thus a stable system (one state) and a higher variability indicates a less stable system.

Figures 6.17 and 6.18 show the periodograms for assortativity in 100 second intervals for the test D5 F2 S4. Periodograms plot the power spectral density and therefore are an effective method for determining variability in a time series (Hernandez 1999). The first noticeable difference is the quantity of samples. As the data was sampled whenever a structural change occurred in the network, the quantity of different samples indicates how much the network changes. The greatest number of samples can be seen in the 1st graph, which represents seconds 0-100 and clearly the convergence of the network. From time 100-500 the graphs are fairly regular, which illustrates the stable state of the system. From seconds 500-1100 the state of the system is clearly variable at each period. Finally network criticality shows a clean drop at this time which could be considered in the same context as connectivity, in that the lower the more resilient the network is structurally.

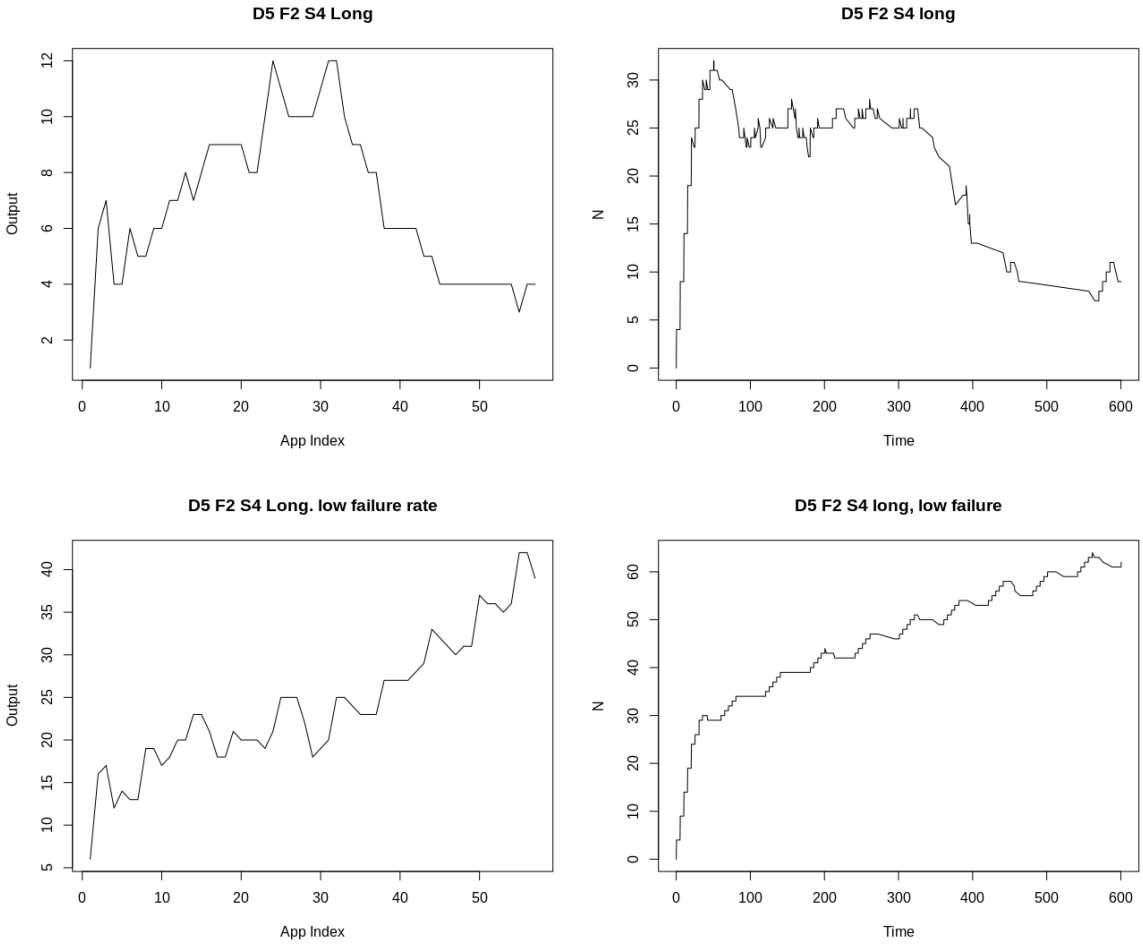


Figure 6.15: D5 F2 S4 for 600 timesteps The figure on the left indicates the application execution performance. The figure on the right Illustrates the rate in change of nodes and therefore the network's ability to self-heal. The figures on the bottom row are the same except for a lower failure rate to illustrate the difference in growth.

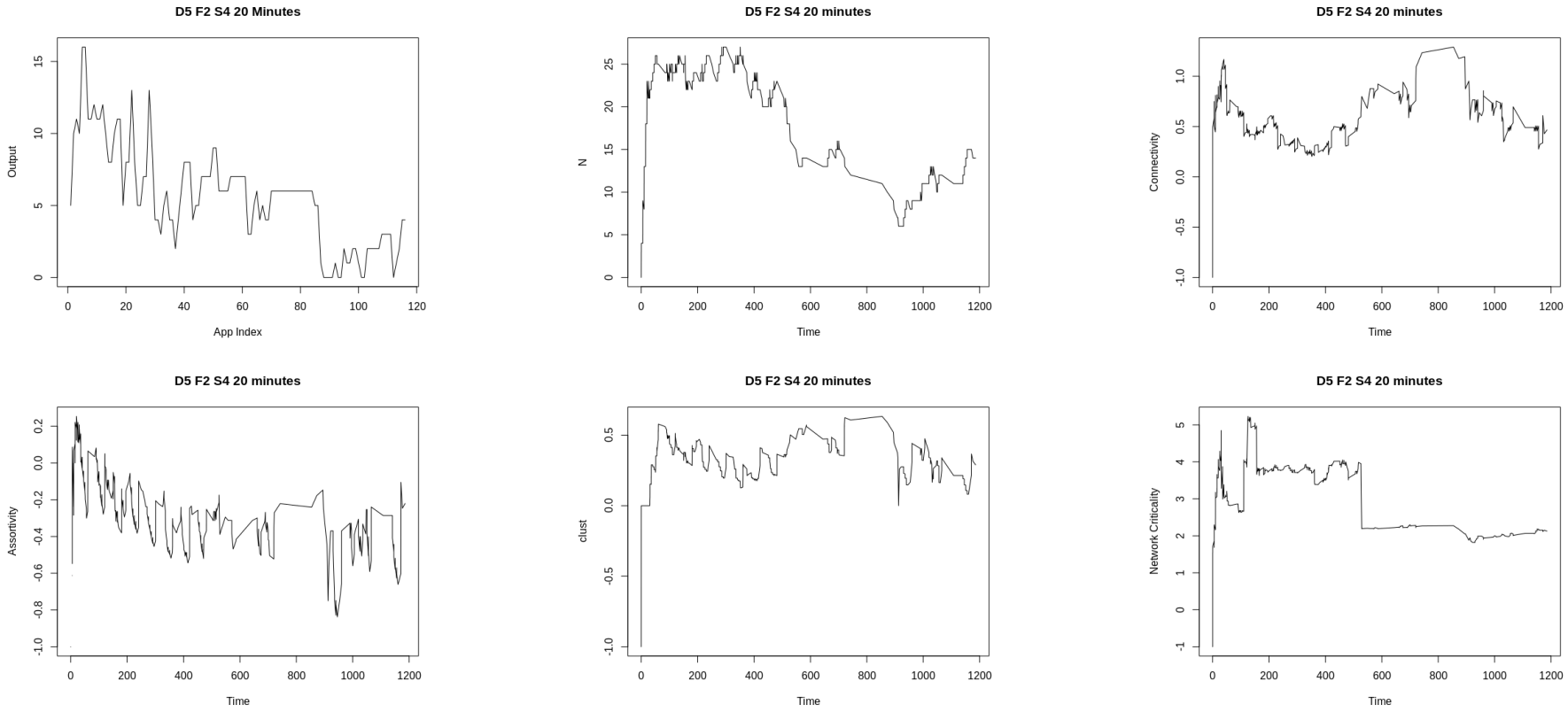


Figure 6.16: Output, N and connectivity along the top row and assortativity, clusters and network criticality for D5 F2 S4 running for 20 minutes (1200) seconds. A range of different metrics can be combined to understand further about the change in system state.

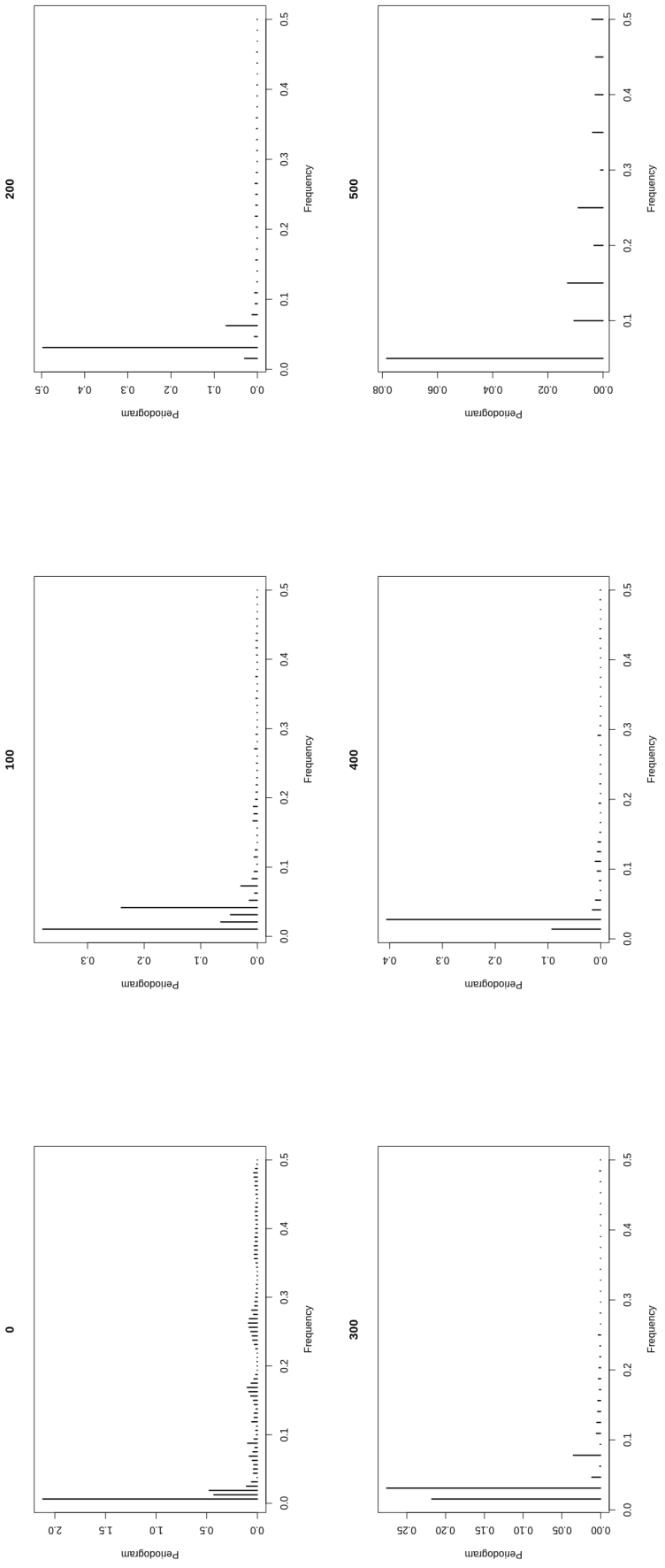


Figure 6.17: Periodograms of assortativity in 100second intervals for the test D5 F2 S4

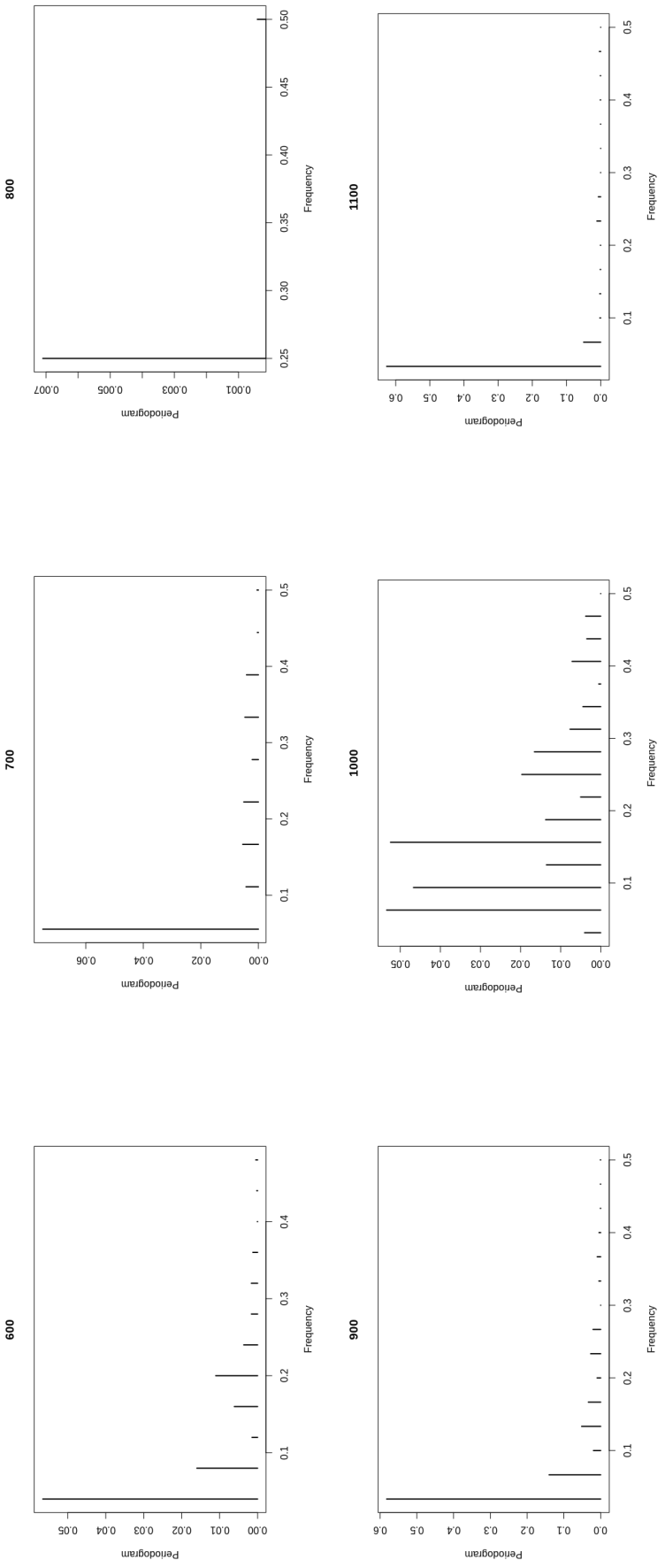


Figure 6.18: Periodograms of assortativity in 100second intervals for the test D5 F2 S4

D6 F2 S6 is the final test sampled for analysis due to its larger size and increased substitutions. Figure 6.19 plots the graph metrics against the output as per the previous test analysed. The output illustrates a steady decline and then a relatively consistent output before an increase again. Unlike the previous tests, the state changes are less clear when observing visually. The decline and then consistency in the quantity of nodes (at about 600 seconds) hints at a change and in performance and can be seen in the output also. An increase in network criticality at this time also illustrates this at about 600 seconds although reverts back to a baseline. Assortativity also illustrates a considerable change in peak height at around this time which unlike the other graphs this change is maintained, representing a distinction between two different system states.

This investigation follows the findings of the previous test, which highlighted that examining the spectral density of assortativity can possibly indicate state change within the system. Therefore figures 6.20 and 6.21 illustrate the periodograms for 100 second periods. The change in state at time period 600-900 can be clearly seen. Either side of this period, at 500 and 1000, the frequency is much more dispersed and could indicate a state transition period.

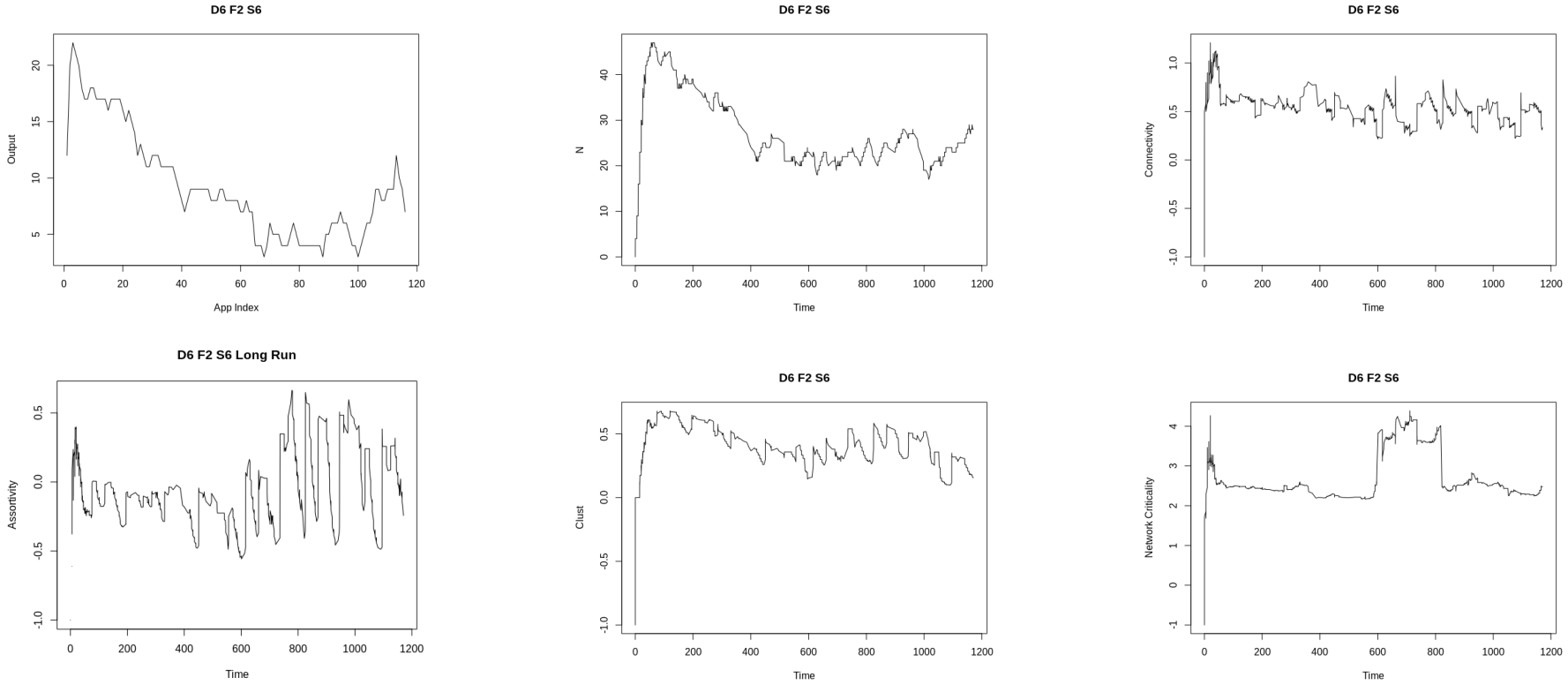


Figure 6.19: Output, N and connectivity along the top row and assortativity, clusters and network criticality for D6 F2 S6 running for 20 minutes (1200) seconds. A range of different metrics can be combined to understand further about the change in system state.

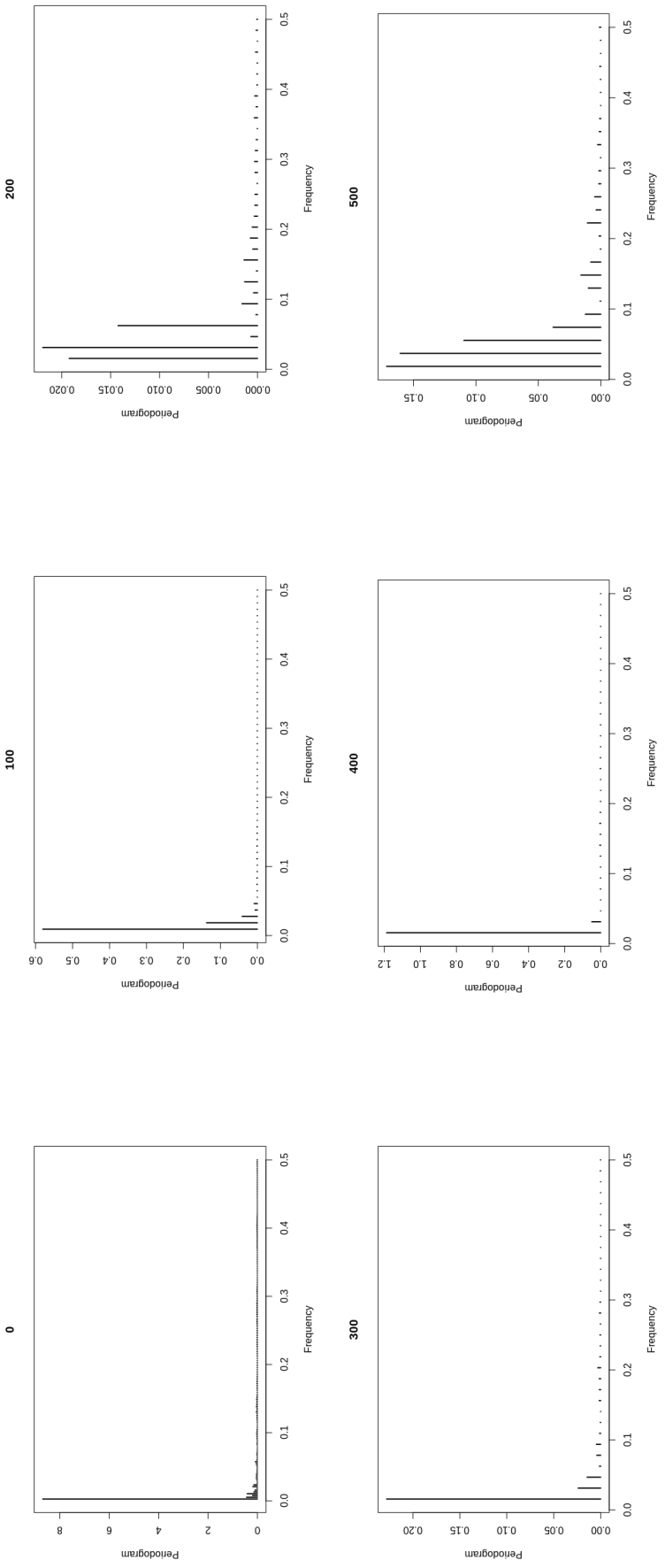


Figure 6.20: Periodograms of assortativity in 100 second intervals for the test D6 F2 S6

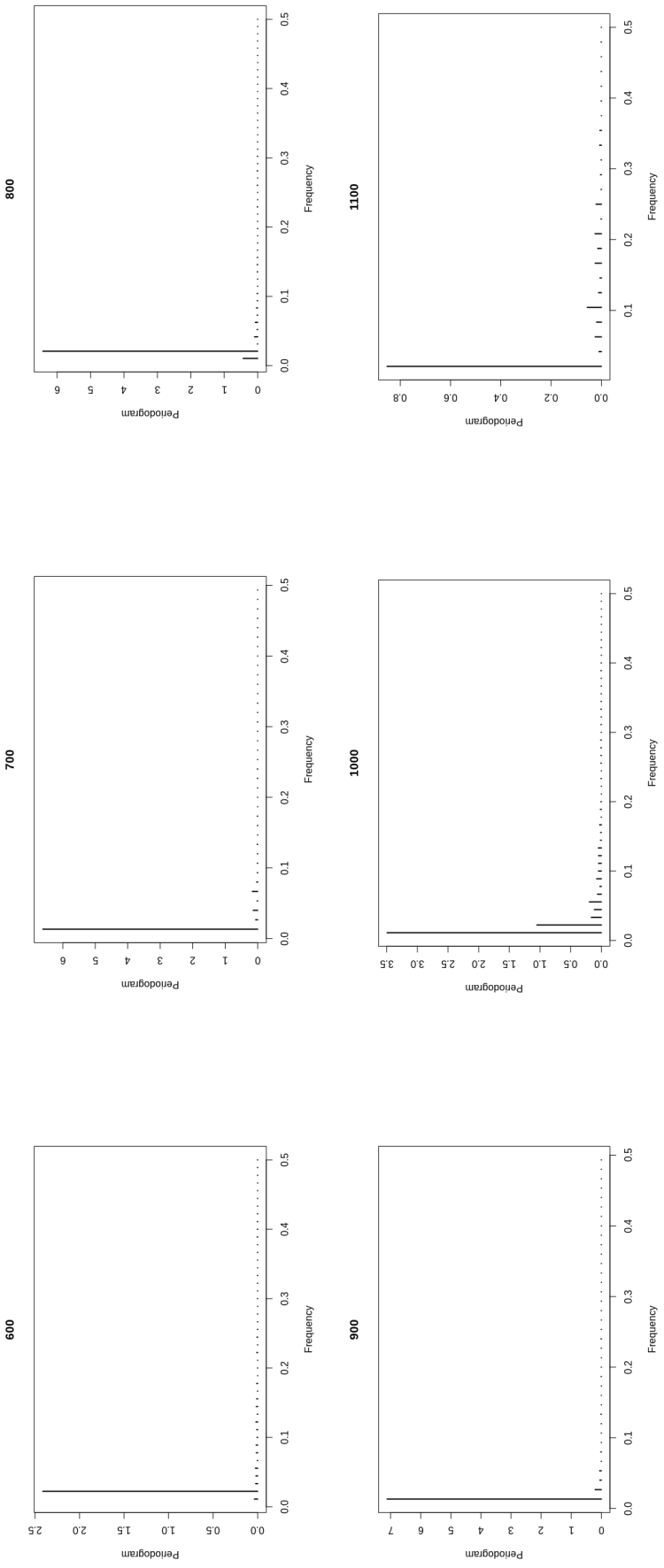


Figure 6.21: Periodograms of assortativity in 100 second intervals for the test D6 F2 S6

D5 F5 S7 Some test cases were borderline. i.e. on average 1 application was successfully executed. On closer examination these are due to bursts of activity which indicate some but not consistent success. Figure 6.22 shows the relevant graph metrics for test case D5 F5 S7. Given that $func < div$ one might consider this application execution to be successful but this is largely not the case. In this instance the general trend for the graph metrics does not indicate much in terms of determining persistent execution success or failure relative to the successful applications. However without plotting the periodograms for assortativity considerable variation in the spectral density can be seen between each peaks. If the hypothesis that assortativity is suitable for measuring state change is to be believed, this would strongly indicate high variability in system states and therefore a chaotic system.

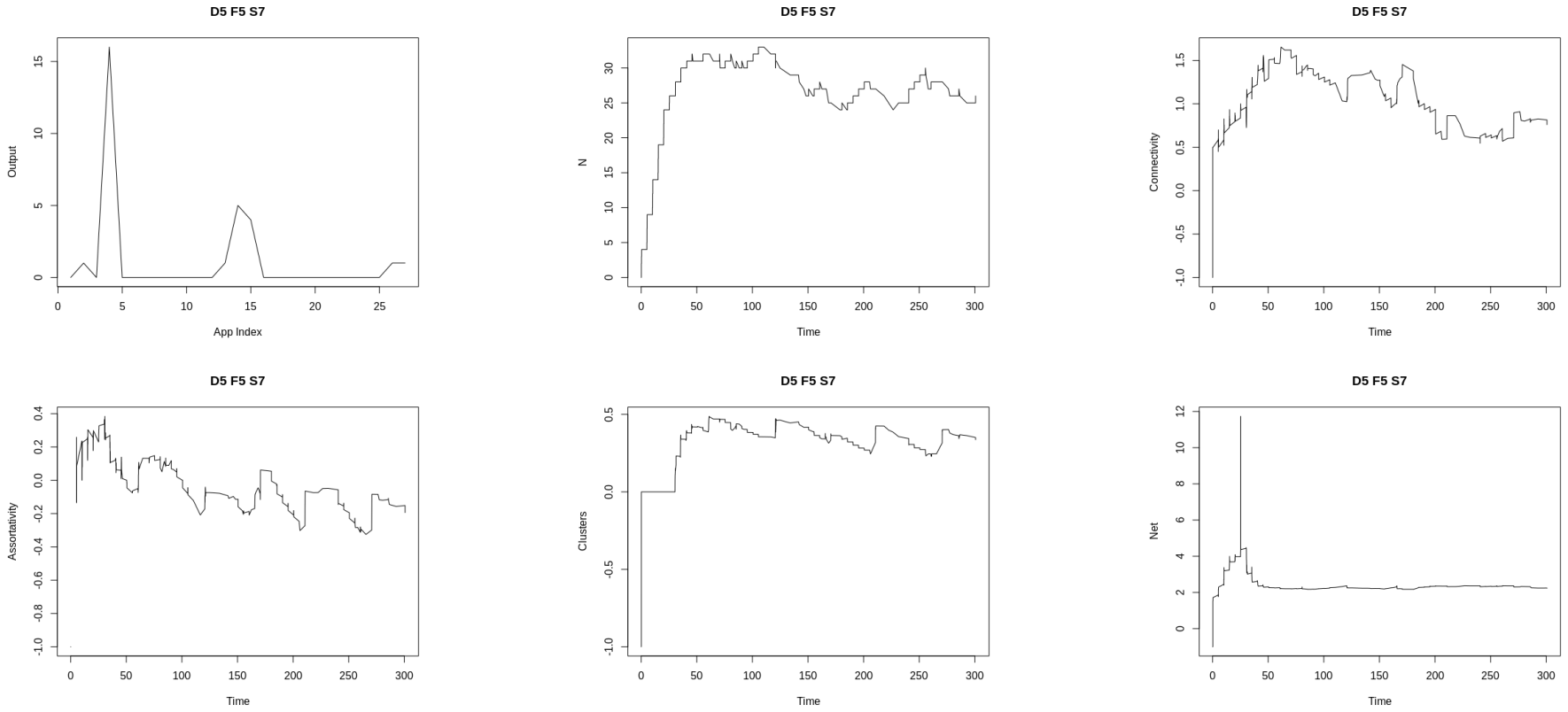


Figure 6.22: Output is plot against relevant graph metrics for D5 F5 S7. A range of different metrics can be combined to understand further about the change in system state during this test with borderline success.

6.4 Discussion

The previous section presented and analysed the test results for the embryonic cloud platform proof-of-concept from two separate groups of tests both of which where used to analyse the system performance. The first where there was no failures instigated and thus the network was not stressed, and therefore did not instigate self-healing. The second was where a high rate of node failures caused the system to be perturbed and therefore instigated self-healing. The first group was conducted to understand more about the network structure whilst the second was mostly conducted to investigate methods of resilience measurement and validate the self-healing functionality. This section provides a discussion of the results in the context of the objectives of this work.

6.4.1 Application Performance

The performance of the application execution was conducted to evaluate the success of the proof-of-concept in delivering its intended goal. i.e. is an application still delivered within the face of external and internal changes to the system successfully, and what are the constraints involved? It is important to note prior to this discussion that the results of this performance analysis are representative of this proof-of-concept only and should therefore be used as a guideline for performance of a production ready implementation of the system. This point is particularly relevant to the timings of the application performance. As the experiment was in a controlled environment, delays as a result of network hardware or links were not introduced to the tests. This factor is useful for empirical consistency but less representative of a real-world environment. Additionally the implementation of the application has not been tuned for performance purposes. However the results are still representative of the architectural model and thus provide insight into performance to enable selection of particular parameters according to use-case.

6.4.1.1 Communication Complexity

The number of messages transmitted in order to execute the application is relevant to the performance of the system as an increase in messages both transmitted and received has a direct effect on the cost of executing the application. Messages transmitted over a network link create contention between other applications whilst messages both sent and received must be processed in one form or another. Using the publisher-subscribe model employed by the embryonic architecture ensures that a message will not be processed by the cell it is not intended for, however the message may still be received by the system and will therefore be processed by the operating system's network stack and the initial layers of the application in one form or another. Therefore the reduction in message complexity is essential to the performance of the system. However an increase in messages does provide redundancy in the system and therefore could increase resilience. Given the above, a trade-off is required between the

probability of application success and any resource strain on the system. Fundamentally this decision is down to the users of the system and the particular use case employed. In this context, the following results were found:

- In the 0 failure rate tests, at the higher end of the output scale, the ratio of messages received to output is extreme, where for approximately 60 output messages there were approximately 600 received. This could be considered a waste of resource usage particularly for constrained resource environments.
- In the tests involving node failures, the resource wastage is drastically reduced to approximately 90 messages received to 13 output. Moreover, the highest tests also performed less well in the failure rate tests than those with no failure rate. While intuitive, this indicates that a key negative aspect, the excess message redundancy, is appropriate for the architecture operating in more hostile conditions.
- Even in tests where no application executed successfully, there was still a large ratio of messages received. This is due to the subscriptions creating more links between cells and the divisions increasing the connectivity of the network and therefore enabling messages to still be passed. The lack of output was due to the complexity of the application (function size) being the constraining factor here. This strengthens the argument for adapting complexity according to the environment, where possible.

To summarise the conclusions of these results, it was shown that larger networks created drastically increased communication complexity, which is intuitive. These results served the investigation well, as it was defined during the previous investigation that it would be necessary to evaluate the performance trade off. In this instance it was shown that the increased redundancy was used effectively to provide increased resilience to networks where nodes failed. Additionally the ability to successfully output the message was largely constrained by the quantity of functions combined with the subscription value.

6.4.1.2 Temporal Performance

In terms of temporal performance of each application, the results illustrated were variable, which was intuitive. The increase in network size would increase overall processing time and the increase in application complexity would also increase processing time with some clear performance gains to be made through a slight reduction in function type. As mentioned previously it should be noted these results are purely representative of this proof-of-concept with many optimisations not included, particularly as it was developed in a high-level language. The strongest finding in these results is that the intuitive results validated the implementation.

6.4.2 Measuring Resilience

The second motivation for undertaking an analysis of the test results is to further provide insight into methods of assessing the resilience of the system, with a particular aim of permitting comparison between disparate cloud architectures. The proposed solution to this is discussed in this section.

During the literature review stage, a number of different types of resilience metrics were discovered and classified from literature (section 2.4.6). A number of graph metrics were chosen to analyse the structural resilience of the network, as these provide ease of comparison between architectures whilst also summarising the current state of the network structure in a succinct manner. According to the evaluated literature, no metric had been converged upon as definitive as performance varied according the underlying network structure and purpose. Therefore a subset of metrics were selected according to certain criteria and applied to every stage of the network in each test during this research. The following insights were gained from this analysis:

- The structural graph metrics chosen did provide a description of the network at each stage which could be compared and contrasted against other networks for comparison. Often they would corroborate the state of the network (e.g. through similar trends) and for this reason allowed insight into the network at that point in time.

- As the metrics described the structure and did not contain information about the application execution, on the whole their relationship with resilience did not always relate to whether an application was successfully executed. This highlights the contrast between service and structural resilience. Therefore it was also important to understand if other constraints were in place (e.g. high application complexity) to determine if an application would execute successfully, prior to attempting to run it on the embryonic platform.

- The metrics employed specifically for resilience, and not simply for description, which were network criticality and effective graph resistance, did not correlate (on the whole) with the resilience of the network. If previous literature is to believed this means that the network is successfully executing applications and self-healing despite the structure being traditionally non-resilient.

- Assortativity showed the most promise in correlating with successful application output in a manner which illustrated a change in system state. As the changing peaks on the graph were roughly correlated with the level of execution success on the output graphs. Further spectral analysis using periodograms highlighted this to be true.

These points highlight a key finding of this work, which is that the embryonic platform is not structurally resilient when compared to traditional architectures using standard metrics. However,

the ability for applications to still be delivered, despite this low level of structural resilience in the network indicates the self-healing and service-resilience of the system is operating correctly.

6.4.2.1 Validating Assortativity Periodicity for State Change Measurement

The result of assortativity being a candidate for representing the quantity of application output and therefore the variability in system state was an informative finding. It links two of the metric categories of graph metrics and state measurement. State measurement is deemed suitable as it is also grounded in complex systems theory. Therefore, these concepts will be combined to determine the particular state of resilience the architecture is in at each point in time.

In this section assortativity is therefore validated as a method for measuring state change. A proven method of determining state change is a a recurrence plot (Eckmann et al. 1995) which is an advanced method often used in chaotic systems analysis to visualise the recurrence of the system to a particular state. Taking a timeseries of data as an input, the function plots a matrix where each element is a point at which a recurrence to the state occurs (Marwan 2008). As recurrence plots are a proven method for determining system state (Marwan et al. 2007), its concurrence with the periodicity of the assortativity would suggest its suitability for measuring state change. Note that recurrence plots are a less suitable method for extemporaneous measurement as they are resource intensive and require capture of the full time series, whereas potting the periodicity of the graph can be conducted on smaller samples and is less resource intensive.

Whilst in the previous section the periodograms for assortivity were plotted against time, these periodograms are plotted against the index of changing graphs to more easily compare against the recurrence plots. For this reason they contain uniform frequency as opposed to the variable seen previously.

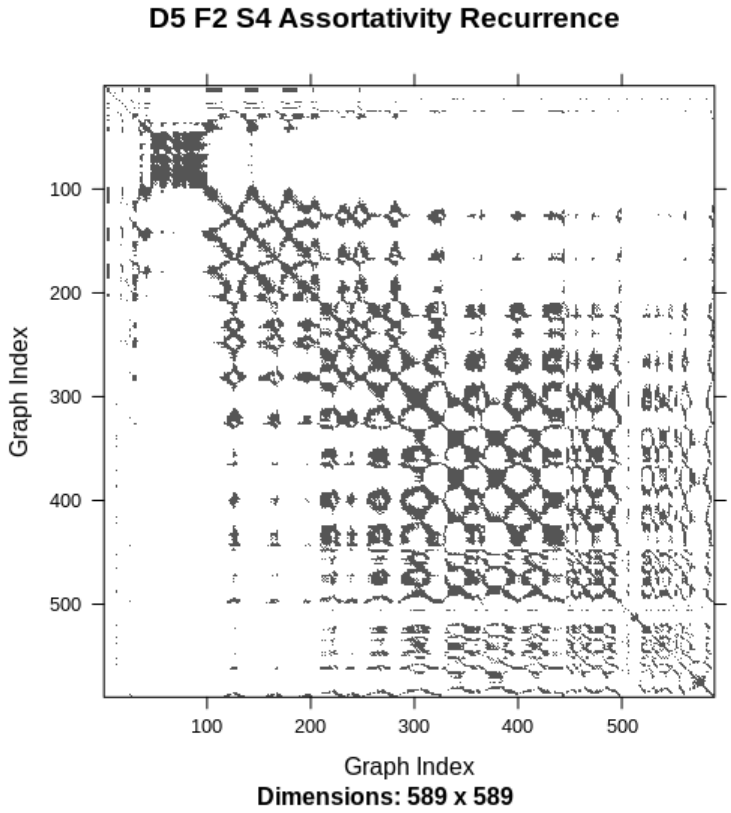


Figure 6.23: D5 F2 S4 Assortativity Recurrence Plot

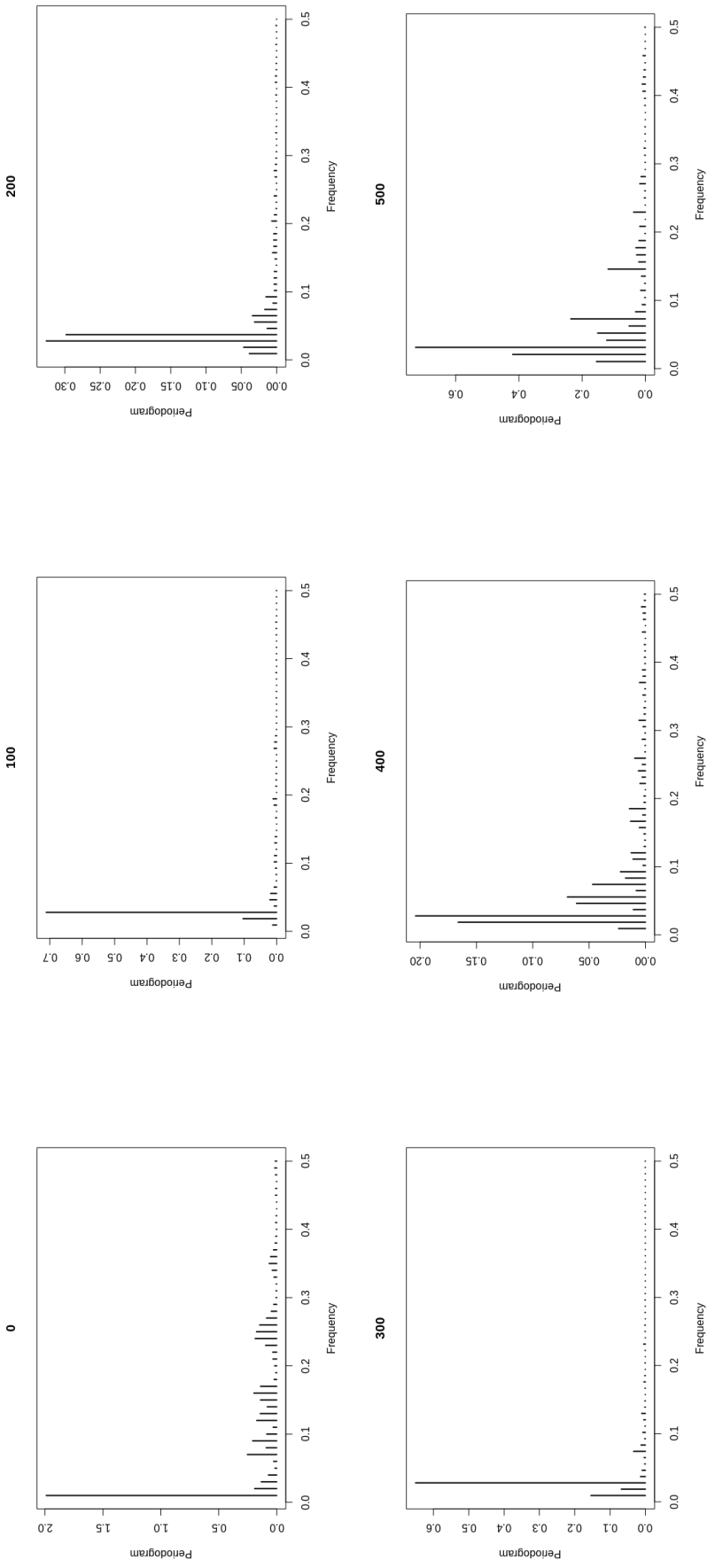


Figure 6.24: D5 F2 S4 Periodograms of every 100 graph changes. The label at the top of each graph indicates the starting graph index.

Figure 6.23 presents the recurrence plot for the test D5 F2 S4 and figure 6.24 shows the periodograms in 100 graph changes. On the recurrence plot, the graph index between approximately 50-100 shows a very condensed area which is individual and does not recur across the rest of the system. When compared to the periodogram starting at index 0, a similar variation can be seen through the long frequency distribution indicating high variety. The periods between 100-200 graph index indicates a particularly consistent time recurrence which is confirmed by the low variability in the frequency of the periodogram starting at index 100. Following from this, the period from approximately 200-450 is divided into two distinct but similar states. A new state appears to be seen from 350-450, which is confirmed in figure 6.25 which plots the periodogram for this period and shows a low variability in frequency. The final period of the graph shows low uniformity in recurrence, although clearly indicates a different state at 450-600 which is confirmed by the variable frequencies in the 400 and 500 periodograms.

In the previous analysis it was shown that assortativity could be used to determine the change in system state, which correlated with the variety in output messages and therefore the resilience of the system at that point. This analysis has shown that a known method for complex system state measurement clearly illustrates that the periodicity of the assortivity is comparable. Therefore this seeks to validate this method for use in measuring the state of the embryonic system.

6.4.2.2 Proposed Method

This section describes the proposed method of measuring the resilience of the system through analysis of the current state. As discussed in section 5.8, a number of states can be seen in the context of resilient service delivery. If the given metric can provide insight into the particular state of the system then the resilience can be determined. According to the given findings a method for determining the resilience of a particular system is given below. A model of each system under parameters will need to first be developed to provide a baseline for comparison. This will them allow selection of an appropriate graph sample size and a characteristics of each system state.

The following steps can then be conducted:

1. The characteristics of the system are chosen within the constraints of *div* and *sub* to maximise success.

2. The graph structure of the system is sampled when a change occurs. A set of graphs is compiled through an aggregation according to a sample rate which is predefined according to the size and rate of change of the network.

3. The assortativity of each graph in the set is calculated and then plotted to illustrate the change in assortivity.

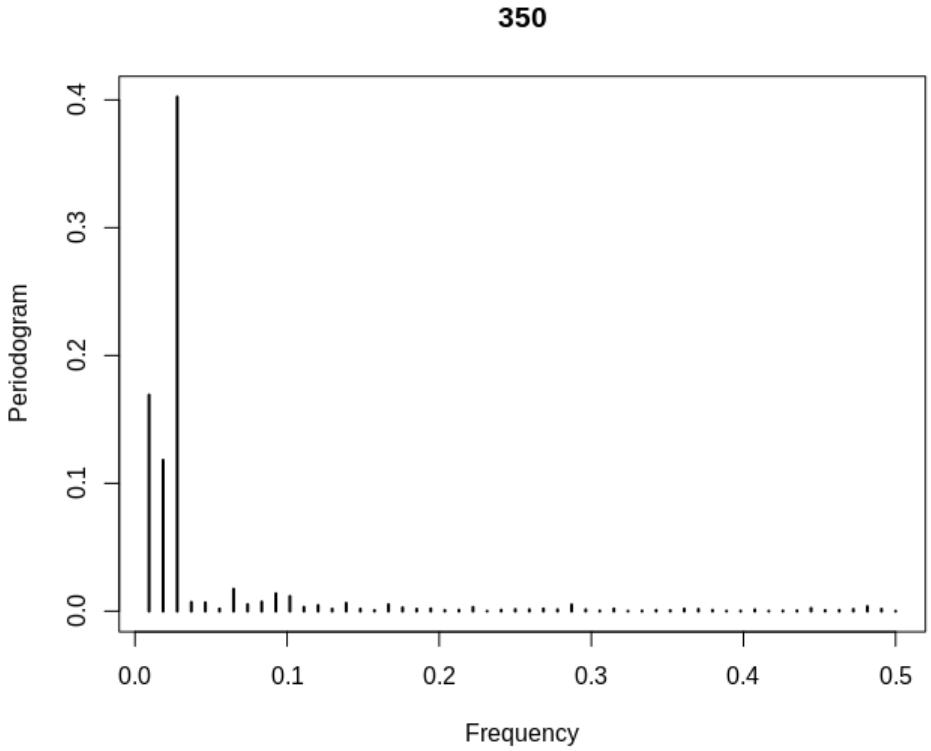


Figure 6.25: D5 F2 S4 Assortativity Recurrence Plot

4. Spectral analysis is then conducted on this assortativity curve to determine different states. In the investigation conducted in this research, periodograms are employed to determine the difference states. Other spectral analysis techniques may be employed for accuracy.

5. The determined state then indicates whether the system is currently operating in a degraded, failed or acceptable manner.

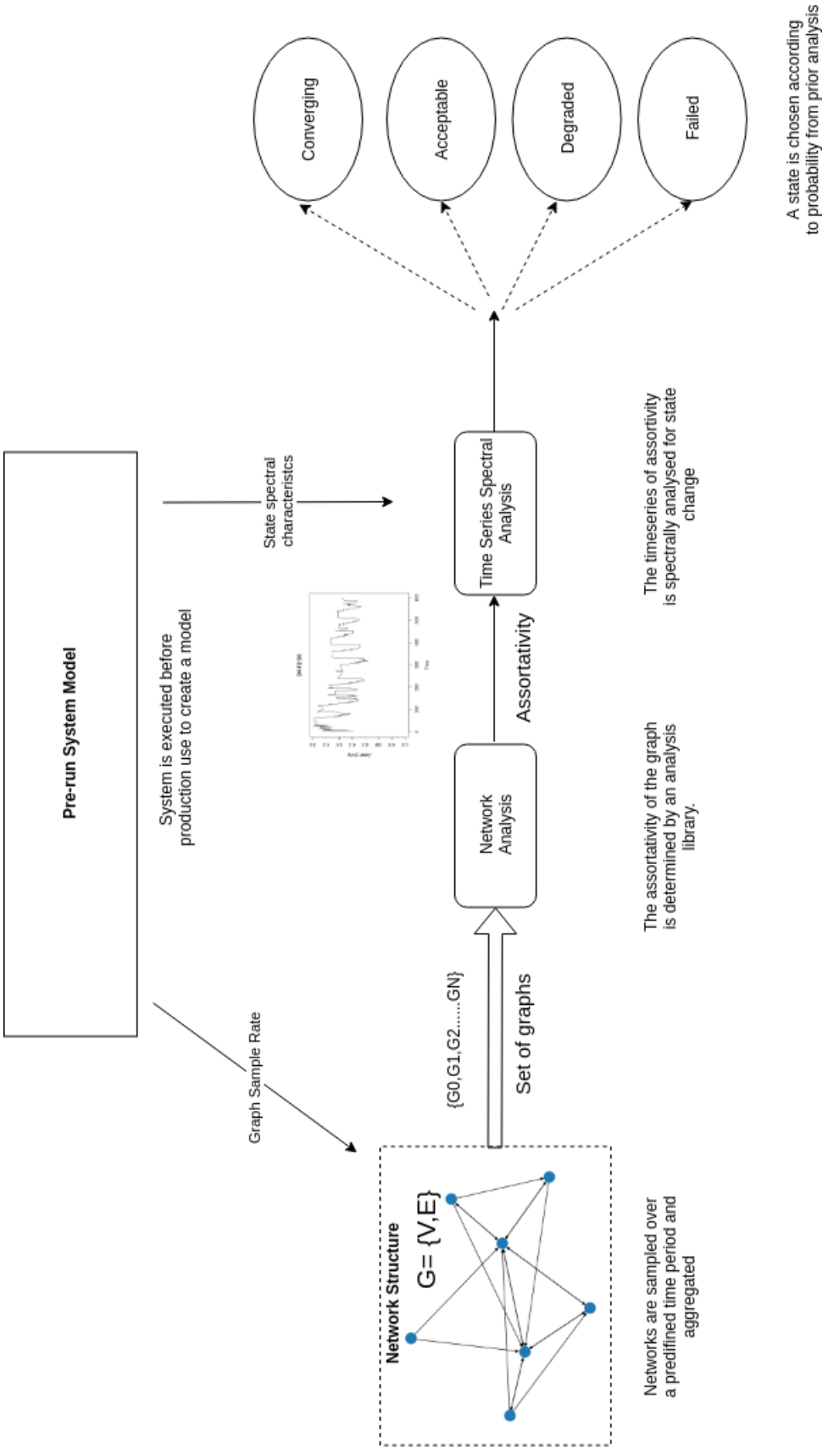


Figure 6.26: The proposed method for measuring system state change to determine resilience.

6.4.2.3 Further Comments

The resilience measurement method defined in this section was the result of a combination of resilience methods defined in literature. Many traditional graph metrics were deemed unsuitable during experimentation due to their lack of correlation or information involving application complexity. One graph metric, assortativity, appeared to indicate changes in system state. Further validation using recurrence analysis showed this method to be accurate. Therefore a linear method of initial model creation and then system sampling leads into a determination of system state, which in turn determines the system resilience. This method follows both graph-based and state-based resilience measurement methods found in literature.

6.5 Summary

This chapter presented the final stage of the investigation into the resilience of embryogenesis inspired resilient cloud architectures. After the implementation of a proof-of-concept which was deemed to have met the resilience and functional requirements previously defined, batches of tests were executed to understand further about the performance and the resilience of the system. A number of findings were determined involving the effect of system parameters on the structure and resilience of the system, application execution success under different system parameters and finally a method to determine the system state according to spectral analysis of the assortativity.

Chapter 7

Conclusion

7.1 Introduction

This chapter concludes the research in this thesis through revisiting the hypothesis and evaluating it against the resulting research contributions of the work. This then highlights limitations in the work which leads to proposals for future research directions. It concludes through highlighting the contributions to knowledge.

7.2 Revisiting the hypothesis

The aim of this work was to test the hypothesis "A P2P architecture, with characteristics inspired by embryonic development, will provide persistent decentralised cloud service delivery in the face of system perturbations." This was conducted in response to the emergence of new, low-latency use-cases which require cloud data processing yet in hostile environmental conditions. The motivation for this was discovered and documented in chapter 2.

To accomplish this, in chapter 3 firstly the requirements for both resilient service delivery and decentralised cloud computing were derived. After background reading into the self-healing and communication characteristics of animal embryonic development, feature mapping successfully occurred between these concepts. This culminated in a proposed architectural model for resilient decentralised cloud service delivery.

The key variables which may affect performance and resilience concerned with this architecture were then investigated under simulation in chapter 4. Cellular Automata were employed as the simulation tool due to their applicability for decentralised, complex systems. After varying both internal parameters (such as the application complexity, the aggressiveness of the cellular division and optimal differentiation algorithms) and the external environment which caused nodes to fail, a number of interesting findings emerged. These included the importance of initial starting conditions

which illustrated that once a system had reached a period of convergence, it would persist despite the hostile conditions. Methods for improving resilience were also shown such as through improving self-organising optimisation and network hop count. Finally, analysis during this simulation hinted that the system cycled through different states, which may be used later for measuring its change in resilience capacity.

After the efficacy of the approach was illustrated through simulation, a proof-of-concept implementation was realised in chapter 5. It was shown during this work, in order to meet the initial requirements, two different variants were proposed. One which responded to user-requests and which was able to self-organise according to the requested functionality. However it became time-constrained according to these prompts, delaying self-healing behaviour. While another variant was entirely autonomic but provided less flexibility in terms of applications processed. The fully autonomic variant was selected as the most applicable to the decentralised cloud use-case due to its quicker self-healing ability. However the user-driven variant would still be applicable to small-scale network environments.

In chapter 6, the selected autonomic embyronic implementation was empirically evaluated to understand more about its performance characteristics. It was shown that the system, which was resilient by design, successfully provided persistent service delivery despite continued node failures. Redundant messages were noted as being quite high, which while beneficial for a high level of resilience, may have a negative impact upon network bandwidth. During this experimentation, network structures were also examined for their resilience against common quantifiable metrics from literature. It was highlighted that the network appeared to not be structurally resilient, according to the resilience metrics taken from literature. However this illustrated that it was able to continue to deliver services resiliently despite its low structural resilience. During the testing of these metrics, it was finally highlighted that the periodicity of assortativity could be employed as a measure of the system's current state and therefore could be employed as part of a metric for later comparison.

Overall, this thesis was successful in answering the hypothesis. It illustrated that through the design and development of a service delivery system based upon the self-healing and self-organising characteristics of embyronic development; the developed system would be resilient by design. Empirical evaluation of the performance and resilience criteria highlighted this to be the case.

7.3 Challenges and Limitations

A number of challenges and limitations were posed to this work and are described below:

- Some limitations are a result of the technical direction of the work. For example the measurement of resilience is conducted in the context of application delivery when analysing technical factors. However the the cost of delivering the service is vital to on-going continuation (and therefore resilience) of the service yet is excluded from the initial hypothesis. Non-technical factors could

be considered in future models of resilience.

- The implementation of the work, while representative of the proposed architecture, is a proof-of-concept. It therefore provides quantitative investigation of different resilience characteristics but the performance variables could change dependent upon further performance tuning. Additionally the tests were conducted in a highly controlled manner to provide for fair and consistent comparison between test cases, this mean that the results, while comparable, were not truly representative of the diverse environments available due to the lack, for example, of link delays between nodes. However given that the research was more strongly focused upon execution success (resilience) than performance this was not deemed to be a crucial factor.

- A recurring problem during the empirical investigation of different concepts in this was the scalability of data analysis. During the the CA, probabilistic modelling and proof-of-concept evaluation, it was quickly realised that small increases in network parameters created large jumps in dataset pre-processing times. This issue contributed to continued delay during experimentation (partially due to the lack of available computational resources) and many blocks of code had to go through optimisation to provide concurrency in order to reduce processing times. Consequently, in the interests of empirical consistency whole groups of tests were re-run, delaying this phase of the work.

7.4 Future Work

Future work could be extended in multiple areas which are defined below:

- To further evaluate the efficacy of this software for real-world use-cases. The proof-of-concept can be developed further, particularly with software optimisations. It could then be tested in a number of real-world scenarios which involved physical hardware and real delays between nodes. The previously defined examples of VANETs and drones would be well suited for this.

- The investigation of the metric highlighted that spectral analysis of certain graph metrics can indicate system state change. The chosen method, assortativity, was valuable in this instance and may be for similar systems. Therefore its efficacy should be evaluated further for other complex system types. This could include complex systems of a non-embryonic type.

- The user-driven proof-of-concept variant was not investigated further during the investigation of this thesis due to lesser alignment with the theoretical foundations of complexity theory. However its ability to provide resilience in smaller scale systems should still be evaluated for use-cases where it is more applicable.

7.5 Contributions to Knowledge

To summarise the findings of this thesis, the following contributions to knowledge can be identified. [1]

- A new classification for cloud computing components and disciplines was defined and a comprehensive survey which analysed the resilience disciplines and components affected per layer was presented. (Appendix B Paper 1)
- A simulation of the decentralised model determined that local-only communication could be leveraged for resilient service delivery under certain parametric constraints. This work produced a dataset and was published with the focus of real-world use-cases (Appendix B Paper 3) (Welsh & Benkhelifa 2019).
- A proof-of-concept implementation (complete with experimentation platform) of the architecture is released as opensource for public use. It confirmed the effectiveness of an embryonic cloud model for delivering resilient services (Appendix B Paper 4).
- A dataset was produced during an empirical evaluation of the embryonic platform which is released for public use. An analysis of this dataset determined that while many structural graph metrics were not suitable for measuring resilience in this instance due to the lack of information concerning the application, spectral analysis of assortativity could be leveraged as a method for measuring the change in system state and therefore its performance.

[1] All source code and links to data sets can be found at : https://github.com/tomwelsh/embryonic-cloud

Bibliography

Aazam, M., Zeadally, S. & Harras, K. A. (2018), 'Deploying fog computing in industrial internet of things and industry 4.0', *IEEE Transactions on Industrial Informatics* **14**(10), 4674–4682.

Ai, Y., Peng, M. & Zhang, K. (2018), 'Edge computing technologies for internet of things: a primer', *Digital Communications and Networks* **4**(2), 77 – 86.
URL: *http://www.sciencedirect.com/science/article/pii/S2352864817301335*

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M. & Ayyash, M. (2015*a*), 'Internet of things: A survey on enabling technologies, protocols, and applications', *IEEE Communications Surveys Tutorials* **17**(4), 2347–2376.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M. & Ayyash, M. (2015*b*), 'Internet of things: A survey on enabling technologies, protocols, and applications', *IEEE communications surveys & tutorials* **17**(4), 2347–2376.

Alenazi, M. & Sterbenz, J. (2015*a*), Comprehensive comparison and accuracy of graph metrics in predicting network resilience, *in* 'Design of Reliable Communication Networks (DRCN), 2015 11th International Conference on the', pp. 157–164.

Alenazi, M. & Sterbenz, J. (2015*b*), Evaluation and improvement of network resilience against attacks using graph spectral metrics, *in* 'Resilience Week (RWS), 2015', pp. 1–6.

Ali, A., Elsaadany, M. & Hamouda, W. (2017), 'Cellular lte-a technologies for the future internet-of-things: Physical layer features and challenges', *IEEE Communications Surveys and Tutorials* **19**(4), 2544–2572.

Amin, R., Ripplinger, D., Mehta, D. & Cheng, B. (2015), 'Design considerations in applying disruption tolerant networking to tactical edge networks', *IEEE Communications Magazine* **53**(10), 32–38.

Aral, A. & Brandic, I. (2018), Dependency mining for service resilience at the edge, *in* '2018 IEEE/ACM Symposium on Edge Computing (SEC)', pp. 228–242.

Araujo Neto, J. P., Pianto, D. M. & Ralha, C. G. (2018), An agent-based fog computing architecture for resilience on amazon ec2 spot instances, *in* '2018 7th Brazilian Conference on Intelligent Systems (BRACIS)', pp. 360–365.

Asghar, M. R., Dán, G., Miorandi, D. & Chlamtac, I. (2017), 'Smart meter data privacy: A survey', *IEEE Communications Surveys Tutorials* **19**(4), 2820–2835.

Atzori, L., Iera, A. & Morabito, G. (2010), 'The internet of things: A survey', *Computer Networks* **54**, 2787–2805.

Baccarelli, E., Naranjo, P. G. V., Scarpiniti, M., Shojafar, M. & Abawajy, J. H. (2017), 'Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study', *IEEE Access* **5**, 9882–9910.

Baktir, A. C., Ozgovde, A. & Ersoy, C. (2017), 'How can edge computing benefit from software-defined networking: A survey, use cases, and future directions', *IEEE Communications Surveys Tutorials* **19**(4), 2359–2391.

Baran, B. & Sosa, R. (2000), A new approach for antnet routing, *in* 'Proceedings Ninth International Conference on Computer Communications and Networks (Cat.No.00EX440)', pp. 303–308.

Beineke, L. W., Oellermann, O. R. & Pippert, R. E. (2002), 'The average connectivity of a graph', *Discrete Mathematics* **252**(1), 31 – 45.
URL: *http://www.sciencedirect.com/science/article/pii/S0012365X01001807*

Bellavista, P., Berrocal, J., Corradi, A., Das, S. K., Foschini, L. & Zanni, A. (2019*a*), 'A survey on fog computing for the internet of things', *Pervasive and mobile computing* **52**, 71–99.

Bellavista, P., Berrocal, J., Corradi, A., Das, S. K., Foschini, L. & Zanni, A. (2019*b*), 'A survey on fog computing for the internet of things', *Pervasive and Mobile Computing* **52**, 71 – 99.
URL: *http://www.sciencedirect.com/science/article/pii/S1574119218301111*

Ben-Jonathan, N. & Liu, J.-W. (1992), 'Pituitary lactotrophs endocrine, paracrine, juxtacrine, and autocrine interactions', *Trends in Endocrinology & Metabolism* **3**(7), 254–258.

Benkhelifa, E., Pipe, A. & Tiwari, A. (2013), 'Evolvable embryonics: 2-in-1 approach to self-healing systems', *Procedia CIRP* **11**, 394 – 399. 2nd International Through-life Engineering Services Conference.
URL: *http://www.sciencedirect.com/science/article/pii/S2212827113005027*

Benkhelifa, E., Welsh, T. & Hamouda, W. (2018), 'A critical review of practices and challenges in intrusion detection systems for iot: Toward universal and resilient systems', *IEEE Communications Surveys Tutorials* **20**(4), 3496–3509.

Benson, K. E., Wang, G., Venkatasubramanian, N. & Kim, Y. (2018), Ride: A resilient iot data exchange middleware leveraging sdn and edge cloud resources, *in* '2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)', pp. 72–83.

Bilal, K., Khalid, O., Erbad, A. & Khan, S. U. (2018), 'Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers', *Computer Networks* **130**, 94 – 120.
URL: *http://www.sciencedirect.com/science/article/pii/S1389128617303778*

Biswas, A. R. & Giaffreda, R. (2014), Iot and cloud convergence: Opportunities and challenges, *in* '2014 IEEE World Forum on Internet of Things (WF-IoT)', IEEE, pp. 375–376.

Borgia, E. (2014), 'The internet of things vision: Key features, applications and open issues', *Computer Communications* **54**(Supplement C), 1 – 31.

Chejerla, B. K. & Madria, S. K. (2017), 'Qos guaranteeing robust scheduling in attack resilient cloud integrated cyber physical system', *Future Generation Computer Systems* **75**, 145 – 157.
URL: *http://www.sciencedirect.com/science/article/pii/S0167739X17302650*

Colman-Meixner, C., Develder, C., Tornatore, M. & Mukherjee, B. (2016), 'A survey on resiliency techniques in cloud computing infrastructures and applications', *IEEE Communications Surveys Tutorials* **18**(3), 2244–2281.

Dastjerdi, A. V. & Buyya, R. (2016), 'Fog computing: Helping the internet of things realize its potential', *Computer* **49**, 112–116.

Davis, L. (1991), 'Handbook of genetic algorithms'.

Desai, N. & Punnekkat, S. (2019), Safety of fog-based industrial automation systems, *in* 'Proceedings of the Workshop on Fog Computing and the IoT', pp. 6–10.

Dorigo, M., Birattari, M. et al. (2007), 'Swarm intelligence.', *Scholarpedia* **2**(9), 1462.

Dressler, F. & Akan, O. B. (2010), 'Bio-inspired networking: from theory to practice', *IEEE Communications Magazine* **48**(11), 176–183.

Eckhoff, D. & Wagner, I. (2017), 'Privacy in the smart city 2013; applications, technologies, challenges and solutions', *IEEE Communications Surveys Tutorials* **PP**(99), 1–1.

Eckmann, J., Kamphorst, S. O., Ruelle, D. et al. (1995), 'Recurrence plots of dynamical systems', *World Scientific Series on Nonlinear Science Series A* **16**, 441–446.

Eisele, S., Mardari, I., Dubey, A. & Karsai, G. (2017), Riaps: Resilient information architecture platform for decentralized smart systems, *in* '2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)', pp. 125–132.

Faniyi, F. & Bahsoon, R. (2015), 'A systematic review of service level management in the cloud', *ACM Computing Surveys (CSUR)* **48**(3), 1–27.

Faruque, M. A. A. & Vatanparvar, K. (2016), 'Energy management-as-a-service over fog computing platform', *IEEE Internet of Things Journal* **3**(2), 161–169.

Fenn, J. (2006), 'Managing citations and your bibliography with bibtex', *The PracTEX Journal,(4)* .

Ferrer, A. J., Marquès, J. M. & Jorba, J. (2019), 'Towards the decentralised cloud: Survey on approaches and challenges for mobile, ad hoc, and edge computing', *ACM Computing Surveys (CSUR)* **51**(6), 1–36.

Forrest, S., Perelson, A. S., Allen, L. & Cherukuri, R. (1994), Self-nonself discrimination in a computer, *in* 'Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on', pp. 202–212.

Frustaci, M., Pace, P., Alot, G. & Fortino, G. (2017), 'Evaluating critical security issues of the iot world: Present and future challenges', *IEEE Internet of Things Journal* **PP**(99), 1–1.

Gan, Y., Zhang, Y., Hu, K., Cheng, D., He, Y., Pancholi, M. & Delimitrou, C. (2019), Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices, *in* 'Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems', pp. 19–33.

Ganek, A. G. & Corbi, T. A. (2003), 'The dawning of the autonomic computing era', *IBM Systems Journal* **42**(1), 5–18.

Gao, H.-H., Yang, H.-H. & Wang, X.-Y. (2005), Ant colony optimization based network intrusion feature selection and detection, pp. 3871 – 3875 Vol. 6.

Gharaibeh, A., Salahuddin, M. A., Hussini, S. J., Khreishah, A., Khalil, I., Guizani, M. & Al-Fuqaha, A. (2017), 'Smart cities: A survey on data management, security, and enabling technologies', *IEEE Communications Surveys & Tutorials* **19**(4), 2456–2501.

Ghosh, R., Longo, F., Naik, V. K. & Trivedi, K. S. (2010), Quantifying resiliency of iaas cloud, *in* 'Proceedings of the 2010 29th IEEE Symposium on Reliable Distributed Systems', SRDS '10, IEEE Computer Society, Washington, DC, USA, pp. 343–347.

Goścień, R. & Walkowiak, K. (2017), 'Modeling and optimization of data center location and routing and spectrum allocation in survivable elastic optical networks', *Optical Switching and Networking* **23**, 129 – 143. Design and modeling of Resilient optical networks RNDM 2015.
URL: *http://www.sciencedirect.com/science/article/pii/S157342771630042X*

Guo, M. & Bhattacharya, P. (2014), Diverse virtual replicas for improving intrusion tolerance in cloud, *in* 'Proceedings of the 9th Annual Cyber and Information Security Research Conference', CISR '14, ACM, New York, NY, USA, pp. 41–44.
URL: *http://doi.acm.org/10.1145/2602087.2602116*

Gutierrez, J., Naeve, M., Callaway, E., Bourgeois, M., Mitter, V. & Heile, B. (2001), 'Ieee 802.15.4: a developing standard for low-power low-cost wireless personal area networks', *Network, IEEE* **15**(5), 12–19.

Ha, W. (2018), Cloud service selection with fuzzy c-means artificial immune network memory classifier, *in* '2018 14th International Conference on Computational Intelligence and Security (CIS)', pp. 264–268.

Haas, Z. J., Halpern, J. Y. & Li, L. (2006), 'Gossip-based ad hoc routing', *IEEE/ACM Transactions on networking* **14**(3), 479–491.

Hariri, S., Eltoweissy, M. & Al-Nashif, Y. (2011), Biorac: Biologically inspired resilient autonomic cloud, *in* 'Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research', CSIIRW '11, ACM, New York, NY, USA, pp. 80:1–80:1.
URL: *http://doi.acm.org/10.1145/2179298.2179389*

Hassija, V., Chamola, V., Saxena, V., Jain, D., Goyal, P. & Sikdar, B. (2019), 'A survey on iot security: application areas, security threats, and solution architectures', *IEEE Access* **7**, 82721–82743.

Heble, S., Kumar, A., Prasad, K. V. V. D., Samirana, S., Rajalakshmi, P. & Desai, U. B. (2018), A low power iot network for smart agriculture, *in* '2018 IEEE 4th World Forum on Internet of Things (WF-IoT)', pp. 609–614.

Hecht, T., Smith, P. & Scholler, M. (2014), Critical services in the cloud: Understanding security and resilience risks, *in* 'Reliable Networks Design and Modeling (RNDM), 2014 6th International Workshop on', pp. 131–137.

Hernandez, G. (1999), 'Time series, periodograms, and significance', *Journal of Geophysical Research: Space Physics* **104**(A5), 10355–10368.

Hintjens, P. (2013), *ZeroMQ: messaging for many applications*, " O'Reilly Media, Inc.".

Hsin, H., Chang, E., Lin, C. & Wu, A. (2014), 'Ant colony optimization-based fault-aware routing in mesh-based network-on-chip systems', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **33**(11), 1693–1705.

Hussein, A., Elhajj, I. H., Chehab, A. & Kayssi, A. (2017), Sdn vanets in 5g: An architecture for resilient security services, *in* '2017 Fourth International Conference on Software Defined Systems (SDS)', pp. 67–74.

Ismagilova, E., Hughes, L., Dwivedi, Y. K. & Raman, K. R. (2019), 'Smart cities: Advances in research—an information systems perspective', *International Journal of Information Management* **47**, 88 – 100.
URL: *http://www.sciencedirect.com/science/article/pii/S0268401218312738*

Jabbar, A. (2010), A Framework to Quantify Network Resilience and Survivability, PhD thesis, Lawrence, KS, USA. AAI3417968.

Jaiswal, V., Sen, A. & Verma, A. (2014), 'Integrated resiliency planning in storage clouds', *Network and Service Management, IEEE Transactions on* **11**(1), 3–14.

Jhawar, R. & Piuri, V. (2013), 'Fault tolerance and resilience in cloud computing environments', *Computer and Information Security Handbook,* pp. 125–141.

Ju, X., Soares, L., Shin, K. G., Ryu, K. D. & Da Silva, D. (2013), On fault resilience of openstack, *in* 'Proceedings of the 4th Annual Symposium on Cloud Computing', SOCC '13, ACM, New York, NY, USA, pp. 2:1–2:16.
URL: *http://doi.acm.org/10.1145/2523616.2523622*

Kahla, M., Azab, M. & Mansour, A. (2018), Secure, resilient, and self-configuring fog architecture for untrustworthy iot environments, *in* '2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)', pp. 49–54.

Kanter, M. & Taylor, S. (2013), Diversity in cloud systems through runtime and compile-time relocation, *in* 'Technologies for Homeland Security (HST), 2013 IEEE International Conference on', pp. 396–402.

Kephart, J. O. & Chess, D. M. (2003), 'The vision of autonomic computing', *Computer* **36**(1), 41–50.

Khabbaz, M. J., Assi, C. M. & Fawaz, W. F. (2012), 'Disruption-tolerant networking: A comprehensive survey on recent developments and persisting challenges', *IEEE Communications Surveys Tutorials* **14**(2), 607–640.

Khalifa, A., Azab, M. & Eltoweissy, M. (2014), Resilient hybrid mobile ad-hoc cloud over collaborating heterogeneous nodes, *in* 'Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on', pp. 134–143.

Knoblich, J. A. (2010), 'Asymmetric cell division: recent developments and their implications for tumour biology', *Nature reviews Molecular cell biology* **11**(12), 849.

Kortuem, G., Kawsar, F., Fitton, D. & Sundramoorthy, V. (2010), 'Smart objects as building blocks for the Internet of things', *Internet Computing, IEEE* **14**, 44–51.

Koza, J. R. et al. (1994), *Genetic programming*, MIT press Cambridge.

Kulig, A., Drożdż, S., Kwapień, J. & Oświcimka, P. (2015), 'Modeling the average shortest-path length in growth of word-adjacency networks', *Physical Review E* **91**(3), 032810.

Kulwa, F., Li, C., Zhao, X., Cai, B., Xu, N., Qi, S., Chen, S. & Teng, Y. (2019), 'A state-of-the-art survey for microorganism image segmentation methods and future potential', *IEEE Access* **7**, 100243–100269.

Laprie, J.-C. (2005), Resilience for the scalability of dependability, *in* 'Network Computing and Applications, Fourth IEEE International Symposium on', pp. 5–6.

Le, M., Song, Z., Kwon, Y. & Tilevich, E. (2017), Reliable and efficient mobile edge computing in highly dynamic and volatile environments, *in* '2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)', pp. 113–120.

Lombardi, F., Di Pietro, R. & Soriente, C. (2010), Crew: Cloud resilience for windows guests through monitored virtualization, *in* 'Reliable Distributed Systems, 2010 29th IEEE Symposium on', pp. 338–342.

Luo, B. & Liu, W. (2011), The sustainability and survivabiltiy network design for next generation cloud networking, *in* 'Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on', pp. 555–560.

Lyu, L., Nandakumar, K., Rubinstein, B., Jin, J., Bedo, J. & Palaniswami, M. (2018), 'Ppfa: Privacy preserving fog-enabled aggregation in smart grid', *IEEE Transactions on Industrial Informatics* **14**(8), 3733–3744.

Mach, P. & Becvar, Z. (2017), 'Mobile edge computing: A survey on architecture and computation offloading', *IEEE Communications Surveys Tutorials* **19**(3), 1628–1656.

Majno, G. & Joris, I. (1995), 'Apoptosis, oncosis, and necrosis. an overview of cell death.', *The American journal of pathology* **146**(1), 3.

Makhdoom, I., Abolhasan, M., Lipman, J., Liu, R. P. & Ni, W. (2018), 'Anatomy of threats to the internet of things', *IEEE Communications Surveys Tutorials* pp. 1–1.

Mange, D., Sanchez, E., Stauffer, A., Tempesti, G., Marchal, P. & Piguet, C. (1998), 'Embryonics: A new methodology for designing field-programmable gate arrays with self-repair and self-replicating properties', *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **6**(3), 387–399.

Mao, Y., You, C., Zhang, J., Huang, K. & Letaief, K. B. (2017), 'A survey on mobile edge computing: The communication perspective', *IEEE Communications Surveys Tutorials* **19**(4), 2322–2358.

Marinai, S., Gori, M. & Soda, G. (2005), 'Artificial neural networks for document analysis and recognition', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **27**(1), 23–35.

Marsh, D., Tynan, R., O'Kane, D. & O'Hare, G. M. P. (2004), 'Autonomic wireless sensor networks', *Engineering Applications of Artificial Intelligence* **17**(7), 741 – 748. Autonomic Computing Systems. **URL:** *http://www.sciencedirect.com/science/article/pii/S0952197604001101*

Martí, L. & Schoenauer, M. (2018), Bio-inspired approaches to anomaly and intrusion detection, *in* 'Proceedings of the Genetic and Evolutionary Computation Conference Companion', pp. 1121–1137.

Maruyama, H. (2013), Towards systems resilience, *in* '2013 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)', pp. 1–4.

Marwan, N. (2008), 'A historical review of recurrence plots', *The European Physical Journal Special Topics* **164**(1), 3–12.

Marwan, N., Romano, M. C., Thiel, M. & Kurths, J. (2007), 'Recurrence plots for the analysis of complex systems', *Physics reports* **438**(5-6), 237–329.

McCulloch, W. S. & Pitts, W. (1943), 'A logical calculus of the ideas immanent in nervous activity', *The bulletin of mathematical biophysics* **5**(4), 115–133.

Mell, P. M. & Grance, T. (2011), Sp 800-145. the nist definition of cloud computing, Technical report, Gaithersburg, MD, United States.

Metallidou, C. K., Psannis, K. E. & Egyptiadou, E. A. (2020), 'Energy efficiency in smart buildings: Iot approaches', *IEEE Access* **8**, 63679–63699.

Miorandi, D., Lowe, D. & Yamamoto, L. (2010), Embryonic models for self–healing distributed services, *in* E. Altman, I. Carrera, R. El-Azouzi, E. Hart & Y. Hayel, eds, 'Bioinspired Models of Network, Information, and Computing Systems', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 152–166.

Mitalipov, S. & Wolf, D. (2009), Totipotency, pluripotency and nuclear reprogramming, *in* 'Engineering of stem cells', Springer, pp. 185–199.

Modares, H., Salleh, R. & Moravejosharieh, A. (2011), Overview of security issues in wireless sensor networks, *in* '2011 Third International Conference on Computational Intelligence, Modelling Simulation', pp. 308–311.

Modarresi, A., Gangadhar, S. & Sterbenz, J. P. G. (2017), A framework for improving network resilience using sdn and fog nodes, *in* '2017 9th International Workshop on Resilient Networks Design and Modeling (RNDM)', pp. 1–7.

Modarresi, A. & Sterbenz, J. P. G. (2017), Toward resilient networks with fog computing, *in* '2017 9th International Workshop on Resilient Networks Design and Modeling (RNDM)', pp. 1–7.

Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R. H., Morrow, M. J. & Polakos, P. A. (2018), 'A comprehensive survey on fog computing: State-of-the-art and research challenges', *IEEE Communications Surveys Tutorials* **20**(1), 416–464.

Neto, A. J. V., Zhao, Z., Rodrigues, J. J. P. C., Camboim, H. B. & Braun, T. (2018), 'Fog-based crime-assistance in smart iot transportation system', *IEEE Access* **6**, 11101–11111.

Ozeer, U., Etchevers, X., Letondeur, L., Ottogalli, F.-G., Salaün, G. & Vincent, J.-M. (2018), Resilience of stateful iot applications in a dynamic fog environment, *in* 'Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services', MobiQuitous '18, ACM, New York, NY, USA, pp. 332–341.
URL: *http://doi.acm.org/10.1145/3286978.3287007*

Oztemel, E. (2019), 'Intelligent manufacturing systems, smart factories and industry 4.0: A general overview', *Digital Manufacturing and Assembly Systems in Industry 4.0* p. 1.

Pan, J. & McElhannon, J. (2018), 'Future edge cloud and edge computing for internet of things applications', *IEEE Internet of Things Journal* **5**(1), 439–449.

Panwar, R. & Supriya, M. (2019), Autonomic resource allocation frameworks for service-based cloud applications: A survey, *in* '2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)', IEEE, pp. 214–219.

Paradis, L. & Han, Q. (2007), 'A survey of fault management in wireless sensor networks', *Journal of Network and systems management* **15**(2), 171–190.

Pereira, J., Ricardo, L., Luís, M., Senna, C. & Sargento, S. (2019), 'Assessing the reliability of fog computing for smart mobility applications in vanets', *Future Generation Computer Systems* **94**, 317 – 332.
URL: *http://www.sciencedirect.com/science/article/pii/S0167739X18307076*

Preden, J. S., Tammemäe, K., Jantsch, A., Leier, M., Riid, A. & Calis, E. (2015), 'The benefits of self-awareness and attention in fog and mist computing', *Computer* **48**(7), 37–45.

Raza, S., Wallgren, L. & Voigt, T. (2013), 'Svelte: Real-time intrusion detection in the Internet of Things', *Ad Hoc Networks* **11**, 2661–2674.

Rios, R., Roman, R., Onieva, J. A. & Lopez, J. (2017), From smog to fog: A security perspective, *in* '2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)', pp. 56–61.

Roman, R., Lopez, J. & Mambo, M. (2018), 'Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges', *Future Generation Computer Systems* **78**, 680 – 698.
URL: *http://www.sciencedirect.com/science/article/pii/S0167739X16305635*

Rudel, D. & Sommer, R. J. (2003), 'The evolution of developmental mechanisms', *Developmental Biology* **264**(1), 15 – 37.
URL: *http://www.sciencedirect.com/science/article/pii/S0012160603003531*

Rutten, E., Marchand, N. & Simon, D. (2017), Feedback control as mape-k loop in autonomic computing, *in* 'Software Engineering for Self-Adaptive Systems III. Assurances', Springer, pp. 349–373.

Saramäki, J., Kivelä, M., Onnela, J.-P., Kaski, K. & Kertész, J. (2007), 'Generalizations of the clustering coefficient to weighted complex networks', *Phys. Rev. E* **75**, 027105.
URL: *https://link.aps.org/doi/10.1103/PhysRevE.75.027105*

Schöler, H. R. (2016), The potential of stem cells: An inventory, *in* 'Humanbiotechnology as social challenge', Routledge, pp. 45–72.

Scholler, M., Bless, R., Pallas, F., Horneber, J. & Smith, P. (2013), An architectural model for deploying critical infrastructure services in the cloud, *in* 'Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on', Vol. 1, pp. 458–466.

Secci, S. & Murugesan, S. (2014), 'Cloud networks: Enhancing performance and resiliency', *Computer* **47**(10), 82–85.

Seo, J., Kim, K., Park, M., Park, M. & Lee, K. (2017), An analysis of economic impact on iot under gdpr, *in* '2017 International Conference on Information and Communication Technology Convergence (ICTC)', IEEE, pp. 879–881.

Singh, A., Singh, U. K. & Kumar, D. (2018), Iot in mining for sensing, monitoring and prediction of underground mines roof support, *in* '2018 4th International Conference on Recent Advances in Information Technology (RAIT)', pp. 1–5.

Sipper, M., Sanchez, E., Mange, D., Tomassini, M., Perez-Uribe, A. & Stauffer, A. (1997), 'A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems', *IEEE Transactions on Evolutionary Computation* **1**(1), 83–97.

Sousa, B., Pentikousis, K. & Curado, M. (2014*a*), 'Methodical: Towards the next generation of multihomed applications', *Computer Networks* **65**, 21 – 40.

Sousa, B., Pentikousis, K. & Curado, M. (2014*b*), Optimizing quality of resilience in the cloud, *in* 'Global Communications Conference (GLOBECOM), 2014 IEEE', pp. 1133–1138.

Souza Couto, R., Secci, S., Mitre Campista, M. & Kosmalski Costa, L. (2014), 'Network design requirements for disaster resilience in iaas clouds', *Communications Magazine, IEEE* **52**(10), 52–58.

Sterbenz, J. P. G., Hutchison, D., Çetinkaya, E. K., Jabbar, A., Rohrer, J. P., Schöller, M. & Smith, P. (2010), 'Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines', *Comput. Netw.* **54**(8), 1245–1265.

Stevens, W. R., Fenner, B. & Rudoff, A. M. (2004), *UNIX network programming*, Vol. 1, Addison-Wesley Professional.

Sun, L., He, J., Wang, C., Dong, H., Ma, J. & Zhang, Y. (2019), 'Survey of cloud sla assurance in pre-interaction and post-interaction start time phases', *Journal of Computers* **30**(1), 23–30.

Sun, Q., Li, H., Ma, Z., Wang, C., Campillo, J., Zhang, Q., Wallin, F. & Guo, J. (2015), 'A comprehensive review of smart energy meters in intelligent energy networks', *IEEE Internet of Things Journal* **3**(4), 464–479.

Tambouratzis, G. (2009), 'Using an ant colony metaheuristic to optimize automatic word segmentation for ancient greek', *IEEE Transactions on Evolutionary Computation* **13**(4), 742–753.

Thedchanamoorthy, G., Piraveenan, M., Kasthuriratna, D. & Senanayake, U. (2014), 'Node assortativity in complex networks: An alternative approach', *Procedia Computer Science* **29**, 2449 – 2461. 2014 International Conference on Computational Science.
URL: *http://www.sciencedirect.com/science/article/pii/S1877050914004062*

Thönes, J. (2015), 'Microservices', *IEEE software* **32**(1), 116–116.

Tortonesi, M., Stefanelli, C., Benvegnu, E., Ford, K., Suri, N. & Linderman, M. (2012), 'Multiple-uav coordination and communications in tactical edge networks', *IEEE Communications Magazine* **50**(10), 48–55.

Tran, P. N. & Boukhatem, N. (2008), The distance to the ideal alternative (dia) algorithm for interface selection in heterogeneous wireless networks, *in* 'Proceedings of the 6th ACM International Symposium on Mobility Management and Wireless Access', MobiWac '08, ACM, New York, NY, USA, pp. 61–68.

Trappe, W., Howard, R. & Moore, R. S. (2015), 'Low-energy security: Limits and opportunities in the internet of things', *IEEE Security Privacy* **13**(1), 14–21.

Tu, M. & Xu, D. (2013), System resilience modeling and enhancement for the cloud, *in* 'Computing, Networking and Communications (ICNC), 2013 International Conference on', pp. 1021–1025.

Vandebroek, S. V. (2016), 1.2 three pillars enabling the internet of everything: Smart everyday objects, information-centric networks, and automated real-time insights, *in* '2016 IEEE International Solid-State Circuits Conference (ISSCC)', pp. 14–20.

Vasconcelos, D., Severino, V., Neuman, J., Andrade, R. & Maia, M. (2018), Bio-inspired model for data distribution in fog and mist computing, *in* '2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)', Vol. 02, pp. 777–782.

Vessey, I. & Skinner, G. (1990), Implementing berkeley sockets in system v release 4, *in* 'Proceedings of the Winter 1990 USENIX Conference, Washington, DC', pp. 177–193.

Viejo, A. & Sánchez, D. (2019), 'Secure and privacy-preserving orchestration and delivery of fog-enabled iot services', *Ad Hoc Networks* **82**, 113 – 125.
URL: *http://www.sciencedirect.com/science/article/pii/S1570870518305493*

Von Neumann, J., Burks, A. W. et al. (1966), 'Theory of self-reproducing automata', *IEEE Transactions on Neural Networks* **5**(1), 3–14.

Vu, K., Hartley, K. & Kankanhalli, A. (2020), 'Predictors of cloud computing adoption: A cross-country study', *Telematics and Informatics* **52**, 101426.
URL: *http://www.sciencedirect.com/science/article/pii/S073658532030085X*

Wang, S., Zhang, X., Zhang, Y., Wang, L., Yang, J. & Wang, W. (2017), 'A survey on mobile edge networks: Convergence of computing, caching and communications', *IEEE Access* **5**, 6757–6779.

Wang, T., Ye, B., Li, Y. & Yang, Y. (2010), Family gene based cloud trust model, *in* '2010 International Conference on Educational and Network Technology', IEEE, pp. 540–544.

Welsh, T. & Benkhelifa, E. (2019), 'Bio-inspired multi-agent embryonic architecture for resilient edge networks', *IEEE Transactions on Industrial Informatics* pp. 1–1.

Wolpert, L. (2008), *The Triumph of the Embryo*, Dover books on biology, psychology and medicine, Dover Publications.
URL: *https://books.google.co.uk/books?id=VfdFOKz3O5UC*

Xu, H., Yu, W., Griffith, D. & Golmie, N. (2018), 'A survey on industrial internet of things: A cyber-physical systems perspective', *IEEE Access* **6**, 78238–78259.

Zanella, A., Bui, N., Castellani, A., Vangelista, L. & Zorzi, M. (2014), 'Internet of things for smart cities', *IEEE Internet of Things Journal* **1**(1), 22–32.

Zheng, C. & Sicker, D. C. (2013), 'A survey on biologically inspired algorithms for computer networking', *IEEE Communications Surveys & Tutorials* **15**(3), 1160–1191.

Zhou, Y., Fang, Y. & Zhang, Y. (2008), 'Securing wireless sensor networks: a survey', *IEEE Communications Surveys & Tutorials* **10**(3), 6–28.

Appendix A

Appendix A - Data

This appendix lists datasets as referenced in the main thesis.

Tables A.1 to A.4 present the averages performance values for 0 failure rate tests grouped by quantity of output messages descending.

Tables A.5 to A.8 present the average performance values for tests with 0.01 failure rate grouped by quantity of output messages descending.

div	func	sub	proc	recv	out	short	long	total
6	2	6	27	601	58	0.24	6.02	7.61
6	2	7	20	510	55	0.23	2.74	4.21
6	3	7	29	568	54	0.46	4.39	5.09
6	4	6	53	647	51	0.87	5.91	7.24
6	3	6	42	591	42	0.58	5.05	6.17
6	2	5	16	333	40	0.25	3.22	5.28
6	3	5	34	370	40	0.47	4.21	4.96
6	4	7	32	495	38	0.68	5.08	5.35
6	5	6	51	534	32	1.25	4.59	5.56
5	2	7	8	289	31	0.14	1.94	3.82
6	2	4	13	185	25	0.21	4.45	6.94
6	5	7	52	528	22	1.18	4.62	4.71
5	3	7	12	175	18	0.28	1.50	2.74
6	2	3	18	179	18	0.25	4.73	5.47
4	2	3	6	94	17	0.15	2.05	2.86
5	4	7	23	315	17	0.49	2.72	3.96
6	5	8	36	475	17	1.07	3.86	4.21
6	4	5	42	270	15	1.02	2.75	4.28
4	3	7	11	146	14	0.22	1.13	1.47
5	2	6	4	100	14	0.06	0.72	0.95
4	3	5	12	126	13	0.26	0.99	1.06
5	3	3	20	120	13	0.46	2.26	3.77
5	3	6	10	145	13	0.21	1.12	1.23
5	2	3	5	69	12	0.12	1.31	2.75
5	3	5	17	162	12	0.32	1.49	2.19
6	3	3	20	112	12	0.34	2.00	4.36
5	2	4	4	76	11	0.13	1.23	2.15
4	2	4	5	70	10	0.15	1.27	1.96
4	2	6	3	73	10	0.10	0.58	0.94
6	3	4	23	150	10	0.44	2.20	4.77

Table A.1: Average performance values for 0 failure rate tests part 1

div	func	sub	proc	recv	out	short	long	total
3	2	6	4	79	9	0.02	0.44	0.73
4	2	5	3	66	9	0.10	0.60	0.83
3	2	7	3	63	8	0.03	0.45	0.61
4	3	3	11	71	8	0.30	1.36	2.88
4	3	6	5	69	8	0.22	0.78	1.03
5	2	5	5	82	8	0.15	1.46	1.71
6	6	7	38	318	8	1.62	3.29	3.37
6	7	7	36	328	7	1.64	3.29	3.38
3	3	5	6	64	6	0.20	0.51	0.59
3	3	6	8	87	6	0.20	0.54	0.80
4	4	6	15	128	6	0.48	1.41	1.43
4	4	7	8	94	6	0.49	0.82	0.88
6	6	6	38	269	6	1.69	3.50	4.13
3	2	5	1	33	5	0.04	0.19	0.35
3	3	7	4	60	5	0.15	0.41	0.53
4	2	7	1	41	5	0.08	0.24	0.48
4	3	4	9	67	5	0.25	1.15	1.67
4	4	4	15	87	5	0.46	1.33	1.47
3	3	4	5	39	4	0.15	0.29	0.48
6	4	4	26	130	4	0.68	1.81	3.11
6	5	5	33	192	4	1.31	2.18	3.33
3	2	4	2	29	3	0.04	0.15	0.34
3	4	6	9	75	3	0.45	0.61	0.74
4	4	5	8	61	3	0.44	0.72	0.74
4	5	7	7	73	3	0.56	0.68	0.83
5	4	3	11	63	3	0.96	1.06	1.67
5	4	6	10	63	3	0.26	0.48	0.54
5	5	7	15	121	3	0.70	1.18	1.96
3	2	3	2	10	2	0.01	0.14	0.13
3	4	4	4	28	2	0.28	0.30	0.29

Table A.2: Average performance values for 0 failure rate tests part 2

div	func	sub	proc	recv	out	short	long	total
3	5	7	6	47	2	0.51	0.63	0.49
4	4	3	12	60	2	0.67	0.93	1.92
4	5	5	10	66	2	0.56	0.80	0.83
5	3	4	8	45	2	0.43	0.53	1.13
5	4	4	13	70	2	1.61	1.14	1.80
5	5	5	11	75	2	1.78	1.30	1.61
5	5	6	14	87	2	0.55	0.80	0.90
3	4	7	4	34	1	0.37	0.50	0.41
3	5	4	7	34	1	0.39	0.35	1.27
4	5	6	10	66	1	0.71	0.69	0.76
4	6	7	10	68	1	0.69	0.61	0.68
5	4	5	8	47	1	0.54	0.62	0.73
5	6	7	17	143	1	1.11	1.52	2.57
6	4	3	18	85	1	1.51	1.71	2.78
6	6	5	26	157	1	1.18	1.65	3.80
6	7	6	26	194	1	1.58	2.41	2.77
3	3	3	3	10	0	1.40	0.76	0.54
3	4	3	1	4	0	-1.00	0.05	0.05
3	4	5	5	23	0	0.21	0.32	0.26
3	5	3	1	3	0	-1.00	-1.00	0.02
3	5	5	8	44	0	0.46	0.41	0.54
3	5	6	4	27	0	0.55	0.36	0.25
3	6	3	1	5	0	-1.00	0.11	0.09
3	6	4	4	20	0	-1.00	0.22	0.31
3	6	5	5	31	0	0.59	0.63	0.38
3	6	6	3	21	0	0.52	0.42	0.23
3	6	7	1	5	0	-1.00	0.01	0.05
3	7	3	2	5	0	-1.00	0.12	0.09
3	7	4	1	7	0	-1.00	0.09	0.10
3	7	5	4	21	0	-1.00	2.95	1.05

Table A.3: Average performance values for 0 failure rate tests part 3

div	func	sub	proc	recv	out	short	long	total
3	7	6	2	13	0	-1.00	0.24	0.22
3	7	7	1	10	0	-1.00	0.14	0.13
4	5	3	8	41	0	0.31	0.48	1.66
4	5	4	9	48	0	0.46	0.63	0.84
4	6	3	7	45	0	-1.00	0.83	3.34
4	6	4	8	39	0	-1.00	0.42	0.58
4	6	5	7	49	0	0.59	0.58	0.59
4	6	6	7	45	0	0.37	0.32	0.42
4	7	3	6	37	0	-1.00	0.35	1.52
4	7	4	1	18	0	-1.00	0.36	0.28
4	7	5	4	24	0	-1.00	0.61	0.61
4	7	6	3	25	0	-1.00	0.27	0.71
4	7	7	1	13	0	-1.00	0.10	0.20
5	5	3	10	44	0	-1.00	0.80	2.37
5	5	4	7	36	0	-1.00	0.37	0.81
5	6	3	5	30	0	-1.00	0.33	1.38
5	6	4	4	27	0	-1.00	0.57	0.72
5	6	5	3	21	0	10.20	0.34	0.40
5	6	6	1	13	0	-1.00	0.13	0.14
5	7	3	7	40	0	-1.00	0.43	2.43
5	7	4	5	40	0	-1.00	0.46	1.21
5	7	5	2	16	0	-1.00	0.14	0.24
5	7	6	3	24	0	-1.00	0.12	0.27
5	7	7	7	49	0	-1.00	0.46	1.07
6	5	3	13	65	0	0.64	0.77	2.53
6	5	4	14	79	0	0.79	1.07	2.43
6	6	3	12	67	0	0.54	0.80	3.32
6	6	4	11	73	0	-1.00	1.08	2.16
6	7	3	10	71	0	-1.00	1.58	2.24
6	7	4	10	69	0	-1.00	0.91	3.14
6	7	5	17	120	0	1.50	1.25	2.87

Table A.4: Average performance values for 0 failure rate tests part 4

div	func	sub	proc	recv	out	short	long	total
6	3	6	14	139	17	0.22	2.07	3.90
6	2	5	10	109	14	0.16	3.44	4.59
6	2	6	4	89	13	0.07	1.91	3.41
6	2	4	8	90	12	0.12	3.52	5.09
6	2	7	5	88	12	0.07	0.94	1.49
6	2	3	13	111	11	0.16	4.69	5.83
4	2	6	5	68	9	0.07	0.64	1.36
5	2	5	6	68	8	0.34	1.91	2.92
5	2	6	3	54	8	0.21	0.83	1.83
5	2	7	6	66	8	0.10	1.49	2.32
5	3	5	9	80	8	0.32	1.39	2.09
6	3	4	13	77	8	0.29	2.72	4.43
4	2	7	3	49	7	0.06	0.47	1.41
5	2	3	4	43	7	0.11	1.79	3.45
6	3	5	15	93	7	0.38	1.46	4.02
6	3	7	8	84	7	0.27	1.33	1.75
5	2	4	1	26	6	0.05	0.19	2.59
3	3	6	6	55	5	0.43	0.79	1.18
4	2	3	5	44	5	0.10	1.62	2.36
4	2	4	2	27	5	0.13	0.81	2.02
6	4	6	15	106	5	0.64	1.36	3.86
3	2	6	3	29	4	0.06	0.74	1.04
3	2	7	1	28	4	0.05	0.42	0.91
3	3	7	7	54	4	0.19	0.45	1.31
4	3	4	6	36	4	0.80	1.18	1.87
6	3	3	12	59	4	1.00	1.90	3.35
3	2	5	2	17	3	0.04	0.36	0.60
4	2	5	2	21	3	0.06	0.25	0.43
4	3	3	7	34	3	0.93	1.21	2.82
4	3	6	5	37	3	0.29	0.88	1.59

Table A.5: Average performance values for failure rate tests part 1

div	func	sub	proc	recv	out	short	long	total
5	3	3	7	38	3	0.95	1.23	3.17
5	3	7	5	39	3	0.17	0.32	1.76
6	4	7	11	89	3	0.53	1.07	1.83
3	2	3	1	11	2	0.05	0.14	0.63
3	2	4	1	10	2	0.06	0.27	0.72
3	4	6	6	35	2	0.38	0.37	1.23
4	3	7	4	32	2	0.16	0.51	0.76
5	3	4	6	30	2	0.20	0.30	2.24
5	4	5	9	51	2	0.71	0.67	1.73
5	4	6	7	50	2	0.77	0.79	1.66
3	3	3	3	11	1	0.25	0.17	0.47
3	3	4	2	10	1	0.14	0.14	0.62
3	4	7	5	26	1	0.62	0.40	0.63
4	3	5	4	20	1	0.57	0.41	0.60
4	4	6	7	40	1	0.33	0.70	1.40
5	3	6	4	22	1	0.18	0.19	0.98
5	4	3	8	37	1	1.48	1.34	2.68
5	4	7	13	67	1	0.53	0.69	1.71
5	5	5	5	34	1	1.24	0.78	2.03
5	5	7	10	66	1	0.85	0.90	1.82
6	4	4	12	58	1	0.70	1.45	2.72
6	4	5	12	59	1	0.30	0.57	3.31
6	5	5	15	86	1	3.46	1.97	3.06
6	5	6	15	90	1	1.36	1.30	2.39
3	3	5	4	14	0	0.15	0.64	0.83
3	4	3	1	5	0	0.21	0.12	0.13
3	4	4	4	13	0	0.25	0.14	0.80
3	4	5	2	6	0	0.21	0.23	0.36
3	5	3	2	6	0	0.89	0.25	0.42
3	5	4	2	10	0	0.35	0.11	0.14

Table A.6: Average performance values for failure rate tests part 2

div	func	sub	proc	recv	out	short	long	total
3	5	5	4	17	0	0.25	0.31	1.06
3	5	6	2	12	0	0.31	0.24	0.51
3	5	7	3	16	0	0.75	0.33	0.63
3	6	3	1	1	0	-1.00	-1.00	0.01
3	6	4	3	13	0	-1.00	0.59	0.67
3	6	5	3	16	0	-1.00	0.23	0.64
3	6	6	3	18	0	0.98	0.42	0.64
3	6	7	3	15	0	-1.00	0.46	1.04
3	7	3	1	1	0	-1.00	-1.00	0.01
3	7	4	3	14	0	-1.00	0.31	0.50
3	7	5	2	9	0	-1.00	0.18	0.38
3	7	6	2	9	0	0.54	0.07	0.45
3	7	7	3	13	0	-1.00	0.25	0.46
4	4	3	5	24	0	0.61	0.36	1.42
4	4	4	5	23	0	0.30	0.31	0.90
4	4	5	5	22	0	0.64	0.43	0.51
4	4	7	4	25	0	0.29	0.62	0.81
4	5	3	4	17	0	1.92	0.41	1.07
4	5	4	6	20	0	0.35	0.22	0.86
4	5	5	3	14	0	0.50	0.34	0.40
4	5	6	6	31	0	0.32	0.48	0.89
4	5	7	5	21	0	0.37	0.32	0.39
4	6	3	5	25	0	-1.00	0.35	1.41
4	6	4	3	18	0	-1.00	0.18	1.12
4	6	5	2	9	0	-1.00	0.22	0.30
4	6	6	2	15	0	0.41	0.31	0.55
4	6	7	2	8	0	-1.00	0.13	0.37
4	7	3	3	18	0	-1.00	0.50	1.69
4	7	4	1	13	0	-1.00	0.24	0.77
4	7	5	1	8	0	-1.00	0.18	0.27

Table A.7: Average performance values for failure rate tests part 3

div	func	sub	proc	recv	out	short	long	total
4	7	6	2	11	0	-1.00	0.25	0.31
4	7	7	5	29	0	0.80	0.37	0.39
5	4	4	8	30	0	0.21	0.25	2.05
5	5	3	7	31	0	-1.00	0.74	1.99
5	5	4	6	27	0	0.26	0.22	0.81
5	5	6	7	33	0	0.45	0.26	1.16
5	6	3	6	28	0	-1.00	0.35	1.92
5	6	4	6	28	0	-1.00	0.48	1.15
5	6	5	5	29	0	0.46	0.42	0.74
5	6	6	3	14	0	-1.00	0.14	0.36
5	6	7	5	25	0	0.86	0.35	0.72
5	7	3	2	17	0	-1.00	0.11	0.74
5	7	4	6	30	0	-1.00	0.26	1.67
5	7	5	3	18	0	-1.00	0.22	0.42
5	7	6	1	8	0	-1.00	0.08	0.08
5	7	7	4	22	0	-1.00	0.72	0.93
6	4	3	10	44	0	3.60	1.42	4.60
6	5	3	7	39	0	0.37	1.19	3.53
6	5	4	10	45	0	0.58	1.14	2.95
6	5	7	7	48	0	0.79	0.71	1.38
6	6	3	6	31	0	-1.00	0.96	2.92
6	6	4	5	33	0	-1.00	0.88	2.37
6	6	5	9	46	0	0.80	0.41	1.72
6	6	6	13	77	0	1.16	1.22	2.35
6	6	7	7	49	0	1.65	0.74	1.38
6	7	3	4	27	0	-1.00	0.31	2.55
6	7	4	4	27	0	-1.00	0.19	2.62
6	7	5	7	42	0	0.45	0.36	3.27
6	7	6	6	36	0	-1.00	0.31	1.78
6	7	7	6	44	0	3.83	1.12	1.54

Table A.8: Average performance values for failure rate tests part 4

Appendix B

Appendix B - Publications

This appendix presents all publications derived from this thesis at time of submission.

1. **On Resilience in Cloud Computing : A survey of techniques across the Cloud Domain** - This paper presents a comprehensive review of cloud computing resilience techniques across both traditional centralised and decentralised models. A subset of this work can be found in Chapter 2. At the time of writing the paper is "in press" for publication in the *ACM Computing Surveys* Journal. It supersedes an initially published survey paper: **Perspectives on Resilience in Cloud Computing: Review and Trends** - which was presented at the *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*.

2. **Embyronic Model for Highly Resilient PaaS** was presented at the *2018 Fifth International Conference on Software Defined Systems (SDS)*. It maps embryonic characteristics to cloud computing and resilience requirements in order to derive the initial embryonic cloud model that was developed for centralised cloud features. Some of this work is presented in chapter 3.

3. **Bio-Inspired Multi-agent Embryonic Architecture for Resilient Edge Networks** was published in the *IEEE Transactions on Industrial Informatics* Journal in May 2019. It presents the Cellular Automata based model of the embryonic architecture including simulation results for the majority of results in chapter 4. It presents the concept in the context of real-world use cases for resilient industrial IoT at the network edge.

4. **Embyronic-Inspired Resilient Service Delivery at the Hostile Mobile Edge** - Is currently under review in *Springer's Peer to Peer Networking and Applications* journal. It presents the performance evaluation results of the proof-of-concept embryonic architecture.

On Resilience in Cloud Computing: A Survey of Techniques across the Cloud Domain

THOMAS WELSH, University of Limerick, Republic of Ireland
ELHADJ BENKHELIFA, Staffordshire University, UK

Cloud infrastructures are highly favoured as a computing delivery model worldwide, creating a strong societal dependence. It is therefore vital to enhance their resilience, providing persistent service delivery under a variety of conditions. Cloud environments are highly complex and continuously evolving. Additionally, the plethora of use-cases ensures requirements for persistent service delivery vary. As a contribution to knowledge, this work surveys resilience techniques for cloud environments. We apply a novel perspective using a layered model of traditional and emerging cloud paradigms. Works are then classified according to the Resilinets model. For each layer, the most common techniques with limitations are derived including an actor's strength in influencing resilience in the cloud with each technique. We conclude with some future challenges to the field of resilient cloud computing.

CCS Concepts: • **General and reference → Surveys and overviews**; • **Computer systems organization → Dependable and fault-tolerant systems and networks**; **Cloud computing**; • **Security and privacy → Distributed systems security**; *Network security*;

Additional Key Words and Phrases: Resilience, cloud, fog, edge, survey

ACM Reference format:
Thomas Welsh and Elhadj Benkhelifa. 2020. On Resilience in Cloud Computing: A Survey of Techniques across the Cloud Domain. *ACM Comput. Surv.* 53, 3, Article 59 (May 2020), 36 pages.
https://doi.org/10.1145/3388922

1 SCOPE: CLASSIFYING RESILIENCE FOR THE CLOUD

Resilience, in the context of computer systems and networks, is defined in many ways. Some consider it synonymous with fault-tolerance [84]. Laprie provides two descriptions: "the persistence of dependability when facing changes" and "the persistence of service delivery that can justifiably be trusted, when facing changes" [62]. Sterbenz et al. provide a similar definition: "the ability of the network to provide and maintain an acceptable level of service in the face of various faults and challenges to normal operation" [112]. Queiroz et al. suggest: "Resilience is the capacity of critical services to adapt in order to provide their functionalities in cases of undesired events compromising parts of the system." [95]. Abdullah et al. [1] consider a business/organisation perspective : "Resilience refers to the capacity of human beings/system/organization to survive and thrive in the face of adversity...it is a property that is closely associated with the capacity to avoid, contain

This work was supported, in part, by Science Foundation Ireland grants 16/RC/3918 and 13/RC/2094.
Authors' addresses: T. Welsh, Tierney Building, University of Limerick, Sreelane, Limerick, V94 NYD3, Republic of Ireland; E. Benkhelifa, Mellor Building, Staffordshire University, College Road Stoke-on-Trent, ST4 2DE, United Kingdom.

0360-0300/2020/05-ART59 $15.00
https://doi.org/10.1145/3388922

and mitigate accidents." Or simply: "the percentage of lost traffic upon failures" [67]. These few definitions indicate the numerous factors that can be considered during the development and deployment of a resilient system. This variety is related to the variety of fields in which resilience is applied. As each will have different characteristics, numerous measurement methodologies will appear. Additionally, specific use-cases may omit certain characteristics due to their lesser relevance.

In this survey, we adhere to the comprehensive definitions of the Resilinets model by Sterbenz et al. They cover a variety of measurable or desirable characteristics of resilience, grouped into *trustworthiness* and *challenge tolerance*, viewed as internal and external factors, respectively. These may be adapted as appropriate to take into account novel features of the cloud. We direct the reader to the resilience discipline definitions categorised within [112].

1.1 Defining the Cloud

Cloud computing is a service-driven computing model whereby an end-user will provision and use computing resources from a Cloud Service Provider (CSP) in line with an agreed upon Service Level Agreement (SLA). The service hosted by the CSP could take many forms, consisting of networking, storage, or computational components [76]. Similar to traditional computing environments, cloud environments are multi-layered. The composition differs depending upon the CSP infrastructure, the application's use-case, or the particular model used for analysis.

A typical cloud datacentre would consist of the underlying physical infrastructure: servers, storage arrays, and networking hardware. Virtualised Infrastructure (VI), a pool of resources: Virtual Machines (VMs) and/or containers running atop of Virtual Machine Monitors (VMM)s with Virtual Storage (VS) devices and Virtual Networks (VNs). These resources are situated upon the Physical Infrastructure (PI) hardware, connected by Physical Networking (PN). A management layer coordinates physical Resource Management (RM) and the service life cycle. Performance is managed through distributing services using Load Balancing (LB). Services are created and managed using Service Orchestration (SO) and executed using Service Scheduling (SCH). Further service-oriented capabilities such as security are also provided.

The datacentre (DC) architecture is relevant when examining resilience within cloud infrastructure, as it is the foundation upon which the cloud service will sit. However, the resilience of the DC is not always relevant to the resilience of a service being hosted. For example, a cloud service may straddle multiple forms of infrastructure and, second, the user/CSP may have no ability to affect the resilience at this layer, dependent upon the cloud service delivery model employed.

In the NIST definition for cloud computing [76] the prominent service delivery models are defined as a layered architecture: Software-as-a-Servce (SaaS), Platform-as-a-Service(PaaS) and Infrastructure-as-a-Service (IaaS). Responsibilities (for the management/configuration/security, etc.) of the service being delivered vary between CSP and the user. This division of responsibility is an important concept within the context of resilience, as the level of control given may determine the user's abilities to affect its resilience. NIST defines a further three actors: auditor, broker, and carrier.

Due to emerging disciplines and delivery models, matters are complicated further. In addition to those layers discussed above, there are layers within the decentralised cloud. Once considered to be an emerging discipline, cloud computing is now arguably emerged, although is constantly evolving. In tandem with new technologies and use-cases, new forms of cloud computing are developed to accommodate emerging disciplines such as the Internet of Things (IoT) and big data. These involve distributing the cloud services across devices or network architectures dissimilar to the typical DC-only model.

Bilal et al. explain: "different emerging technologies situated at the edge of the network to provide computational and storage resources to deliver real-time communication with minimum

latency" [20]. While Baktir et al. explain that despite differences, these disciplines all largely attempt to accomplish the same goal and are variations of edge disciplines. What varies is their use-case and presumably the underlying technologies in which the new processing occurs [11].

A summary of these emerging disciplines are below:

- **Fog Computing** - seen first as an extension to the cloud but now as complementary or independent from it. It involves a hierarchy of services where some processing/storage is executed closer to the edge of the network while analytics can occur in the cloud. This can occur in small-scale clouds but also on a variety of different hardware such as base stations, routing hardware, and so on [20, 82, 90, 98].
- **Mobile Cloud Computing (MCC)** - the concept of resource augmentation from a mobile to a remote device to maximise resource efficiency and power consumption. Originally intended for centralised cloud DCs, the potential for processing at the edge is now seeing interest [20, 98, 124].
- **Cloudlets** - involve the deployment of small clouds used to reduce short falls in mobile cloud computing [2, 20].
- **Mobile Edge Computing (MEC)** - provides cloud services at the edge of cellular networks such as 5G nodes, this increases performance through latency reduction, traffic optimisation, and enhanced services, e.g., location-driven [20, 72, 73, 90, 98, 124].
- **Mist Computing** - pushes data processing services as far as possible to the sensor and actuator devices [93, 119].

These definitions illustrate that decentralised disciplines involve distributing cloud services closer to the edge of the network, where the end-user device, sensor, or actuator will be. Within the context of this work, to manage the complexity associated with non-standardised and evolving definitions, these cloud disciplines are grouped into three layers. This creates a new hierarchy of centralised and decentralised cloud architectures, where services may be positioned in one or more layers. The topmost layer is the centralised cloud infrastructure within a data centre. The middle layer is the fog, where cloud services and data processing can occur during transit to the cloud or in a constrained manner upon devices closer to the application edge. The final layer, mist, is where the sensors, actuators, and user devices sit and where minimal processing may occur. This model represents the hierarchical layered cloud family of disciplines; components of these disciplines (i.e., the physical devices, protocols, and actors) sit within these layers.

The following points are made considering resilience in emerging cloud disciplines:

- The cloud infrastructure's distinct architecture is relevant to understanding its own resilience but not always responsible for guaranteeing service resilience. Therefore, the relationship between resilience techniques operating in lower levels and a service on a higher level should be established.
- Emerging disciplines cause services to be delivered on decentralised cloud infrastructure far away from the DC, sometimes independently from it.
- The chosen service delivery model will affect the ability of the user or CSP to adjust the resilience of the service. Therefore, this is a key factor in resilience technique selection. All delivery models can be employed upon all architectures although with greater constraints closer to the edge.

We illustrate in Figure 1 the relationship between the centralised and decentralised cloud disciplines and their underlying architectural constituents. The diagram shows that cloud disciplines (coloured) may span one or more architectural layers, potentially encompassing a variety of different hardware configurations in addition to physical and logical architectures.

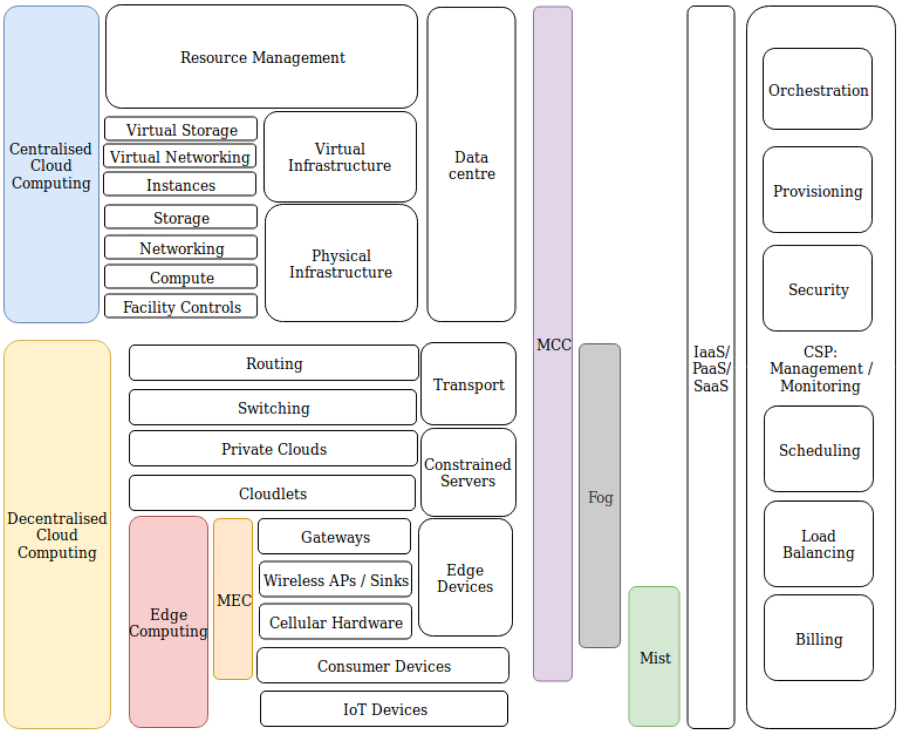


Fig. 1. Decentralised cloud computing model. Illustrating the relationship and overlap between different cloud models and architectural components. MEC=Mobile Edge Computing, MCC=Mobile Cloud Computing.

1.2 Related Work in Cloud Resilience

Due to the novelty of this area, surveys in resilience are limited. This work extends our previous survey in Reference [125], which lacks the detailed analysis according to techniques and disciplines presented here, in addition to the further analysis of the decentralised cloud. Moreover, this work complements our previous survey on intrusion detection for resilient IoT [17], which is out of the scope of this article.

A few surveys encompass some aspects of this work. Cheraghlou et al. provide a survey of fault-tolerant-specific architectures in the cloud [29], Milani et al. present a survey of data replication techniques in the cloud [78], and Mistrik et al. discuss fault-tolerant workflow management techniques [91]. Colman-Meixner et al. present the only survey on resilience techniques in cloud computing infrastructure [30]. Employing a layered model, they provide an in-depth study involving classification of resilience approaches used. Their primary findings highlight that for cloud systems, replication and checkpointing are the most common techniques for storage, virtualisation, and migration from the storage side and multi-layer protection for networking. However, a considerable number of techniques they evaluated are based on non-cloud environments that may be applicable to the cloud. That approach is avoided in this work due to its open-ended nature. Additionally, their model fails to account for emerging/decentralised cloud disciplines. Without this

consideration it becomes difficult to understand how integration and interoperability between emerging disciplines will affect the resilience at different layers. In contrast, this review will focus upon resilience techniques that enable a given CSP to deliver services resiliently upon their platform using any delivery model.

As a contribution to knowledge, we propose a model that encompasses the decentralised cloud and its relationship with the centralised cloud. We review only cloud-centric techniques. Definitions from the Resilinets model [112] are then applied to provide a rigid classification of techniques employed throughout literature. Combining these two models, each work can now be classified appropriately in terms of cloud layer, cloud components used, and resilience disciplines enabled. This is important to manage the scope and complexity of the survey. This survey classifies work in cloud service resilience according to a number of factors:

- Where in the hierarchy of centralised/decentralised cloud disciplines the work is situated.
- The architectural components according to the model in Figure 1 the work applies to.
- The cloud delivery model that the work applies to.
- The resilience disciplines that apply to the work, according to the Resilinets model [112].
- The techniques used to accomplish the resilience.

The work will therefore be considered within the following categories, which are representative of the centralised/decentralised cloud environment:

(1) **Physical and data-centre resilience** - techniques used to enhance the resilience of the datacentre. Placed here for completeness and to illustrate the underlying resilience.
(2) **Virtual Resource Abstraction** - the resilience of virtualised/abstracted resources such as VS, VI, VMMs, and so on.
(3) **Cloud Management** - techniques associated with cloud middleware/management, e.g., orchestration of services.
(4) **Decentralised Cloud** - resilience techniques for cloud environments that are closer to the network edge.
(5) **Alternative Architectures** - methods using unconventional architectures.

1.3 Paper Selection Criteria

Papers were selected for inclusion after searching popular databases, namely, ACM Library, IEEE Explore, Science Direct, and Google Scholar. A number of keyword permutations were used that involved "resilience" combined with different cloud computing models and architectures, e.g., "resilient PaaS," "Cloud resilience," and so on. During this search it quickly transpired that resilience definitions vary considerably from author to author. This is the motivation for leveraging the Resilinets model [112], as it provided a consistent definition and classification of these disciplines. This model is also frequently found referenced in other works and therefore provided support to its accuracy. These disciplines then replaced the resilience keyword with variations of cloud computing to procure more works for evaluation.

Scope management is an important issue due to the range of disciplines involved and the complex nature of the cloud. First, papers were omitted that did not focus solely upon cloud environments. While intuitive, the authors in Reference [30] include a number of non-cloud works that could still be applicable. Although relevant to resilience in general, it was decided this these papers could skew our analysis and could cause the review to be open-ended. The one exception is works that focus upon data-centre resilience where deemed relevant. In a similar manner a number of security-oriented papers were excluded, as techniques for firewalls, intrusion detection systems, and so on, can largely be applied anywhere, but the authors did not have a cloud resilience specific

Table 1. Data and Physical Layer Resilience

Method	Model	Components	Disciplines	Limitations
Back-up links [71]	Phy	PI	SRV	High cost
Optimal geo-distribution of datacentres [110]	Phy	DC	DT	High cost
Server placement for VM backups [32]	IaaS	PI; DC	DT	Physical layer only
Optimal routing using load balancing and graph analysis [81]	Phy	PI; DC	R,FT	Non-cloud
Topology and demographic/economic focused DC placement [44]	Phy	PI; DC	SRV	Only available to some CSP
Monte-Carlo tree search for resource provisioning [3]	Phy	PI; DC; RM	SRV	Only available to some CSP
Datacentre provision for resource optimisation [130]	Phy	PI; DC; RM	SRV	High cost and complexity
User aggregated link sharing [64]	Phy,	PI; DC; RM	SRV	Limited compartmentalisation
Simulating survivability scenarios [33]	Phy	PI; DC	SRV	Simulation only
Leverage SDN for survivability [27]	Phy	PI; DC; VN; RM	SRV	Centralised management

PI=Physical Infrastructure, DC=Data Centres, RM=Resource Management, VN=Virtual Networking, SRV= Survivability, FT=Fault Tolerance, R=Resilience, DT=Disruption Tolerance.

goal. Finally, works older than 10 years at the time of first selection were omitted. Overall, these works were selected to answer the following: What was the current state-of-the-art in cloud computing resilience, including what techniques were used, how effective are they, and what are their limitations? It was also conducted to identify any further gaps in the field to provide situation of the work.

2 STATE-OF-THE-ART IN CLOUD RESILIENCE

This section surveys literature according to the layers defined previously. Each work is analysed according to the techniques used and cloud components applied. A summary table is given for each section.

2.1 Physical and Data-centre Resilience

DCs are used as a strong argument that the security and resilience of cloud computing is greater than that of traditional paradigms, largely because cloud infrastructures are typically hosted in DCs with greater facilities than those possible to finance or manage by a single organisation. Highly redundant resources, excellent power resilience, excellent physical security, and strong network links to the internet backbone. Therefore, these underlying characteristics ensure these environments are resilient by nature. This section discusses a number of works in this layer that attempt to improve this further. Table 1 summarises this literature.

Mohamed discusses the area comprehensively, without the cloud context. He reiterates the inherent resilience of data centres and evaluates routing protocols, load balancing, and graph analysis techniques to provide enhanced resilience [81].

Lou et al. consider Cloud network survivability and sustainability within the context of energy-aware solutions: Energy Aware Backup Protection (EABP) [71]. They argue that as the requirement for energy-efficient services increases, survivability and resilience should not be ignored. They present a new model that allows multiple links to share one backup, drastically reducing capacity requirements with only a small increase in energy consumption. It is not certain whether the system will maintain resilience in the case that the number of links failed exceeds the capacity of the backup. Goścień and Walkowiak also consider survivability [44]. They undertake a

study to investigate the physical placement of DCs along optical fibre links from a topology and demographic-economical perspective. They illustrate that placement policies have a strong impact upon survivability. Applying a Monte-Carlo tree search for resource allocation has been shown to optimise this process [3].

Conversely, Couto et al. illustrate that survivability techniques including placement and redundancy can have grave negative effects upon the latency of applications in highly survivable situations (up to 80% degradation) [33], which is intuitive yet relevant. They also discuss the design for clouds that must be resilient in the face of disasters [110], in contrast to many surveyed works. Their motivation is that most SLAs do not cover disaster resilience, (e.g., hurricanes). A key component for their resiliency is geographical distribution and fail-over systems for activation during the event of a failure. They present a methodology for developing disaster-resilient networks and also a VM placement algorithm [32].

Zhang et al. propose resource orchestration as a technique to enable survivability in optical networks through minimising datacentre provisioning [130]. Additionally, they aggregate backup "k-node" links for multiple users to improve surivivability in a resource-optimal manner upon disaster [64].

Chandna et al. present a survivability solution in optical DC networks using Software Defined Networking (SDN) [27]. They illustrate the strength of these techniques and highlight that most future methods are likely to employ SDN. However, while geo-distribution improves resiliency it does not guarantee it. They highlight the necessity for VM placement algorithms as a high priority for guaranteeing resilience for cloud SLAs. This work highlights the need to understand the differing network layers when considering resilience in the cloud. Therefore, an important question is: How effectively can resilient virtual networks be designed without information about its lower layers?

2.2 Virtual Resource Abstraction

The works reviewed in the following sub-sections describe techniques that are used to provide resilience to a particular VR such as VS, instances, VNs, or the VMMs such as hypervisors.

2.2.1 Storage. The most simplistic method of ensuring resiliency of data is replication. When a failure happens, copies are accessed or migrated. This principle is the basis of many storage resiliency techniques such as the commonly deployed Redundant Array Inexpensive Disks (RAID). Storage, however, is expensive, so techniques to reduce the cost of redundancy are sought after. This is not just the cost of the hardware but also methods ensuring consistency. There is a current drive to move the majority of an organisation's storage to cloud infrastructures, due to their inherent storage resilience upon mass redundancy of resources and global remote access. Therefore, as the standard cloud could be considered a resilient storage mechanism, the techniques discussed will focus on more extreme scenarios or optimisation techniques. Table 2 summarises this literature.

Jaiswal et al. present Resilient Storage Cloud Map (RSCMap) [53]. Its goal is optimising the design of resilient cloud storage via disaster recovery planning. It permits an appropriate replication function to be selected according to the data type, needs, and cost available. This is applicable to a variety of use-cases. The efficacy of relying on disaster recovery for resilience is questionable due to the number of situations (e.g., data leakage, safety-critical systems) where resilience is required prior to a system fault.

Westmark provides another technique focusing upon resilience to disasters (disruption tolerance) [126]. They propose Rapid Data Evacuation (RDE) in which *a priori* knowledge of an imminent disaster permits a heuristic to determine the external network links with the least delay. This

Table 2. Storage Resilience Techniques

Method	Model	Components	Disciplines	
Cost-based disaster recovery planning, replication [53]	IaaS	VS	A, DT	Disaster-recovery not always beneficial to service delivery
Code-based recovery [24]	IaaS	VS	A, FT	Cost resulting from groups of nodes
Rapid Data Evacuation (RDE) prior to disaster event using a heuristic to select least delay paths [126]	IaaS/	VS	DT	*A priori* disaster knowledge, bandwidth requirements
Network overlay on diverse storage across multi DCs [43]	PaaS	VS, VN, RM, SO	FT	High replication cost
Replication covering correlated and non-correlated failures [68]	IaaS	VS	FT	High replication cost
Storage across diverse cloud-of-clouds [19]	IaaS	VS, RM	FT, SEC	High replication cost
Provide security to the client side of cloud storage services [75]	Client	CD	SEC	Client side only
A survey and hybrid technique between erasure coding and replication [83]	IaaS	VS, RM	FT	High replication cost
Efficient replication for unknown query rate [94]	IaaS	VS, RM	FT	High replication cost
Homomorphic token and erasure data [123]	IaaS	VS	FT, I	Data loss with loss of key
Component-based workflow RM from client to cloud [128]	PaaS	VS, RM	FT, SEC	Complex modelling process
Multi-DC backup disaster routing, shortest window for disaster RB [129]	IaaS	DC, VS, RM	DT	*A priori* disaster knowledge, high replication cost

VS=Virtual Storage, VN=Virtual Networking, RM=Resource Management, SO=Service Orchestration, CD=Client Devices, DC=Data Centres, A=Availability, DT=Disruption Tolerance, FT=Fault Tolerance, SEC=Security, I=Integrity.

enables safe and quick data migration from the cloud. Similarly, Yao et al. determine the shortest window necessary for backups to optimise the cost of replication in case of a disaster scenario [129].

An alternative to needing to evacuate data is to ensure it is replicated across diverse locations in the first instance. Gonzalez et al. present a network overlay to optimise data management across multiple data-centre sites [43]. Bessani et al. develop a software library called DEPSKY that utilises consumer cloud storage solutions from multiple providers to provide diverse replication for object storage [19]. Whereas Matos et al. give a different perspective providing the resilience from the client through enhancing the security of the client machine [75].

Methods of optimising the replication, typically for cost reasons, can be frequently found. Calis and Koyluoglu focus upon mitigating blocks (groups of storage nodes) that fail simultaneously [24], using Block Failure Resilient codewords that facilitate the recovery of a block from neighbour blocks utilising load balancing. Nachiappan et al. conduct a survey on both coding and replication techniques [83]. They suggest that a hybrid method of both is the only cost-effective way of guaranteeing storage resilience in the cloud.

Liu and Shen argue that the majority of resilient storage techniques in the cloud are effective at either correlated or non-correlated failures [68]. They propose a Multiple failure Resilient Replication scheme (MRR) accommodating both forms. They use a nonlinear integer programming approach to accommodate multiple objectives, which are a reduction in network latency to optimise

Table 3. Instance and VMM Resilience Techniques

Method	Model	Components	Disciplines	Limitations
Snapshots with error ranking [85]	IaaS	VI	FT	High replication cost and storage, state-management
Snapshots with VM introspection and anomaly detection [69]	IaaS	VI	FT	High replication cost and storage, state-management, windows only
Proactive recovery [96]	IaaS	VI	FT	High replication cost and storage, state-management, no physical fault protection
Periodic snapshots and recovery [54]	IaaS	VI	FT	High replication cost and storage, state-management
Process-level replication, checkpointing [36]	PaaS/IaaS	VI, RM	FT	Medium replication cost and storage, state-management, VM integrity
Process checkpointing [115]	PaaS/IaaS	VI, RM	FT	Medium replication cost and storage, state-management, VM integrity
VI replication [101]	IaaS	VI	FT	High replication cost and storage, state-management
VI replication [34]	IaaS	VI, RM	FT	High replication cost and storage, state-management
Replication, checkpointing of HV using introspection [21]	IaaS	PI, VI	RB, FT	Liable for anomaly subversion
Compiler and runtime software diversity [57]	IaaS	PI, VI, RM	SRV	Requires application source-code
VMM replication/mirrors with minimal overhead [127]	IaaS	PI, VI, RM	FT	High replication cost

VI=Virtual Infrastructure, RM=Resource Management, PI=Physical Infrastructure, FT=Fault Tolerance, RB=Robustness, SRV=Survivability.

consistency and the optimal number of replications and storage upon inexpensive media while still maintaining high availability.

Qu and Xiong attempt to optimise the quantity of needed replications for a specific use-case, which is unknown quantity of web requests [94]. Their method of replication across all nodes is in contrast to the typical micro-services architecture and raises questions of scalability and compartmentalisation. Wang et al. focus simply on ensuring the integrity of data in cloud infrastructures [123]. Their technique is unconcerned with whether the data are altered maliciously or through hardware fault. A block-based storage mechanism enables the location of data corruption to be identified to permit fast recovery. Finally, Yanez-Sierra take a holistic approach by attempting to provide resilience through modelling the entire data workflow, from client to cloud [128]. Their component-based approach allows modularity and features to be applied and examined at each stage.

2.2.2 Instance and VMM Techniques. Within this section, application layer resilience is referred to as examining an individual instance such as a VM or container and not considering the platform of multiple applications. Table 3 provides a summary.

Nguyen et al. argue that network and system fault-tolerance is well covered by conventional architectures and therefore they focus upon application resilience [85]. They propose a test bed facilitating error detection and recovery of cloud applications. They explain that errors may be detected long after their cause was executed and that tracing the exact cause can be difficult due

to multiple software layers and component dependencies. They cite the complexity of scalable software as a cause of these issues. To accommodate management of errors appropriately, they deploy an error ranking system according to severity. This enables selection of an appropriate action such as to ignore, restart, or revert. The software takes advantage of virtualisation within cloud environments to accomplish this. The implementation details are scarce, but it is important to note that this method does take into account generalised application faults so would detect malicious and non-malicious faults although it is not explicitly stated as a goal of the software.

VI is a common solution to issues such as intrusion due to the ability to revert the system in question to a clean state and migrate them to other nodes. Lombardi et al. present a system for enhanced resilience of Windows VMs, recovering from malicious and non-malicious faults [69]. The solution employs VM introspection and anomaly-based integrity verification to detect intrusions or errors arising. A reactive solution using the VI is then employed. However, it could be argued that repetition of malicious actions could allow a malicious user to plot the state changes taken and thus understand the intent of the IDS. Likewise during the evaluation of the system, the authors claim that despite a noticeable increase in resource use when implementing the introspection system, the attacker would not be able to detect the use of the system. Again, this is not a sufficient means of disguising the use of the system, as comparative analysis would result in wide variations in performance against a system that did not employ the introspection.

Reizer and Kapitza also employ VI for a proactive recovery (periodic node refreshing) system [96], primarily for intrusion relation failures. They explain that proactive recovery reduces support for recovery from genuine faults and also decreases system availability time. In their solution a domain is replicated across numerous, isolated guest domains where all network activity is proxied via a remote server. Diversity between the guest and primary domains ensures attacks will not affect the replica domains. This will hold true as long as the service is developed to be deterministic. Although unable to protect against a physical fault, this system will prevent against malicious and non-malicious faults. However, there are large cost and time implications due to the additional resources required.

Jhawar and Vincenzo propose a similar method named Remus [54]. It uses replication leveraging VI to provide a high degree of fault tolerance. The system periodically snapshots the host's state, storing a backup in memory. This ensures prompt availability. It is possible that anomalous data would cause both systems to crash, whereas the diversity in the previous work clearly protects against this. VM replication techniques are found often such as in References [101] and [34]. Egwutuoha et al. [36] and Tchana et al. [115] examine resilience at the process level. They provide replication and checkpointing of individual processes to recover from faults. These techniques could be considered more resource-optimal than checkpointing an entire VM while also more relevant to container-based architectures. However, the security implications of changing the integrity of a running VM are concerning.

Binun et al. focus upon the resilience of the VMM [21]. They present a novel self-stabilising hypervisor for increased robustness against malicious faults. A Stability Manager examines the VMM and its VMs for any misbehaviour, resetting the VM, software, or physical machine when a subversion is detected. While the system does provide an adequate method of resisting intrusions, it is uncertain whether it is possible to recover from a persistent threat. Without adequate constraints, it might be easy for an attacker to perform a DoS upon the machine through corruption, requiring a constant reboot. There are further performance issues, as with the integrity check, requiring the entire system to freeze.

Kanter and Taylor present a hypervisor that uses compiler and run-time techniques to increase diversity within an application, making attacks more costly [57]. Their combination of techniques prevents all memory addresses within the application being known to an attacker *a priori*, however,

Table 4. Virtual Networking Resilience Techniques

Method	Model	Components	Disciplines	Limitations
Mapping between virtual and physical, data centre hand over [22]	Phy/IaaS	PI, VI	SRV	Uniform DC distribution with minimal performance increase
Optimal overlay networks [13]	Phy/Iaas/Paas	VN vs PN	SRV, FT	High VNet design complexity
Leverage VN for layer selection [14]	Phy/Iaas/Paas	VN vs PN	SRV, FT	Complex high-level management
A framework for layer selection [15]	Phy/Iaas/Paas	VN vs PN	SRV, FT	Complex high-level management
Shared protection with backup links [12]	Phy/Iaas/	VN, PN	SRV	Lack of compartmentalisation
Scalable heuristic-driven shared protection [47]	Phy/Iaas/	VN, PN	SRV	Lack of compartmentalisation
Shared protection with rerouting [23]	Phy/Iaas/	VN, PN	SRV	Lack of compartmentalisation
Overlay networks creating path diversity [105]	IaaS/PaaS	VN	SRV	High complexity and centralised management
Selection of physical, virtual, or hybrid networking [15]	Phy/IaaS	VN, PN	R, FT	High-level simulation
Cost vs RB across different layers [48]	Phy+	VN, PN	R	High-level and constrained simulation
Cloud DDoS survey and mitigation framework [87]	Phy+	VN, RM	TT	Restricted failure types

PI=Physical Infrastructure, VI=Virtual Infrastructure, VN=Virtual Networking, PN=Physical Networking, RM=Resource Management, SRV=Survivability, FT=Fault Tolerance, R=Resilience, TT=Traffic Tolerance, RB=Robustness.

it does require the application's source code. This extreme case of diversity across the cloud infrastructure enables high resilience. The method is used in the deployment of an OS named *Bear* consisting of a minimal kernel and a VMM, where the kernel and all other components, including device drivers, are periodically refreshed with new, diverse replacements. A point of note is that it does not attempt to detect intrusions and operates without any further information. This is a useful characteristic that mitigates detection-related issues. They mention that performance is degraded due to the additional processes, the compile time decreases at 5% typically and sometimes up to 16%. However, the benefits of having a different set of binaries for every individual host in the cloud outweighs the performance hits.

Xu and Huang focus upon VMM resilience. [127], where execution of each VMM is replicated across a another. This provides resilience against hardware layer faults. Essentially providing redundancy for hardware, the system is successful with minimal overhead. However, the authors mention that the replication mechanism is not self-protecting.

2.2.3 Virtual Networking. Providing resilience within the networking layers of a cloud architecture is the focus of a number of works seen in literature. Resilience within networks may often be considered in terms of its survivability, distributed information systems being the focus of the term survivability [126]. As networking operates on a variety of different layers, the resilience may again differ, depending on the layer in question. Clouds may be distributed across multiple geo-locations and therefore require resilience on the physical layer. They also employ considerable quantities of virtualisation and with the advent of SDN networking in the upper application layers can become complex. Table 4 summarises the following literature.

Bui et al. investigate two methods for ensuring resilience in virtual networks [22]. They consider network resilience from the perspective of both the PN providers and VN operators and

in particular, the mapping between these two layers. Their resilience models involve handover to another DC during primary failure, with the resilience techniques involving source routing to the secondary DC. Two tests were conducted where data centres were uniformly distributed and locally paired. The results showed that the VN routing performed slightly better than the PN routing during uniformed location distribution, while during the paired locations there was very little difference.

Barla et al. consider the performance of network resilience methods [13] with the motivation of SLAs guaranteeing various quality of IT services but not end-to-end communications. VN is cited as a solution to this issue. The study simulates two scenarios: in the first the VN is responsible for the resilience, and in the second, the PN operator is responsible for the resilience. The results indicated that the VN always outperformed the PN. Complexity of the design is mentioned for consideration. The authors expand this further in References [14] and [15], where they provide models for developing the VNs. These were shown to outperform previous approaches by finding a resilient solution in every case, drastically reducing communication delays. Barla et al. next examine resilience in VNs using redundant back-up links (referred to as shared protection) [12], similar to the approach employed in Reference [71] for physical networks. They use redundant resources shared among multiple VNs, accomplished through appropriate information exchange between the PN operator and VN operator. As before, the VNs outperformed the PNs with the addition of cost-saving benefits through optimised set-up. This work highlights the effectiveness of providing high-level resilience.

Harter et al. [47] confirm the benefits of shared protection mechanisms, with cost savings of 10%–20%. This model includes heuristics to make the algorithm highly scalable.

Bui et al. consider the problem of mapping VNs to PNs under varying time constraints [23]. Their solution enables the selection of appropriate PN and (consequently) DC resources for resilient rerouting of networks under varying time constraints. As an improvement upon schemes that allow bandwidth sharing through various backup links, this system allows backup links to be reprogrammed as needs change over time. Their results show that the benefits of reconfiguration are only applicable if the standard working paths are also reconfigured. Also, they are only applicable if the majority of traffic "does not have Quality of Service (QoS) requirements that prohibit path reconfiguration," which could be an issue when considering the variety of use cases for cloud traffic.

Secci and Murugesan provide discussion regarding the current cloud network architectures [105]. They stress the need for resilient clouds, as without resilience their services are sub-optimal or even useless, due to the service-oriented nature. They explain that conventional cloud RM is considered "dumb" due to over-provisioning of resources and, in particular, inefficient methods of bandwidth utilisation. They argue that this has caused high centralisation in geo-distributed clouds, which contributes to high risk of failure and, therefore, low resilience. As services must be distributed across multiple cloud services for them to be resilient, this conventional bandwidth utilisation is at odds with this requirement and, therefore, the authors suggest that further decentralisation is necessary. To provide further decentralisation, appropriate overlay networks must be employed to ensure network paths diversity and DC end points to ensure the necessary resilience. The authors conclude by noting that resilience is not just a requirement of highly dependable services but necessary to fulfil the fundamental cloud SLA. To provide this, the current architecture must change.

Harter et al. provide a comprehensive discussion regarding the comparing the resilience of different layers and a novel consideration of the business-oriented responsibilities of each cloud delivery scenario [15]. They provide a method, determined through simulation, to determine the most favourable layer to provide resilience depending on the use-case. They conduct a similar study, investigating which layer is the most effective in terms of cost and fault tolerance to provide the

Table 5. Cloud Management Resilience Techniques

Method	Model	Components	Disciplines	Limitations
Comparison of task-scheduling algorithms [25]	IaaS	RM, SCH	RB	VM only
Energy-aware scheduling for software FT [40]	IaaS	SCH	FT, DT	Power-related faults only, resource-intensive modelling
Monitor and optimal selection of hosts [65]	PaaS	SO	RB	In-depth application analysis
TCloud, a modular middleware and layered multi-cloud solution [120]	PaaS/IaaS	DC, RM	SEC, RB	Strong interoperability issues, high cost, contentious reliance upon trust
Resilient state management using a remote handler [106]	PaaS	VI	FT	Single point of failure
Chains of virtual network instances [103]	IaaS	VN, VI, SO	SRV	Centralised management
Brownout experience downgrade [60]	PaaS/IaaS	VI, RM	RB, TT	Single replica only, experience reduction
Brownout experience downgrade with LB [61]	PaaS/IaaS	VI, RM, LB	RB, TT	Single replica only, experience reduction
SO using multi-agent monitoring and feedback [116]	PaaS/IaaS	VI, RM, SO,	FT	High communication cost
SO using diverse OS configurations [45]	PaaS/IaaS	VI, SO,	FT/SRV	Conceptual only
Service to hardware dependency modelling, migration, and FT monitoring [77]	PaaS/IaaS	VI, VN, RM	FT/SRV	Static application requirements
SO using self-organisation and self-management [26]	PaaS/IaaS	RM, VI, SO	SRV	High complexity and resource-intensive monitoring

RM=Resource Management, SCH=Scheduling, SO=Service Orchestration, DC=Data Centres, VI=Virtual Infrastructure, VN=Virtual Networking, LB=Load Balancing, RB=Robustness, FT=Fault Tolerance, DT=Disruption Tolerance, SEC=Security, SRV=Survivability, TT=Traffic Tolerance.

resilience [48], this time considering PN, VN, and overlay networks. They give a framework for ease of selection.

Osanaiye et al. focus upon one particular attack type/resilience problem [87]. They provide a survey and framework of DDoS mitigation techniques in the cloud. While effective as a traffic-tolerant technique, the lack of coverage for diverse failure types leaves this form of work behind the others.

2.3 Cloud Management

This section reviews literature applying techniques that are operated upon via the cloud-management layer. Tables 5 and 6 provide a summary.

As redundancy is a key component of resilience, task placement can influence its efficacy. Cartlidge and Sriram present an analysis of the effect of different scheduling algorithms on resiliency [25]. This should be considered as resiliency of IaaS. They evaluated random, packed (FILO) and clustered VM allocation, showing that packed was the least resilient, which is obvious when considering single point of failure. Additionally, their results illustrated a clear link between hardware redundancy and resiliency, although the pack scheduling algorithm was not always consistent. When adjusting the DC architecture, they concluded that the pack was sensitive to infrastructure types. Although this work is intuitive, it is interesting when understanding if resilience designed at high layers can overcome the shortcomings of poor resiliency at lower layers.

Table 6. Cloud Management Resilience Techniques II

Method	Model	Components	Disciplines	Limitations
Checkpoint-restart of stateless applications upon decentralised DHT [70]	IaaS	VR, SO, RM	FT, A	Stateless only
Self-reconfiguring moving-target defence [122]	IaaS	VI, RM,	FT	High complexity and replication cost
Optimised checkpoint-restart for HPC using VI [86]	IaaS	VR, RM	FT	High replication cost
GA-based SCH, homogeneous spread of components across all nodes [39]	IaaS	VI, SCH	A, FT	High initial resource cost and web app only
Balance reduce for data locality and job time reduction [8]	IaaS	VI, SCH	FT	Constrained use-cases
Batches of jobs weighted to prevent malicious faults [66]	IaaS	VI, SCH	FT, SEC	Pull and then process can cause synchronisation errors
Component quality ranking and service construct to reduce faults [132]	IaaS	VI, SO,	FT	High complexity
Risk minimisation using content ranking and placement [38]	IaaS	VR, PR, DC	DT	High complexity
Mixed integer-linear programming [5]	IaaS	VR, PR, SCH	A	Conceptual online
Replica-oriented middleware [51]	IaaS/PaaS	VI, RM	FT	Complexity and resource cost
Multi-component middleware to enforce policies, determined by anomaly detection [107]	Phy/IaaS	RM, SCH, PM	FT, SEC	Complexity and resource cost
Networking protocol (sockets, web, etc.) replication [131]	IaaS/PaaS	VI, VN, RM	FT	High complexity and resource cost, web sockets only

VR=Virtual Resources, SO=Service Orchestration, RM=Resource Management, FT=Fault Tolerance, A=Availability, VI=Virtual Infrastructure, SCH=Scheduling, PR=Physical Resources, DC=Data Centres, PN=Physical Networking, SEC=Security, VN=Virtual Networking.

Gao et al. present an energy-aware scheduling algorithm for cloud resilience [40]. Their framework allows reliability when faced with soft-errors, often a consequence of varying voltage levels. It has a performance increase of up to 50% achieved through a hybrid method, first conducting an assessment of static reliability requirements that then leads into dynamic analysis that can occur at run-time. Their implemented system also considers financial data, which is an often overlooked yet is a principal component of service-oriented cloud systems.

Liang and Lee consider resiliency when developing PaaS clouds [65]. They take a robustness approach that allows varied and unexpected program input. Their work appears to be concerned with reliability through analysing sub-component effects upon the application. A SO approach is applied to minimise failures through accurate selection or replacement of individual components, maintaining a low failure rate.

Verissimo et al. present a novel paradigm, *cloud-of-clouds*, with their system: TCloud [120]. They argue that DC distribution is not enough to provide resilience of applications within a cloud, as the security aspects of federated clouds are not addressed. To provide resilience, the authors argued that a user must be able to combine clouds from multiple providers providing high diversity. Additionally, open architectures are necessary to prevent proprietary vendor lock-in and security features from the lower layers up. Their system accommodates these requirements through providing multiple solutions to "build layers of progressively more trusted components and middleware systems," which allows layers on the top layer to be trusted due to trusted lower layers. A flaw in this system is that trust in a lower layer cannot always be guaranteed. The authors suggest an intrusion detection system as an example of lower-layer security, which is regularly circumvented

to allow malicious traffic obfuscation. The system includes modularised components to provide middleware to enable the paradigm, providing a secure and resilient PaaS.

Sharma et al. present the development of a resilient PaaS leveraging state management, known as ReLo [106]. Resilient state-management enables a session to persist during application down time. Within their system, agents reside within an application. If the agent goes down, a handler agent redirects the session to another application. Essentially the system employs redundancy via a middleware management solution. The single point of failure with the handler agent is a questionable choice. The authors mention that memory constraints and router time-outs have a negative effect upon the resilience.

Scholler et al. describe their method of deploying vNFC (chains of virtual instances, providing networking services) [103] resiliently using their Tenant Infrastructure Management Software (TIMS) upon OpenStack. Different components within the network service have different resiliency requirements, (e.g., scalability, redundancy). The service describes the requirement and TIMS manages it appropriately. Both resource and network requirements are given through Application Layer Traffic Optimisation (e.g., maximum delay between two components). The system is dependent upon OpenStack's availability zone feature, which groups pools resources, which allows critical components of one service to be grouped in different failure locations. The authors identified a number Openstack shortcomings such as low resource information within the pools.

Klein et al. discuss the brownout programming paradigm [60], proposed to provide enhanced robustness within cloud services. It attempts to mitigate the requirement to provision large amounts of replicated instances during traffic increase. This should prevent service run-time failures such as flash crowds. A brownout program will downgrade a user experience, such as with enhanced features, to prevent excessive use of the system. They extend this by combining with load balancing, as the combination currently creates conflict [61]. They propose two novel algorithms and a production-ready load balancer. Their results indicate strong performance compared to alternative solutions.

Torres and Holvoet examine service composition architectures [116]. Their decentralised system relies upon two distinct agents: the first monitors the network for appropriate, available subtasks to compose a service with; the second evaluates available resources within the system. These two agents continuously and dynamically assess the current status of the service, enabling a rapid response to faults. Each agent delegates work to lower-level agents, which would appear to be biologically inspired by ant processes. Empirical evaluation indicated that performance was better than the common, reactive approach, with lower composition times of between 4% and 25%. The authors note that the system suffers from high communication costs.

Minzhe and Prabir investigate diverse replica software in Reference [45], where the configuration of the OS in which the service is built upon is varied across the service. The authors present a game theoretical approach to the problem.

Mihailescu et al. consider the mapping of components of a service to hardware resources [77]. Their algorithm is more optimal than global shuffling algorithms and will eventually converge on a stable global configuration as long as application requirements remain consistent. The system improves resilience from hardware faults and network errors through understanding component inter-dependencies. It models them as graphs where a division equates to VM migrations. The system is dynamic, allowing an end-user to select the required resiliency. Further work might consider the effect of cost upon this feature.

Carvalho et al. take a biologically inspired approach to cloud [26]. They employ a multi-layered method with a focus on distributed service management. The authors focus on mission continuity and survivability during attacks. The work focuses upon the application of bio-inspired methods of self-organisation and self-management, as well as distributed coordination, to the

service discovery and orchestration processes in the cloud. The system layers consist of first detecting the damage through distributed sensors, optimised resource management, and then response/immunisation to the threat.

Louati et al. focus upon stateless applications, which are easier to provide resilience to than statefull [70]. Their solution uses checkpointing and application restart combined with a back-end built upon Distributed Hash Tables (DHT) for resilient decentralised storage. Nicolae and Cappello also consider checkpointing/restart of applications to mitigate failures [86]. This time for High Performance Computing (HPC) applications using efficient Virtual Disk Image (VDI) snapshots.

Villarreal-Vasquez et al. argue that the typical replication techniques employed for cloud resilience increase the attack surface of an application and thus are detrimental [122]. They propose a solution that uses Moving Target Defence (MTD), migrating instances once an anomaly has been detected and providing self-reconfiguration to return to baseline state.

Frîncu applies Genetic Algorithms (GA) to component scheduling optimisation [39]; high availability web applications within the constraint of cost is considered. Two distinct algorithms are considered: the first optimises the maximum number of components upon each node within the cost; the second is sub-optimal, finding the minimum required so that the application is still available given that all but one node fails. Antony et al. also investigate scheduling to optimise resource usage, this time for bandwidth consumption [8]. They provide a heuristic that optimises data locality to reduce the job completion time and provider fault tolerance to the Balance Reduce (BAR) algorithm.

Liao and Cheng propose a resilient scheduling method that involves servers retrieving batches of jobs and then processing them according to a specific weighting to mitigate the effect of a malicious fault [66]. Zheng et al. take a similar approach [132]. They rank a component's value and orchestrate a service so that a fault will have a reduced or no impact upon operation. Ferdousi et al. take a similar approach [38], again applying ranking, with a greater focus upon the placement of content as opposed to the components themselves.

Al-Ayyoub et al. provide a framework that leverages mixed integer linear programming to consider multiple objectives to optimise cost-effective resilience across all levels of cloud infrastructure [5].

Imran et al. developed A middleware that uses watchdogs, checkpointing, and journaling [51]. It is used to create, back up, and store replicas of application to provide fault tolerance. Whereas Zhao et al. provide a replica-oriented middleware yet with comparatively considerable resource optimisation [131]. While only for replications of network protocols (sockets, web protocols, etc.), it is an interesting approach—although the integrity and confidentiality of the application is in question.

2.4 Decentralised Cloud Resilience

While the previous sections discussed centralised cloud architectures residing in data centres, this section presents decentralised cloud models such as fog and edge computing. Table 7 summarises this literature.

Due to the constrained nature of IoE devices, data processing, storage, and representation must be provided by a third-party platform, typically the cloud. However, the high latency, non-deterministic wireless mediums and high volume of data make this relationship difficult. Fog computing is the medium in which pseudo-cloud services, mostly temporary data processing, are provided closer to the edge of the network. Processing data in this form has a greater requirement for resilience due to device mobility, open wireless mediums, constrained device resources, heterogeneous device types, cyber-physical systems, and hostile environmental conditions.

Service orchestration (SO) is an important process to conduct securely in fog computing. To optimise constrained device resources, only the minimum amount of nodes necessary will be

Table 7. Decentralised Cloud Resilience Techniques

Method	Model	Components	Disciplines	Limitations
SO using Attribute-based encryption [121]	PaaS/SaaS	EDGE, RM, SO	C, I	Resource-intensive cryptography and hierarchical network structure
Multi-component federated fog architecture [97]	IaaS/Paas	RM, EDGE, CONST, TRAN	SEC	Conceptual model for resilience, high complexity
Hierarchical data replication and service downgrade using p2p networking [63]	Paas/SaaS	CONS, EDGE, CONST	A, DT	Questionable energy consumption per task
Fog computing for resilience [80]	IaaS	FOG	SRV, DT, TT	High resource cost and governance issues
Decentralised SDN for 5G VANETS [50]	IaaS	CONS, IOT, TRAN, EDGE	DT, SRV	Centralised management
Fog-enabled anomaly detection for SDN [79]	IaaS	TRAN	SRV, SEC	Centralised management
SDN middleware for critical events using rerouting and backup links [18]	PaaS/SaaS	EDGE, TRAN	SRV, FT,	Single point of failure
Anomaly detection and moving target defence [56]	IaaS/PaaS	VI, CONST, RM	I, FT	Questionable resource cost
Watchdog-based multi-layer programming architecture [37]	PaaS	VI, VN, RM	FT	High complexity
Dependency mining for replica prediction and optimisation [9]	IaaS/PaaS	VI, RM	FT	Theoretical
Agent-based spot-instance survival reasoning [10]	IaaS/PaaS	VI, RM	DT	Specific to one cloud provider
Uncoordinated application checkpointing and replication [88]	IaaS/PaaS	VI, RM	FT, SRV	Single point of failure
Resource-predicting hybrid mobile cloud [59]	IaaS	PH	DT	Dependent upon fixed nodes
Game theoretic with Bayesian approach to SCH during real-time attack [28]	IaaS	SCH, VI, EDGE	A, SEC	Attack-specific

C=Confidentiality, I=Integrity, SEC=Security, A=Availability, DT=Disruption Tolerance, SRV=Survivability, TT=Traffic Tolerance, RM=Resource Management, SO=Service Orchestration, CONS=Constrained Devices, TRAN=Transportation, EDGE=Edge Devices, VI=Virtual Infrastructure, FOG=Fog Computing, IoT=Internet of Things devices, VN=Virtual Networking, PH=Physical Hardware, SCH=Scheduling, FT=Fault Tolerance.

provisioned for an end-user. This necessitates service requirements to be broadcast for a network that provides a number of security issues, particularly confidentiality. Viejo and Sánchez use Ciphertext-Policy Attribute-Based Encryption (CP-ABE), whereby nodes will have keys corresponding only to the attributes they are allowed to process [121]. Their network is structured hierarchically so that nodes pass messages to those it can control further down the tree. The nodes will require a generalised key. For example, a message containing "temperature" will also need a "weather" key to process it. These messages form policies such as "temperature, zone 1," which are then encrypted separately and transmitted. If any messages can be decrypted by a node it means that further nodes in the hierarchy can also be decrypted so the service discovery can continue. Once the service has been orchestrated between the required nodes, the client and nodes exchange keys to communicate securely. Chejerla et al. instead chose to develop a scheduling algorithm that uses a game-theoretic and Bayesian approach to mitigate against attack in real time for Cyber Physical Systems (CPS) [28].

Rios et al. explain that modelling fog networks in a hierarchical manner, with a singular provider, is oversimplified and detrimental to its security [97]. They should instead be considered as a federated architecture with numerous service providers within different trust domains. The authors propose an architecture (SMOG) to provide resilience in fog networks. It consists of a number of baseline characteristics such as *secure interconnection*, *authentication and authorisation*, *protection of virtualised environments,* and *situational awareness*. They list enhanced characteristics as *trust services*, *distributed decision making*, *privacy capabilities,* and *digital evidence management*. They explain that these baseline requirements are largely missing from literature and are necessary to ensure a secure and resilience fog.

Edge nodes are without doubt a point of failure in any decentralised cloud network. Le et al. give a solution to partial failures in MEC (e.g., connectivity loss) between the edge nodes [63]. Their architecture is again hierarchical, with mobile nodes storing local backup data dispersed among them. If partial failure with the edge nodes occurs, the devices switch to a P2P model, processing data collaboratively. This is an alternative mobile computing model, and the results show good time-reduction performance when the task is disrupted across the nodes. However, the power consumption is likely to be highly variable according to the difference between nodes and therefore the suitability will not be universal.

Modarresi and Sterbenz consider fog computing as a solution for resilient IoT/edge computing in Reference [80]. They argue that the uncertainty surrounding resource, link, and bandwidth availability ensures that typical edge computing is not resilient for IoT processing. For example, too many clients can overload resources and thus cause a denial-of-service. They argue that the introduction of fog nodes between the edge and the cloud creates greater autonomy within the network. If a connection is lost between the edge and the cloud, the fog maintains this network and increases the *survivability* of the ecosystem. Further to this, they suggest that the diversity of standards, protocols, and network links, which cause fog computing to be quite complex, is actually beneficial to its resilience due to the increase in variety. They also indicate that through fog-reducing traffic further in the core and distribution network, its implementation provides *traffic tolerance*. Finally, *disruption tolerance* is enhanced through a reduction in latency, permitting applications to be processed quicker and thus any disruption has less impact. The authors support these statements with numerous simulations inclusive of the fog environment.

Hussein et al. provide a mobile edge computing solution that applies Software Defined Networking (SDN) to 5G, providing resilient processing to Vehicle Area Networks (VANETS) [50]. Safety concerns are paramount in vehicles and as such so is the resilience of VANETs. Their proposed solutions provide enhanced security through an additional security layer using SDN. As opposed to a traditional centralised SDN approach or a traditional distributed VANET approach, they present a hybrid method. A centralised 5G base station is used to manage SDN security functions distributed across a number of roadside controllers. This approach illustrates a strong example of custom networking hierarchy technologies being supported at the edge for specific use-cases and resilience requirements.

Modarresi deploys SDN again in tandem with fog for resilience in Reference [79]. This time fog nodes are used to detect anomalies in network traffic and notify the SDN controller. This can make security-focused decisions about what traffic to drop or restrict—a strongly illustrative example of the application of fog for greater network resilience, although it does not help to strengthen resilience of the fog nodes themselves.

Bensen et al. take a middleware approach to provide continued operation of critical events from IoT devices when their connection to the cloud fails [18]. Their system contains two components: the first periodically probes different paths to the cloud, detecting possible faults or failures; the

second provides multicast message dissemination according to information received from the first component. They again use SDN to provide this information and use it to create "resilient overlays." This middleware approach enables varied support for IoT devices, as the middleware works seamlessly.

Kahla et al. provide a solution to low trust in IoT environments [56]. They leverage moving target defence to migrate targeted or subverted virtual instances to another host fog machine. It is not clear how this would prevent a number of different attacks or heal the instance once it had migrated, although the autonomic aspect of integrity verification is commendable.

Eisele et al. state that resilience is necessary to consider in edge environments due to both resource and network uncertainty [37]; while security is important due to the resource-constrained nature of edge devices preventing virtualisation providing adequate isolation. They propose a novel programming paradigm: RIAPS (Resilient Information Architecture Platform for Smart Grid), which provides a platform for distributed applications to be deployed resiliently. The platform provides a diverse number of different services and managers (such as for security, persistence, fault management, etc.). While the platform appears to be complex and thus has an increased attack surface, given the number of required components, it illustrates the notion of an underlying platform providing resilience to higher levels.

Arval et al. use Bayesian belief networks to mine dependencies between replicated edge nodes. Their solution uses past server performance from logs and temporal dependencies to highlight the probability of when failures may occur concurrently. Although their current solution is theoretical, it shows strong optimisation through replica reduction [9].

Neto et al. tackle a somewhat different resilience problem [10], where fog-enabled service does not suffer a fault but an outage related to the CSPs SLA. They focus on Amazon's Spot Instances, which are transient servers acquired by the user when the maximum they wish to pay (bid) is greater than the value of the instance. Due to the nature of this acquisition, the continued operation of these servers cannot be guaranteed. Therefore, in this fog platform the failure results from the unavailable CSP back-end. To mitigate this, they propose an agent-based case-based-reasoning solution that aims to predict the survival time of an instance. This enables checkpoints to be made to resume the work in case of application fault. Their solution could be modified for application processing closer to the edge, although the resource requirements for checkpoints must be considered.

Ozeer et al. have a similar focus on recording and reverting to application states [88]. They take an uncoordinated approach, recording application events with a corresponding recovery timer. Expiration indicates lack of synchronisation with the physical world and therefore can't be ignored. Event details are logged in a global and failure-free storage system to permit recovery to any node from a central location. This centralised storage suffers from central point of failure. The authors present a competent yet complex solution consisting to enable system fault tolerance. The question of how failures are to be handled in the system handling the failures is still open.

Khalifa et al. move away from a traditional cloud architecture, improving the resilience of Hybrid Mobile Clouds [59]. Mobile clouds require greater resilience than a static system due to the dynamic network characteristics. The proposed architecture is interesting due to its flexibility in running on diverse devices, essentially ignoring the underlying hardware. The resilience requirements are aided through a resource prediction mechanism and an early failure detection mechanism to facilitate handover of vital services. The system proves successful, although performance is still dependent upon the quantity of fixed nodes within the cloud, making the system not purely mobile. However, overall it exhibits a good example of how cloud systems can be built upon non-deterministic environments.

Table 8. Alternative Architecture Resilience

Method	Model	Components	Disciplines	Limitations
SlapOS—cloud distributed across homes [113]	IaaS	VR, RM, CONS	R	Low/non-deterministic redundancy
Modular and highly diverse SlapOS [31]	IaaS	VR, RM, CONS	R	Low/non-deterministic redundancy
Diversity using community clouds [41]	IaaS	VR, RM, CONST	DT, SEC	Low redundancy, uncertain governance
DefCloud using strong diversity across all layers [111]	IaaS	VR, PR, RM	SRV, FT,	High complexity
MEERKATS constant evolving diversity [58]	IaaS	VR, RM	SRV, FT,	Resource-intensive
DREME replica execution diversity [16]	IaaS	VR, RM	SRV, FT,	Resource-intensive
SCE+ using mass redundancy [99]	IaaS	VR, RM	FT	Resource-intensive
BioRac, cell-based diversity and redundancy [46]	IaaS	VR, RM	SRV, FT,	Resource-intensive, high Complexity
SDN for resilient industrial IoT [100]	IaaS	CONS, CONST	R	Centralised management, high complexity

VR=Virtual Resources, RM=Resource Management, CONST=Constrained Devices, CONS=Consumer Devices, PI=Physical Infrastructure, R=Resilience, DT=Disruption Tolerance, SEC=Security, SRV=Survivability, FT=Fault Tolerance.

2.5 Alternative Architectures

Some work will choose to encourage a conventionally different cloud architecture to provide increased resilience. Table 8 summarises this literature.

An alternative to the infrastructure layer, Suciu et al. present SlapOS [113]. They choose to provide a purely distributed cloud architecture where single point of failure is remedied through distributed cloud resources over multiple PCs within homes, as opposed to within DC. While this might bring forth bandwidth, capacity, and latency issues, the benefits of reducing single point of failure are considerable, particularly for safety-critical events such as during disasters.

Courteaud et al. consider further resilience of SlapOS [31]. They refer to the concept of community cloud, whereby the cloud is collaboratively built from personal devices. The main current issues are summarised as: (1) Migrating from commodity cloud to resilient, secure, and dependable clouds, (2) promoting diverse and open ecosystems, and (3) building a coherent, modular, and reusable architecture. They also consider the leader selection problem (the process of selecting the next master node after loss of the current). Further issues include: implementation and accurate failure-detection methods, and methods of replicating the master database prior to handover to another master node. The authors note that conventional delivery models (IaaS/PaaS/SaaS) become obsolete. Finally, the authors explain that an implementation of hierarchical masters (such as with DNS) will be implemented for increased resilience. While the architecture and delivery model is certainly interesting, there are issues directly relating to resilience concerning master node hierarchies that undoubtedly cause problems. A decentralised system such as this is not as resilient as one that is purely distributed.

Garlick also considers community cloud-based resilience [41]. They promote the model as an enhancer for organisational resilience. As with SlapOS, the author highlights the ownership and location of current cloud models being unsuitable for providing resilience. The authors note that for natural disasters, centralised disaster recovery is too late and excessive. They argue that disaster recovery must be conducted at the community level. The breakdown of communication networks is cited as a key issue, where the more effective communication was developed by the decentralised

communities. The author explains that community cloud models enable the benefits of public cloud offerings with greater control. Issues surrounding community clouds, such as malicious users, are said to be mitigated through user vetting, a process that may not always be practical or effective. Sathiaseelan et al. present a similar discussion in Reference [100], where they similarly highlight that commodity hardware-based community clouds have considerable advantages over resilience due to the highly decentralised nature.

Sterbenz and Kulkarni present DefCloud [111], which attempts to provide greater resilience through increasing diversity and redundancy within all layers of the cloud architecture. Highly flexible, it allows resilience to be adjusted in a "service-aware manner." This might be argued to be similar in concept to the usability vs security trade-off. Such a feature is likely necessary for a cloud platform that accommodates a wide spectrum of use-cases. The authors argue that the first point in designing the infrastructure is the removal of monoculture, as it enables malware and attacks to propagate effectively through only needing to attack one type of hardware architecture or software application. This concept is then applied to all layers of the cloud. First is the *infrastructure layer diversity*. This encompasses *data-centre diversity* and *cloud diversity*. They consider DC diversity where sub-trees of features where similar trees are not selected in tandem to maximise diversity. For example, similar trees will utilise the same network vendor hardware or operating systems. While diversity provides resilience against security related failures, it does not protect against failures from direct physical DC attacks, e.g., natural disasters or military attacks (such as an EMP). To mitigate these issues, the architecture applies cloud diversity through distributing the cloud over multiple geo-locations, using varying ISPs. After the infrastructure layer, the DefCloud then assures resilience through *Process-level Program diversity*, where diversity focuses upon distribution via space and time. Spatial diversity is concerned with distribution of different software versions. Temporal diversity is concerned with varying application configurations over time. Application diversity ensures binaries are diverse, e.g., an attack on one application binary will not apply to another. While this has consequences on the current state of 0-day exploits, it complicates the software development process. Although DefCloud undoubtedly covers resilience in the cloud through adaptations of the conventional architecture, the system lacks real implementation or simulation and thus its resilience is yet to be determined. For one, the complexity of the system is clearly greater, which increases the attack surface.

Keromytis uses similar diversity in their MEERKATS system [58]. It is a fully novel architecture for a security mission critical cloud. The system constantly evolves across all aspects, reducing monoculture and increasing diversity. One component of the system, DREME [16], is concerned with execution diversity of replicas and provides a framework for I/O redirection.

IBM presents a somewhat novel architecture named SCE+ [99], which is built from the ground up to be highly resilient. The authors make the distinction between typical cloud architectures employed by Amazon and Google by explaining that they are constructed from "redundant, inexpensive, expendable building blocks," whereas the IBM SCE+ employs "high-end building blocks with significant internal redundancy and an established track record of very high MTBF for every element." It would appear that the contrast is in SCE+ employing mature and extremely resilient fewer components with conventional architectures employing many less mature components and relying upon replication/redundancy. The architecture applies resiliency to differing cloud layers. The physical layer is designed to avoid single point of failure through division of resources and replication in separate geo-locations with a backup dark-fibre link. Software resilience is then considered from multiple aspects. Components are deployed in redundant pairs and constant "health-checks" are in place to monitor correct functioning. In addition, redundancy of data and regular backups ensure resiliency within the data layer. The authors explain that standardisation of hardware within the system components aids the resilience; however, this is contentious, as diversity

within hardware is surely a necessity for resiliency. They cite virtualisation as an enabling factor of the resilience; however, this is typically a component of cloud infrastructures anyway and, therefore, offers the environment no additional advantage. Overall, the architecture offers a variety of additional components for resilience, although some are questionable, such as the physical distance between components as well as the added complexity within the system.

Hariri et al. present an architecture based on biologically inspired processes that allows tunable redundancy at multiple cloud levels, known as BioRAC [46]. One layer of the architecture involves division of components into "cells," which allows dynamic real-time configuration and combine together to form an "organism" that is then applied to a particular goal. In an additional layer, the system provides high levels of diversity through varying execution and finally it provides intelligent algorithms for collaborative threat alert and detection. Although lacking an implementation, the architecture is interesting for providing a system designed with resilience from the ground up with novel components, as opposed to those adapted on top of conventional systems. However, the system complexity due its multiple layers has an adverse effect upon its resilience.

2.6 Evaluation and Models

As with the resilience disciplines, measurement of cloud resilience could follow traditional performance-based resilience metrics such as Mean Time Between Failures (MTBF) and Mean Time Taken to Repair (MTTR) and the corresponding availability that is easily calculated from the two. However, these metrics could be considered primitive at best [30], considering the complexity of these environments. The Resilinets model [112] provides a method of determining which resilience features are available through binary selection of distinct features (e.g., the network provides confidentiality or it does not). Other non-cloud-specific resilience metrics also suffice, such as graph metrics. Graph metrics are noted for their ease of comparing distinct architectures, as they examine the structural characteristics of a network. Alenazi and Sterbenz evaluate a number of graph metrics for resilience in References [6] and [7]. These include elementary metrics such as the quantity of nodes, node connectivity (the average number of connected nodes to each other node), node centrality (the most important nodes), and so on. They also include those specifically for network resilience through removal of links and nodes, e.g., network criticality and effective graph resistance. All of the above are arguably strongest when examining a distinct service as opposed to the entire cloud environment. Some of the works surveyed according to layer perform some form simulation of a model to evaluate resilience within the context of that particular use-case; the following works focus upon more generalised models for resilience.

Jabbar states that resilience is more difficult to measure than traditional security metrics due to the need to evaluate how effectively the service is still being delivered [52]. They propose that resilience should be measured as a state space considered in terms of degradation, where a service is more resilient if it contains more states in which it stays operational and not severely degraded. Such a high-level approach may be applicable to complex environments.

Ghosh et al. provide a model for resiliency based on stochastic reward nets [42]. The work is interesting in that the metrics for resiliency focus upon evaluating how effectively the job is scheduled through Quality of Service metrics. Those given are the rate of rejected jobs and the delay in VM provision. Following from the definition of resiliency, "quantification of service delivery during changes," the authors evaluate changes as fluctuations in job arrival rate and the quantity of physical machines. Their results showed a faster provision rate was more resilient. Also that removal of a hot physical machine has an adverse effect upon resiliency, whereas removal of a cold one has a minimal effect.

Ju et al. evaluate the resilience of OpenStack [55]. They develop a novel fault injection framework for both the architecture and its services. They uncovered 23 different bugs that developed

into faults in the system, highlighting the lack of effective resilience considerations within the stock cloud management software.

Tu and Xu present a resilience model built for a typical IaaS cloud using Eucalyptus [118]. They explain that resilience and robustness are strongly connected in complex systems, where both properties describe the system's ability to react to disturbances but vary in how they do so. Considering the cloud as a multi-component, hierarchical system, the model evaluates the component interacting and interdependency upon resource consumption. Resilience is modelled by the strength of interactions between the components, where the strength is the percentage needed to consume from another component. A disturbance within the system results in a large queue, exhausting resources causing the services to fail. The authors describe system-wide resilience as the quantity of processes that fail due to the inability to consume. They note that this system does not take into account factors that may influence the interaction strength such as one-to-many and many-to-one resource consumption interactions. It is mentioned that resiliency is accomplished through redundancy, which has an adverse effect upon cost. To fit in line with the author's model, they explain that increased redundancy weakens the requirements for resource consumption links between individual components. While redundancy is a key component of resiliency, it is not the only method, and poor implementations can even reduce resiliency under certain circumstances. The authors then attempt to understand more about this effect, examining replication algorithms with modularisation of a cloud system. Their results show that as size increases, modulation is more important to prevent duplicate replication updates. However, they also mention that poor modularisation implementation can create a single point of failure and thus become an enabling factor for poor resilience.

Scholler et al. present an archtiectural model that enables insight into the security implications of cloud architectures [49, 104]. Their motivation is that current cloud services do not accommodate security and resiliency for critical infrastructures. Their model distinguishes between the different roles (such as the physical provider, service developer, and service user) as well as the different infrastructures (the physical and virtual) to assess the given requirements against the system. It promotes greater logging for audit purposes, as well as increased transparency between the physical and virtual layers, to increase trust between the users. Arguably, many issues within current cloud architectures ensure their unsuitability for a wide range of critical infrastructure services.

Sousa et al. conduct an evaluation of Quality of Resilience evaluation criteria within the cloud to activate appropriate proactive resilience measures [109]. They propose to use multiple criteria to evaluate the resilience, partly due to the wide variety of requirements associated with resilience and also because many proactive mechanisms require further information. The authors implement proactive resilience systems using multiple criteria for the cloud, MeTH [108] and TOPSIS [117]. The results showed that both methods improved the resilience of protocols, which were unable to detect cloud layer faults, but MeTH provided the greater performance in both fault and non-fault scenarios.

A classification of types of resilience metrics found within cloud computing is described below:

- **Binary feature**–based metrics are those relating to the Resilinets model such as confidentiality that either exist in the service of cloud or do not.
- **State-based** are those that examine the degradation of service to determine when resilience has failed.
- **Performance-oriented** metrics are the traditional type such as MTTR or QoS, which typically involve examining one distinct service.
- **Graph-based** metrics examine issues in topologies such as network criticality.

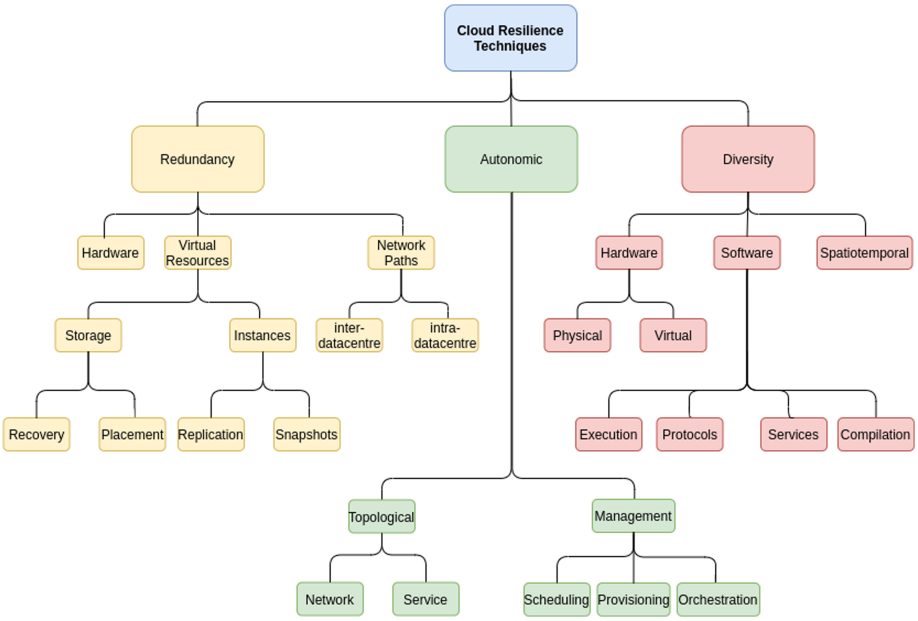


Fig. 2. Classification of techniques used in cloud resilience.

- **Multi-criteria** metrics aggregate and summarise a variety of metrics into one to take into account very complex systems.

3 STATE-OF-THE-ART ANALYSIS

After reviewing the work in the previous section, an analysis of the state-of-the-art of resilience in cloud literature is given in this section. It summarises the techniques used at each layer and the limitations of these techniques within the context of resilience in cloud environments. The complexity of the cloud environment is reflected in the multitude of characteristics and methods involved that enable resilience in cloud systems.

3.1 Techniques and Disciplines—Discussion and Limitations

To reiterate, the work in the previous section was grouped depending upon the layer of the cloud architecture it focuses upon. Some of the techniques employed may be seen across different layers but in different forms. Diversity and redundancy are two characteristics that are necessary attributes for a resilient system, albeit inherently costly. Both may be seen throughout the literature of cloud resilience in differing forms. The other techniques are autonomic, enabling dynamic adaptation to persist in service delivery.

The review in the previous section examined literature on cloud resilience for technique, architectural component applied, resilience disciplines used, and the cloud layer in which the work is situated. Figure 2 classifies the techniques used to achieve resilience in the cloud into three separate categories: redundancy, diversity, and autonomic management. Many of these techniques require no description, such as redundancy and diversity in hardware. However, the autonomic techniques may be considerably more complex and will provide for new avenues of research. For example, the

Table 9. Summary of Cloud Resilience Techniques and Their Limitations

Layer	Techniques Summary	Limitations
Physical layer and data centre	Redundant network links and data centres	Costly redundancy
	Diversity through data-centre and server placement	Not suitable, variable for majority of cloud users
	Autonomic management	Centralisation and complexity
Storage resilience	Resource replication optimisation	State-management complexity. Costly and could even be eliminated in certain use-cases
Atomic instance	Replication with some optimisation, e.g., checkpointing	State-management complexity. Costly and could even be eliminated in certain use-cases
Virtual networking	Intelligent link selection and lower-level mapping	Requires accurate information exchange, complexity
	Traffic aggregation	Reduces compartmentalisation and therefore incident isolation
	Autonomic management	Centralisation and complexity
Cloud management	Service orchestration	Complexity, multiple conflicting goals (SLAs)
	Task scheduling	Complexity, multiple conflicting goals (SLAs)
	Component monitoring	Resource-intensive
	Centralised security management	Complexity, resource-intensive,
	Instance redundancy: replication, checkpointing, etc.	Costly
Decentralised architectures	Data-driven security methods	Cryptography-heavy and resource-intensive
	Some redundancy	Questionable resource usage
	Decentralised autonomic management	Complexity
Alternative architectures	Strong diversity techniques	Experimental nature suffering from interoperability, "vendor" lock-in

management techniques to schedule and orchestrate services, and provision hardware are popular research topics although not always for resilience; while software diversity techniques also see many different methods from dynamically altering protocols in transit to altering execution path diversity. Table 9 summarises the techniques and their limitations at each layer in the cloud.

Data-centre and physical resilience techniques leverage redundancy and diversity in both network links and data-centre distribution, with survivability a notable discipline that is intuitive given the resilience upon networking at this layer. Most of these works tend to focus on resilience in inter-data-centre optical networks and most strongly in the placement and provisioning of data centres. The latest studies push towards software-defined techniques and evaluation/optimisation with upper-layer techniques. This point illustrates the weakness in attempting resilience at this layer due to the centralisation of management. Focusing resilience upon these layers is not feasible for the mast majority of cloud users/providers. Only those that manage at the data centre or optical fibre link layer can affect this resilience.

Storage resilience techniques largely rely upon fault tolerance, optimising replication to both reduce the cost and optimise the processes involved. This is likely due to the most prevalent issue occurring in storage resilience is the failure of physical mediums. These can involve low-level techniques such as erasure coding or topology/policy-based placement methods. Due to the wealth of data involved in storage, techniques involving the CIA triad are also prevalent. Arguably, data

storage is a point of weakness in environments due to its cost and high target for theft. Therefore, a better method of resilience might be to prevent its long-term storage at all, where possible. These techniques, although attempting to optimise storage usage, are still strongly resource-intensive.

Atomic instance resilience techniques, as with storage, mostly focus upon fault-tolerance techniques enabling process and system-level snapshots and replication. Again, as with storage, these techniques are very resource-intensive and highlight the weakness in statefull applications operating in these environments. On a slightly higher level, hypervisor resilience techniques can again be seen using replication techniques, although the diversity in compiler and runtimes is certainly an interesting and effective method if deployed correctly and with managed complexity.

Virtual networking techniques also tend to focus on survivability with some fault tolerance and traffic tolerance also. This is again an intuitive finding, given the network-oriented nature of these works. Overall, the techniques seen will often use some intelligence to optimise the way virtual overlay networks select links or route traffic upon the lower-level physical networks, with some resource optimisation occurring to aggregate traffic on links with similar requirements and diversity selected to maximise survivability. Overall, physical-aware virtual techniques show strong performance results. Virtual networks have been used for resiliency for a considerable time, and it could be argued that many elementary networking techniques (e.g., VLANS) are virtual abstractions of underlying resources leveraged to increase resilience. The current drive towards software-defined-networking [102] is therefore a strong contender and proponent for using virtual networking to provide resilience. However, software-defined environments have considerable complexity and are often found in centralised form, therefore must be managed to enable its resilience.

Cloud management resilience techniques have the largest body of works. These tend to focus on the management of virtual resources (mostly virtual instances) to enable a wide variety of disciplines, such as fault tolerance and robustness, but also a number of security disciplines, too. This shift in focus from the other layers (which tend to focus on one isolated discipline or cloud component) is likely due to the more holistic perspective at this layer. Component monitoring is a prominent method here, with the orchestration and scheduling of services also highly ranked. The majority of these techniques are accomplished through a novel middleware, with interoperability a crucial enabling factor. Overall, this layer provides a clear advantage to conduct resilience and is arguably essential in either monitoring or execution. Some limitations of techniques at this layer include the lack of ability to affect lower-level resilience, the huge complexity involved, and the contrasting objectives between different applications.

Decentralised cloud resilience studies are spread across a number of disciplines. Survivability and fault tolerance are present as before, but a number of security disciplines can be seen in addition to disruption and traffic tolerance. These techniques often focus on networking, such as routing or middleware. However, the data-driven nature of these environments, coupled with the inherently low security, drives data security methods. Techniques such as ABE and anomaly intrusion detection are low-resource alternatives to traditional security solutions, designed to operate in hostile environments. Redundancy is still prevalent despite the lack of resources through the use of instance checkpointing, although the efficacy of this is questionable. What is missing from these disciplines are those disciplines that provide strong network resilience due to high node churn, which may become an issue in contested and highly mobile smart environments in the future.

Alternative architectures present some variety on those previously mentioned. Diversity techniques are strongly represented, from topology-based geo-spatial techniques on consumer hardware, execution-level techniques, or simply including diversity into every aspect of the architecture. The lack of redundancy-based techniques over diversity could be argued as an attempt to move away from the costly methods that undoubtedly have a negative effect. The primary

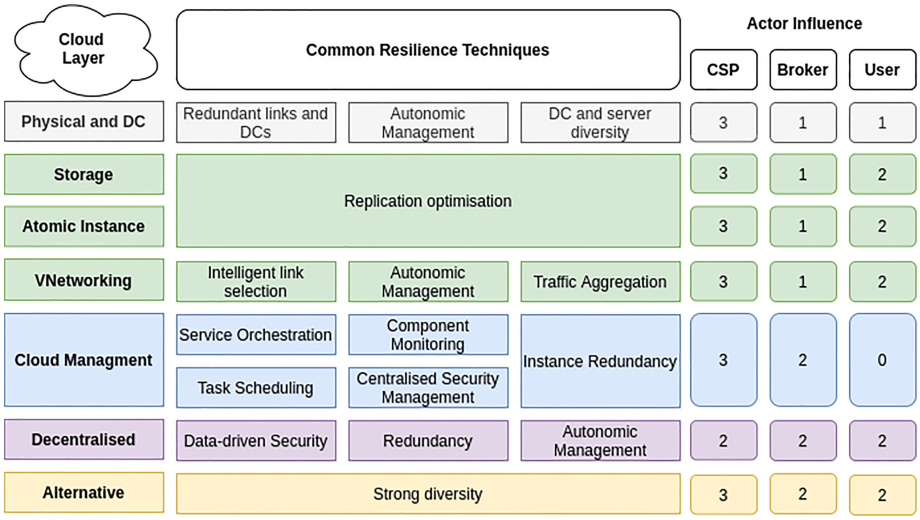


Fig. 3. Cloud resilience techniques used at each layer with corresponding actor influence upon resilience. 0=No influence, 1=Mild Influence, 2=Moderate Influence, 3=Definite Influence.

limitations of these works is that most of these architectures are experimental, yet evidence that drastically diverging from the traditional cloud architecture to provide stronger resilience yet meet the functional requirements is possible and effective.

3.2 Resilience Techniques and Actor Influence

It is also necessary to understand which actors can influence resilience at each layer. Therefore, Figure 3 illustrates the different techniques applied by actors at each layer. As the resilience techniques discovered focus mostly upon the resilience of the service as it is composed, the three most relevant actors were chosen: the CSP, the user, and also the cloud broker. The cloud auditor and cloud carrier (as defined by NIST [76]) were excluded due to their lesser influence upon service composition. The rationale for each actor's influence at each layer is defined in Table 10, ranked from 0 (no influence) to 3 (definite influence). The stronger user influence with VR can be strongly seen. Additionally, the more even distribution of influence at decentralised vs centralised cloud layers highlights a shift in responsibility.

3.3 Research Gaps

After an analysis of the previous work, a number of gaps may be seen in literature that may be addressed in future work. This work can span multiple levels but specifically concerns resilience in cloud environments that are disparate from the traditional cloud architectures.

3.3.1 Focus on specific layers. A key factor that is deemed relevant to the growing field of cloud, which is one only investigated in an isolated context, is how does the effect of resilience upon one layer affect the resilience of another layer? For example, if resilience is enabled by a user in the platform/service-oriented layers but the underlying physical layer has low resilience, to what extent is it still possible to enable increase of resilience in this manner? Such a topic is highly relevant to the way in which cloud architectures are evolving to more mobile and less deterministic

Table 10. Rationale for Actor Influence upon Cloud Resilience Techniques

Layer	CSP	Broker	User
DC and Physical	3 – Has definite control over the selection of DCs, links, and cloud management.	1 – Minimal selection capacity	1 – Minimal selection capacity
Storage and atomic instance	3 – Ultimate control over replication techniques chosen	1- Minimal technique selection	2 – Some control over data and software
VNetworking	3 – Ultimate control over traffic, links, and management	1- Minimal technique selection	2- Some control over virtual overlays
Cloud management	3 – Ultimate control over all aspects of cloud management	2 – Some capacity for orchestration, etc.	0 – No capacity
Decentralised	2 – Cloud roles and responsibilities become dynamic	2 – Some capacity to select providers	2 – Influence with providers, geospatial, and own devices
Alternative	3 – Has ultimate control over management although often autonomic	2 – Some capacity to select providers	2 – Stronger influence with user-driven models

networks and away from highly deterministic data centre environment. This has been touched on in some works such as the mapping from VNETS to physical networks [22], but there does not exist models or metrics to determine it for whole delivery platforms, i.e., a resilient PaaS on a non-resilient infrastructure. In addition, this touches on the efforts of the layered resilience model, which was the focus of the state-of-the-art survey.

In terms of physical layer resilience, the exact effect upon resilience in the cloud with different levels of diverse hardware has seen minimal work. Therefore, future research directions in this area could see the exact effect of diversification of hardware resources upon the resilience of a system be investigated. Barriers to this research mostly involve cost and time; as the necessary hardware, proprietary licenses, and practical work involved in evaluating these scenarios ensures it is difficult to implement. However, simulations may enable a realisation of evaluating this approach.

3.3.2 Constraints and Adaptive Resilience. The ability to dynamically adjust within constrained environments is another area not touched upon sufficiently. While some work within engineering has focused upon applying dynamic algorithms to graph analysis and optimisation, little work has been conducted that leverages this for the cloud. Again, this has particular relevance for mobile environments due to the constrained resources available for optimisation and the more dynamic environment. Autonomic optimisation in WSNs is not a new concept [74]. Portocarrero et al. conduct a systematic review [92] and a number of optimisation and routing techniques [35]. Autonomic self-* characteristics are employed in certain types of networking but further work can involve an evaluation and comparison of different algorithms for both traditional and mobile cloud environments. Another area that could be expanded upon is the theoretical nature of enabling resilience within the context of various constraints. This has particular relevance to cloud SLA but also to constrained models. In short, it concerns the analysis of requirements to enable the degree of resilience for the service. As resilience can be considered a scale (i.e., with state-based metrics) as opposed to a binary value, such a model could aid the construction of a service within its given constraints across all cloud service models.

3.3.3 Emerging Cloud Paradigms. Although not resilience-specific, techniques are continuously emerging that attempt to further optimise cloud processes. These are worthy of consideration, given their potential effect upon resilience, although studies are largely lacking in literature. For example, data-centre disaggregation is one such technique that involves managing cloud DCs in a resource-centric manner. This is in contrast to traditional server-centric DCs where physical

resources (e.g., compute and memory) are stored on a single server. Through disaggregation, similar resources can be physically decoupled and mounted together in same-resource blades or even racks [114]. This is envisioned to vastly optimise resource management. Pages et al. illustrate a 50% increase in virtual instance capacity in optically connected intra-DCs [89]. Not only limited to the centralised cloud, Ajibola et al. illustrate a reduction in 50% of required fog nodes for specific tasks [4]. Strong performance increases across a variety of paradigms enhances the possibility of uptake. However, few works can be seen that highlight how these techniques affect resilience. At first glance, homogenous resources spatially grouped together increases the likelihood of low availability of a specific resource and thus is not resilient. Conversely, enhanced and dynamic resource optimisation may enable dynamic fault tolerance. There is considerable variation in effects upon resilience with this new paradigm, which should therefore be studied prior to its implementation.

3.3.4 Decentralised Cloud. Arguably, the requirement for resilience at the decentralised cloud layer (i.e., close to the edge) is greater than the centralised due to data-centre hardware being resilient by nature. Such constrained environments have less ability to fall back upon redundancy and cryptographic methods to provide their resilience and generally operate in a hostile environment. They might also employ a variety of diversity-related techniques due to disparate hardware involved. The foundation of IoT networks of WSNs and MANETS have seen bodies of literature [17, 133] attempting to optimise resilient communication, security, and so on. It could therefore be argued that the entire focus of these disciplines is in delivering a resilient platform given the hostile environment in which they operate. However, the decentralised disciplines defined previously consist of more than simply IoT networks. There is now an entire ecosystem where continuously evolving use-cases demand rich data processing at any and all layers from the IoT device, to the transportation/fog layer back to the centralised cloud. These networks are heterogeneous and non-deterministic, which further complicate matters. Traffic will traverse multiple governance domains, operate on a diverse plethora of hardware/software configurations, and requirements for performance and resilience will change in fractions of a second according to external and internal requirements.

3.4 Challenges for Resilience in Cloud Computing

A final meta-analysis of the results of this survey highlights challenges for resilience in cloud computing, which we envision will drive new avenues of research. Characteristics that create challenges to cloud resilience are discussed below:

(1) **Use-case Diversity** - While cloud environments are inherently employed to provide resources for diverse use-cases, resilience techniques tend to be developed for specific use-cases. This highlighted the need for cloud environments to provide *adaptive resilience* according to the need. Integrating a plethora of techniques and selecting the most appropriate is thus an ongoing challenge for the current and emerging cloud.
(2) **Uncertain and dynamic governance and responsibility** - Traditional cloud delivery models (SaaS/PaaS/IaaS) define clear responsibility boundaries between the CSP and the user. They can assist in determining which actor can affect resilience at which layer. However, in decentralised cloud disciplines, particularly those with node mobility (e.g., MEC and fog computing), these actors can dynamically change according to physical boundaries and network requirements. Ensuring the capacity to both understand and monitor who has responsibility for resilience extemporaneously is crucial to providing resilience in decentralised clouds.
(3) **Evolving cloud paradigms** - Summarising a key concern during this survey is the manner in which cloud computing, as a concept, is continuously in flux. Driven by both

changing use-cases and continuous strives for optimisation, the deployment of new and emerging cloud paradigms poses a challenge to service resilience. The resilience of new techniques should be considered during their development and not post-deployment.

4 CONCLUSION

In conclusion, within this paper we provided a contribution to knowledge through a comprehensive review and analysis of literature that focuses upon providing resilience across the entire cloud computing consortium. The analysis was structured according to a novel methodology using specific layers within the cloud architecture to accommodate the complexity of these environments. This provided a greater insight into the techniques employed and which were lacking at each layer. We highlighted a number of gaps in literature that focused mostly on the greater need for resilience at decentralised layers and the edge. First, we note that almost no works consider the resilience of both centralised and decentralised cloud architectures in tandem. Applications for resilience that vary according to the underlying requirements and can be distributed across multiple disciplines are essential due to the increasing and wide ranging use-cases for these areas. We also note that many cloud resilience techniques rely on the costly method of redundancy. The "seemingly unlimited" resources available in centralised cloud environments will be a key driver of these techniques. However, for decentralised cloud disciplines, this is less applicable due to their resource-constrained nature. One solution is to move to stateless applications where storage redundancy is not needed. Autonomic techniques for managing decentralisation are highlighted as a strong candidate for resilience in constrained environments. Finally, understanding and considering the dynamic boundaries of responsibilities for resilience in emerging and decentralised cloud is vital.

REFERENCES

[1] N. A. S. Abdullah, N. L. Md Noor, and E. N. M. Ibrahim. 2013. Resilient organization: Modelling the capacity for resilience. In *Proceedings of the International Conference on Research and Innovation in Information Systems (ICRIIS'13)*. 319–324. DOI: https://doi.org/10.1109/ICRIIS.2013.6716729

[2] Yuan Ai, Mugen Peng, and Kecheng Zhang. 2018. Edge computing technologies for Internet of Things: A primer. *Dig. Commun. Netw.* 4, 2 (2018), 77–86. DOI: https://doi.org/10.1016/j.dcan.2017.07.001

[3] M. Aibin and K. Walkowiak. 2018. Monte Carlo tree search for cross-stratum optimization of survivable inter-data center elastic optical network. In *Proceedings of the 10th International Workshop on Resilient Networks Design and Modeling (RNDM'18)*. 1–7. DOI: https://doi.org/10.1109/RNDM.2018.8489841

[4] Opeyemi O. Ajibola, Taisir E. H. El-Gorashi, and Jaafar M. H. Elmirghani. 2019. Disaggregation for improved efficiency in fog computing era. In *Proceedings of the 21st International Conference on Transparent Optical Networks (ICTON'19)*. IEEE, 1–7.

[5] Mahmoud Al-Ayyoub, Muneera Al-Quraan, Yaser Jararweh, Elhadj Benkhelifa, and Salim Hariri. 2018. Resilient service provisioning in cloud based data centers. *Fut. Gen. Comput. Syst.* 86 (2018), 765–774. DOI: https://doi.org/10.1016/j.future.2017.07.005

[6] M. J. F. Alenazi and J. P. G. Sterbenz. 2015. Comprehensive comparison and accuracy of graph metrics in predicting network resilience. In *Proceedings of the 11th International Conference on the Design of Reliable Communication Networks (DRCN'15)*. 157–164. DOI: https://doi.org/10.1109/DRCN.2015.7149007

[7] M. J. F. Alenazi and J. P. G. Sterbenz. 2015. Evaluation and improvement of network resilience against attacks using graph spectral metrics. In *Proceedings of the Resilience Week Symposium (RWS'15)*. 1–6. DOI: https://doi.org/10.1109/RWEEK.2015.7287447

[8] S. Antony, S. Antony, A. S. A. Beegom, and M. S. Rajasree. 2012. Task scheduling algorithm with fault tolerance for cloud. In *Proceedings of the International Conference on Computing Sciences*. 180–182. DOI: https://doi.org/10.1109/ICCS.2012.71

[9] A. Aral and I. Brandic. 2018. Dependency mining for service resilience at the edge. In *Proceedings of the IEEE/ACM Symposium on Edge Computing (SEC'18)*. 228–242. DOI: https://doi.org/10.1109/SEC.2018.00024

[10] J. P. Araujo Neto, D. M. Pianto, and C. G. Ralha. 2018. An agent-based fog computing architecture for resilience on Amazon EC2 spot instances. In *Proceedings of the 7th Brazilian Conference on Intelligent Systems (BRACIS'18)*. 360–365. DOI: https://doi.org/10.1109/BRACIS.2018.00069

[11] A. C. Baktir, A. Ozgovde, and C. Ersoy. 2017. How can edge computing benefit from software-defined networking: A survey, use cases, and future directions. *IEEE Commun. Surv. Tutor.* 19, 4 (2017), 2359–2391. DOI: https://doi.org/10.1109/COMST.2017.2717482

[12] I. B. Barla, K. Hoffmann, M. Hoffmann, D. A. Schupke, and G. Carle. 2013. Shared protection in virtual networks. In *Proceedings of the IEEE International Conference on Communications Workshops (ICC'13).* 240–245. DOI: https://doi.org/10.1109/ICCW.2013.6649236

[13] I. B. Barla, D. A. Schupke, and G. Carle. 2012. Delay performance of resilient cloud services over networks. In *Proceedings of the IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA'12).* 512–517. DOI: https://doi.org/10.1109/ISPA.2012.75

[14] I. B. Barla, D. A. Schupke, M. Hoffmann, and G. Carle. 2013. Optimal design of virtual networks for resilient cloud services. In *Proceedings of the 9th International Conference on the Design of Reliable Communication Networks (DRCN'13).* 218–225.

[15] I. B. Barla Harter, D. A. Schupke, M. Hoffmann, and G. Carle. 2015. Optimal design of resilient virtual networks. *IEEE/OSA J. Opt. Commun. Netw.* 7, 2 (Feb. 2015), A218–A234. DOI: https://doi.org/10.1364/JOCN.7.00A218

[16] A. Benameur, N. S. Evans, and M. C. Elder. 2013. Cloud resiliency and security via diversified replica execution and monitoring. In *Proceedings of the 6th International Symposium on Resilient Control Systems (ISRCS'13).* 150–155. DOI: https://doi.org/10.1109/ISRCS.2013.6623768

[17] E. Benkhelifa, T. Welsh, and W. Hamouda. 2018. A critical review of practices and challenges in intrusion detection systems for IoT: Toward universal and resilient systems. *IEEE Commun. Surv. Tutor.* 20, 4 (2018), 3496–3509. DOI: https://doi.org/10.1109/COMST.2018.2844742

[18] K. E. Benson, G. Wang, N. Venkatasubramanian, and Y. Kim. 2018. Ride: A resilient IoT data exchange middleware leveraging SDN and edge cloud resources. In *Proceedings of the IEEE/ACM 3rd International Conference on Internet-of-Things Design and Implementation (IoTDI'18).* 72–83. DOI: https://doi.org/10.1109/IoTDI.2018.00017

[19] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. 2013. DepSky: Dependable and secure storage in a cloud-of-clouds. *Trans. Stor.* 9, 4, Article 12 (Nov. 2013). DOI: https://doi.org/10.1145/2535929

[20] Kashif Bilal, Osman Khalid, Aiman Erbad, and Samee U. Khan. 2018. Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. *Comput. Netw.* 130 (2018), 94–120. DOI: https://doi.org/10.1016/j.comnet.2017.10.002

[21] A. Binun, M. Bloch, S. Dolev, M. R. Kahil, B. Menuhin, R. Yagel, T. Coupaye, M. Lacoste, and A. Wailly. 2014. Self-stabilizing virtual machine hypervisor architecture for resilient cloud. In *Proceedings of the IEEE World Congress on Services (SERVICES'14).* 200–207. DOI: https://doi.org/10.1109/SERVICES.2014.44

[22] Minh Bui, B. Jaumard, and C. Develder. 2013. Anycast end-to-end resilience for cloud services over virtual optical networks. In *Proceedings of the 15th International Conference on Transparent Optical Networks (ICTON'13).* 1–7. DOI: https://doi.org/10.1109/ICTON.2013.6603032

[23] Minh Bui, Ting Wang, B. Jaumard, D. Medhi, and C. Develder. 2014. Time-varying resilient virtual network mapping for multi-location cloud data centers. In *Proceedings of the 16th International Conference on Transparent Optical Networks (ICTON'16).* 1–8. DOI: https://doi.org/10.1109/ICTON.2014.6876287

[24] Gokhan Calis and Onur Ozan Koyluoglu. 2014. Repairable block failure resilient codes. *CoRR* abs/1406.7264 (2014).

[25] John Cartlidge and Ilango Sriram. 2011. Modelling resilience in cloud-scale data centres. *CoRR* abs/1106.5457 (2011).

[26] Marco Carvalho, Dipankar Dasgupta, Michael Grimaila, and Carlos Perez. 2011. Mission resilience in cloud computing: A biologically inspired approach. In *Proceedings of the 6th International Conference on Information Warfare and Security.* 42–52.

[27] Sonali Chandna, Nabil Naas, and Hussein Mouftah. 2019. Software defined survivable optical interconnect for data centers. *Opt. Switch. Netw.* 31 (2019), 86–99. DOI: https://doi.org/10.1016/j.osn.2018.10.001

[28] Brijesh Kashyap Chejerla and Sanjay K. Madria. 2017. QoS guaranteeing robust scheduling in attack resilient cloud integrated cyber physical system. *Fut. Gen. Comput. Syst.* 75 (2017), 145–157. DOI: https://doi.org/10.1016/j.future.2017.02.034

[29] Mehdi Nazari Cheraghlou, Ahmad Khadem-Zadeh, and Majid Haghparast. 2016. A survey of fault tolerance architecture in cloud computing. *J. Netw. Comput. Applic.* 61 (2016), 81–92. DOI: https://doi.org/10.1016/j.jnca.2015.10.004

[30] C. Colman-Meixner, C. Develder, M. Tornatore, and B. Mukherjee. 2016. A survey on resiliency techniques in cloud computing infrastructures and applications. *IEEE Commun. Surv. Tutor.* 18, 3 (2016), 2244–2281. DOI: https://doi.org/10.1109/COMST.2016.2531104

[31] R. Courteaud, Yingjie Xu, and C. Cerin. 2012. Practical solutions for resilience in SlapOS. In *Proceedings of the IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom'12).* 488–495. DOI: https://doi.org/10.1109/CloudCom.2012.6427511

[32] Rodrigo S. Couto, Stefano Secci, Miguel Elias M. Campista, and Luís Henrique M. K. Costa. 2015. Server placement with shared backups for disaster-resilient clouds. *Comput. Netw.* 93 (2015), 423–434. DOI: https://doi.org/10.1016/j.comnet.2015.09.039

[33] R. S. Couto, S. Secci, M. E. M. Campista, and L. H. M. K. Costa. 2014. Latency versus survivability in geo-distributed data center design. In *Proceedings of the IEEE Global Communications Conference*. 1102–1107. DOI: https://doi.org/10.1109/GLOCOM.2014.7036956

[34] Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield. 2008. Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*. USENIX Association. Retrieved from https://www.usenix.org/conference/nsdi-08/remus-high-availability-asynchronous-virtual-machine-replication

[35] Miguel Franklin de Castro, Levi Bayde Ribeiro, and Camila Helena Souza Oliveira. 2012. An autonomic bio-inspired algorithm for wireless sensor network self-organization and efficient routing. *J. Netw. Comput. Applic.* 35, 6 (2012), 2003–2015. DOI: https://doi.org/10.1016/j.jnca.2012.07.023

[36] I. P. Egwutuoha, S. Chen, D. Levy, and B. Selic. 2012. A fault tolerance framework for high performance computing in cloud. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID'12)*. 709–710. DOI: https://doi.org/10.1109/CCGrid.2012.80

[37] S. Eisele, I. Mardari, A. Dubey, and G. Karsai. 2017. RIAPS: Resilient information architecture platform for decentralized smart systems. In *Proceedings of the IEEE 20th International Symposium on Real-time Distributed Computing (ISORC'17)*. 125–132. DOI: https://doi.org/10.1109/ISORC.2017.22

[38] S. Ferdousi, F. Dikhiyik, M. F. Habib, and B. Mukherjee. 2013. Disaster-aware data-center and content placement in cloud networks. In *Proceedings of the IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS'13)*. 1–3. DOI: https://doi.org/10.1109/ANTS.2013.6802881

[39] Marc Eduard Frîncu. 2014. Scheduling highly available applications on cloud environments. *Fut. Gen. Comput. Syst.* 32 (2014), 138–153. DOI: https://doi.org/10.1016/j.future.2012.05.017

[40] Yue Gao, S. K. Gupta, Yanzhi Wang, and M. Pedram. 2014. An energy-aware fault tolerant scheduling framework for soft error resilient cloud computing systems. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'14)*. 1–6. DOI: https://doi.org/10.7873/DATE.2014.107

[41] G. Garlick. 2011. Improving resilience with community cloud computing. In *Proceedings of the 6th International Conference on Availability, Reliability and Security (ARES'11)*. 650–655. DOI: https://doi.org/10.1109/ARES.2011.100

[42] Rahul Ghosh, Francesco Longo, Vijay K. Naik, and Kishor S. Trivedi. 2010. Quantifying resiliency of IaaS cloud. In *Proceedings of the 29th IEEE Symposium on Reliable Distributed Systems (SRDS'10)*. IEEE Computer Society, Washington, DC, 343–347. DOI: https://doi.org/10.1109/SRDS.2010.49

[43] J. L. Gonzalez, Jesus Carretero Perez, Victor J. Sosa-Sosa, Luis M. Sanchez, and Borja Bergua. 2015. SkyCDS: A resilient content delivery service based on diversified cloud storage. *Simul. Modell. Pract. Theor.* 54 (2015), 64–85. DOI: https://doi.org/10.1016/j.simpat.2015.03.006

[44] Róża Goścień and Krzysztof Walkowiak. 2017. Modeling and optimization of data center location and routing and spectrum allocation in survivable elastic optical networks.*Opt. Switch. Netw.* 23 (2017), 129–143. DOI: https://doi.org/10.1016/j.osn.2016.06.004

[45] Minzhe Guo and Prabir Bhattacharya. 2014. Diverse virtual replicas for improving intrusion tolerance in cloud. In *Proceedings of the 9th Cyber and Information Security Research Conference (CISR'14)*. ACM, New York, NY, 41–44. DOI: https://doi.org/10.1145/2602087.2602116

[46] Salim Hariri, Mohamed Eltoweissy, and Youssif Al-Nashif. 2011. BioRAC: Biologically inspired resilient autonomic cloud. In *Proceedings of the 7th Workshop on Cyber Security and Information Intelligence Research (CSIIRW'11)*. ACM, New York, NY. DOI: https://doi.org/10.1145/2179298.2179389

[47] I. B. B. Harter, M. Hoffmann, D. A. Schupke, and G. Carle. 2014. Scalable resilient virtual network design algorithms for cloud services. In *Proceedings of the 6th International Workshop on Reliable Networks Design and Modeling (RNDM'14)*. 123–130. DOI: https://doi.org/10.1109/RNDM.2014.7014941

[48] I. B. B. Harter, D. A. Schupke, M. Hoffmann, and G. Carle. 2014. Network virtualization for disaster resilience of cloud services. *IEEE Commun. Mag.* 52, 12 (Dec. 2014), 88–95. DOI: https://doi.org/10.1109/MCOM.2014.6979957

[49] T. Hecht, P. Smith, and M. Scholler. 2014. Critical services in the cloud: Understanding security and resilience risks. In *Proceedings of the 6th International Workshop on Reliable Networks Design and Modeling (RNDM'14)*. 131–137. DOI: https://doi.org/10.1109/RNDM.2014.7014942

[50] A. Hussein, I. H. Elhajj, A. Chehab, and A. Kayssi. 2017. SDN VANETs in 5G: An architecture for resilient security services. In *Proceedings of the 4th International Conference on Software Defined Systems (SDS'17)*. 67–74. DOI: https://doi.org/10.1109/SDS.2017.7939143

[51] A. Imran, A. U. Gias, R. Rahman, A. Seal, T. Rahman, F. Ishraque, and K. Sakib. 2014. Cloud-Niagara: A high availability and low overhead fault tolerance middleware for the cloud. In *Proceedings of the 16th International Conference on Computer and Information Technology*. 271–276. DOI: https://doi.org/10.1109/ICCITechn.2014.6997344

[52] Abdul Jabbar. 2010. *A Framework to Quantify Network Resilience and Survivability*. Ph.D. Dissertation. University of Kansas.

[53] V. Jaiswal, A. Sen, and A. Verma. 2014. Integrated resiliency planning in storage clouds. *IEEE Trans. Netw. Serv. Manag.* 11, 1 (Mar. 2014), 3–14. DOI: https://doi.org/10.1109/TNSM.2013.120713.120349

[54] Ravi Jhawar and Vincenzo Piuri. 2013. Fault tolerance and resilience in cloud computing environments. *Computer and Information Security Handbook*. Morgan Kaufmann, 125–141.

[55] Xiaoen Ju, Livio Soares, Kang G. Shin, Kyung Dong Ryu, and Dilma Da Silva. 2013. On fault resilience of OpenStack. In *Proceedings of the 4th Symposium on Cloud Computing (SOCC'13)*. ACM, New York, NY. DOI: https://doi.org/10.1145/2523616.2523622

[56] M. Kahla, M. Azab, and A. Mansour. 2018. Secure, resilient, and self-configuring fog architecture for untrustworthy IoT environments. In *Proceedings of the 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE'18)*. 49–54. DOI: https://doi.org/10.1109/TrustCom/BigDataSE.2018.00018

[57] M. Kanter and S. Taylor. 2013. Diversity in cloud systems through runtime and compile-time relocation. In *Proceedings of the IEEE International Conference on Technologies for Homeland Security (HST'13)*. 396–402. DOI: https://doi.org/10.1109/THS.2013.6699037

[58] A. D. Keromytis, R. Geambasu, S. Sethumadhavan, S. J. Stolfo, Junfeng Yang, A. Benameur, M. Dacier, M. Elder, D. Kienzle, and A. Stavrou. 2012. The MEERKATS cloud security architecture. In *Proceedings of the 32nd International Conference on Distributed Computing Systems Workshops (ICDCSW'12)*. 446–450. DOI: https://doi.org/10.1109/ICDCSW.2012.42

[59] A. Khalifa, M. Azab, and M. Eltoweissy. 2014. Resilient hybrid mobile ad hoc cloud over collaborating heterogeneous nodes. In *Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom'14)*. 134–143.

[60] C. Klein et al. 2014. Brownout: Building more robust cloud applications. In *Proceedings of the 36th International Conference on Software Engineering*.

[61] C. Klein et al. 2014. Improving cloud service resilience using brownout-aware load-balancing. In *Proceedings of the IEEE 33rd International Symposium on Reliable Distributed Systems (SRDS'14)*. 31–40. DOI: https://doi.org/10.1109/SRDS.2014.14

[62] J.-C. Laprie. 2005. Resilience for the scalability of dependability. In *Proceedings of the 4th IEEE International Symposium on Network Computing and Applications*. 5–6. DOI: https://doi.org/10.1109/NCA.2005.44

[63] M. Le, Z. Song, Y. Kwon, and E. Tilevich. 2017. Reliable and efficient mobile edge computing in highly dynamic and volatile environments. In *Proceedings of the 2nd International Conference on Fog and Mobile Edge Computing (FMEC'17)*. 113–120. DOI: https://doi.org/10.1109/FMEC.2017.7946417

[64] X. Li, T. Gao, L. Zhang, Y. Tang, Y. Zhang, and S. Huang. 2018. Survivable K-node (edge) content connected virtual optical network (KC-VON) embedding over elastic optical data center networks. *IEEE Access* 6 (2018), 38780–38793. DOI: https://doi.org/10.1109/ACCESS.2018.2852814

[65] Qianhui Liang and Bu-Sung Lee. 2011. Delivering high resilience in designing platform-as-a-service clouds. In *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD'11)*. 676–683. DOI: https://doi.org/10.1109/CLOUD.2011.72

[66] Hsien-Chun Liao and Chien-Fu Cheng. 2014. A malicious-resilient protocol for consistent scheduling problem in the cloud computing environment. *Comput. J.* 58, 2 (04 2014), 315–330. DOI: https://doi.org/10.1093/comjnl/bxu028

[67] Guanglei Liu and Chuanyi Ji. 2009. Scalability of network-failure resilience: Analysis using multi-layer probabilistic graphical models. *IEEE/ACM Trans. Netw.* 17, 1 (Feb. 2009), 319–331. DOI: https://doi.org/10.1109/TNET.2008.925944

[68] J. Liu and H. Shen. 2016. A low-cost multi-failure resilient replication scheme for high data availability in cloud storage. In *Proceedings of the IEEE 23rd International Conference on High Performance Computing (HiPC'16)*. 242–251. DOI: https://doi.org/10.1109/HiPC.2016.036

[69] F. Lombardi, R. Di Pietro, and C. Soriente. 2010. CReW: Cloud resilience for windows guests through monitored virtualization. In *Proceedings of the 29th IEEE Symposium on Reliable Distributed Systems*. 338–342. DOI: https://doi.org/10.1109/SRDS.2010.48

[70] Thouraya Louati, Heithem Abbes, and Christophe Cérin. 2018. LXCloudFT: Towards high availability, fault tolerant cloud system based Linux containers. *J. Parallel Distrib. Comput.* 122 (2018), 51–69. DOI: https://doi.org/10.1016/j.jpdc.2018.07.015

[71] Bing Luo and W. Liu. 2011. The sustainability and survivabiltiy network design for next generation cloud networking. In *Proceedings of the IEEE 9th International Conference on Dependable, Autonomic and Secure Computing (DASC'11)*. 555–560. DOI: https://doi.org/10.1109/DASC.2011.103

[72] P. Mach and Z. Becvar. 2017. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Commun. Surv. Tutor.* 19, 3 (2017), 1628–1656. DOI: https://doi.org/10.1109/COMST.2017.2682318

[73] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief. 2017. A survey on mobile edge computing: The communication perspective. *IEEE Commun. Surv. Tutor.* 19, 4 (2017), 2322–2358. DOI: https://doi.org/10.1109/COMST.2017.2745201

[74] David Marsh, Richard Tynan, Donal O'Kane, and Gregory M. P. O'Hare. 2004. Autonomic wireless sensor networks. *Eng. Applic. Artif. Intell.* 17, 7 (2004), 741–748. DOI: https://doi.org/10.1016/j.engappai.2004.08.038

[75] David R. Matos, Miguel L. Pardal, Georg Carle, and Miguel Correia. 2018. RockFS: Cloud-backed file system resilience to client-side attacks. In *Proceedings of the 19th International Middleware Conference (Middleware'18)*. ACM, New York, NY, 107–119. DOI: https://doi.org/10.1145/3274808.3274817

[76] Peter M. Mell and Timothy Grance. 2011. *SP 800-145. The NIST Definition of Cloud Computing*. Technical Report. National Institute of Science and Technology.

[77] Madalin Mihailescu et al. 2011. Enhancing application robustness in cloud data centers. In *Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research (CASCON'11)*. IBM Corp., 133–147.

[78] Bahareh Alami Milani and Nima Jafari Navimipour. 2016. A comprehensive review of the data replication techniques in the cloud environments: Major trends and future directions. *J. Netw. Comput. Applic.* 64 (2016), 229–238. DOI: https://doi.org/10.1016/j.jnca.2016.02.005

[79] A. Modarresi, S. Gangadhar, and J. P. G. Sterbenz. 2017. A framework for improving network resilience using SDN and fog nodes. In *Proceedings of the 9th International Workshop on Resilient Networks Design and Modeling (RNDM'17)*. 1–7. DOI: https://doi.org/10.1109/RNDM.2017.8093036

[80] A. Modarresi and J. P. G. Sterbenz. 2017. Toward resilient networks with fog computing. In *Proceedings of the 9th International Workshop on Resilient Networks Design and Modeling (RNDM'17)*. 1–7. DOI: https://doi.org/10.1109/RNDM.2017.8093032

[81] Yehia H. Khalil Mohamed. 2011. Data center resilience assessment: Storage, networking and security. PhD Thesis. University of Louisville. http://ir.library.louisville.edu/cgi/viewcontent.cgi?article=1995&context=etd.

[82] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos. 2018. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Commun. Surv. Tutor.* 20, 1 (2018), 416–464. DOI: https://doi.org/10.1109/COMST.2017.2771153

[83] Rekha Nachiappan, Bahman Javadi, Rodrigo N. Calheiros, and Kenan M. Matawie. 2017. Cloud storage reliability for big data applications: A state-of-the-art survey. *J. Netw. Comput. Applic.* 97 (2017), 35–47. DOI: https://doi.org/10.1016/j.jnca.2017.08.011

[84] W. Najjar and J.-L. Gaudiot. 1990. Network resilience: A measure of network fault tolerance. *IEEE Trans. Comput.* 39, 2 (Feb. 1990), 174–181. DOI: https://doi.org/10.1109/12.45203

[85] Toan Nguyen, J.-A. Desideri, and L. Trifan. 2012. Applications resilience on clouds. In *Proceedings of the International Conference on High Performance Computing and Simulation (HPCS'12)*. 60–66. DOI: https://doi.org/10.1109/HPCSim.2012.6266891

[86] B. Nicolae and F. Cappello. 2011. BlobCR: Efficient checkpoint-restart for HPC applications on IaaS clouds using virtual disk image snapshots. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*. 1–12. DOI: https://doi.org/10.1145/2063384.2063429

[87] Opeyemi Osanaiye, Kim-Kwang Raymond Choo, and Mqhele Dlodlo. 2016. Distributed denial of service (DDoS) resilience in cloud: Review and conceptual cloud DDoS mitigation framework. *J. Netw. Comput. Applic.* 67 (2016), 147–165. DOI: https://doi.org/10.1016/j.jnca.2016.01.001

[88] Umar Ozeer, Xavier Etchevers, Loïc Letondeur, François-Gaël Ottogalli, Gwen Salaün, and Jean-Marc Vincent. 2018. Resilience of stateful IoT applications in a dynamic fog environment. In *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous'18)*. ACM, New York, NY, 332–341. DOI: https://doi.org/10.1145/3286978.3287007

[89] Albert Pages, Rubén Serrano, Jordi Perelló, and Salvatore Spadaro. 2017. On the benefits of resource disaggregation for virtual data centre provisioning in optical data centres. *Comput. Commun.* 107 (2017), 60–74.

[90] J. Pan and J. McElhannon. 2018. Future edge cloud and edge computing for Internet of Things applications. *IEEE Internet Things J.* 5, 1 (Feb. 2018), 439–449. DOI: https://doi.org/10.1109/JIOT.2017.2767608

[91] Deepak Poola, Mohsen Amini Salehi, Kotagiri Ramamohanarao, and Rajkumar Buyya. 2017. A taxonomy and survey of fault-tolerant workflow management systems in cloud and distributed computing environments. In *Software Architecture for Big Data and the Cloud*, Ivan Mistrik, Rami Bahsoon, Nour Ali, Maritta Heisel, and Bruce Maxim (Eds.). Morgan Kaufmann, Boston, MA, 285–320. DOI: https://doi.org/10.1016/B978-0-12-805467-3.00015-6

[92] Jesús M. T. Portocarrero, Flávia C. Delicato, Paulo F. Pires, Nadia Gámez, Lidia Fuentes, David Ludovino, and Paulo Ferreira. 2014. Autonomic wireless sensor networks: A systematic literature review. *J. Sensors* 2014 (2014). https://www.hindawi.com/journals/js/2014/782789/.

[93] J. S. Preden, K. Tammemäe, A. Jantsch, M. Leier, A. Riid, and E. Calis. 2015. The benefits of self-awareness and attention in fog and mist computing. *Computer* 48, 7 (July 2015), 37–45. DOI: https://doi.org/10.1109/MC.2015.207

[94] Y. Qu and N. Xiong. 2012. RFH: A resilient, fault-tolerant and high-efficient replication algorithm for distributed cloud storage. In *Proceedings of the 41st International Conference on Parallel Processing*. 520–529. DOI: https://doi.org/10.1109/ICPP.2012.3

[95] C. Queiroz, S. K. Garg, and Z. Tari. 2013. A probabilistic model for quantifying the resilience of networked systems. *IBM J. Res. Dev.* 57, 5 (Sept. 2013), 3:1–3:9. DOI: https://doi.org/10.1147/JRD.2013.2259433

[96] H. P. Reiser and R. Kapitza. 2007. Hypervisor-based efficient proactive recovery. In *Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems (SRDS'07)*.. 83–92. DOI: https://doi.org/10.1109/SRDS.2007.25

[97] R. Rios, R. Roman, J. A. Onieva, and J. Lopez. 2017. From SMOG to fog: A security perspective. In *Proceedings of the 2nd International Conference on Fog and Mobile Edge Computing (FMEC'17)*. 56–61. DOI: https://doi.org/10.1109/FMEC.2017.7946408

[98] Rodrigo Roman, Javier Lopez, and Masahiro Mambo. 2018. Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges. *Fut. Gen. Comput. Syst.* 78 (2018), 680–698. DOI: https://doi.org/10.1016/j.future.2016.11.009

[99] V. Salapura, R. Harper, and M. Viswanathan. 2013. Resilient cloud computing. *IBM J. Res. Dev.* 57, 5 (Sept. 2013), 10:1–10:12. DOI: https://doi.org/10.1147/JRD.2013.2266972

[100] Arjuna Sathiaseelan, Mennan Selimi, Carlos Molina, Adisorn Lertsinsrubtavee, Leandro Navarro, Felix Freitag, Fernando Ramos, and Roger Baig. 2017. Towards decentralised resilient community clouds. In *Proceedings of the 2nd Workshop on Middleware for Edge Clouds & Cloudlets (MECC'17)*. ACM, New York, NY. DOI: https://doi.org/10.1145/3152360.3152363

[101] Daniel J. Scales, Mike Nelson, and Ganesh Venkitachalam. 2010. The design of a practical system for fault-tolerant virtual machines. *SIGOPS Oper. Syst. Rev.* 44, 4 (Dec. 2010), 30–39. DOI: https://doi.org/10.1145/1899928.1899932

[102] Sibylle Schaller and Dave Hood. 2017. Software defined networking architecture standardization. *Comput. Stand. Interf.* 54 (2017), 197–202. DOI: https://doi.org/10.1016/j.csi.2017.01.005 SI: Standardization SDN&NFV.

[103] M. Scholler et al. 2013. Resilient deployment of virtual network functions. In *Proceedings of the 5th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT'13)*. 208–214. DOI: https://doi.org/10.1109/ICUMT.2013.6798428

[104] M. Scholler, R. Bless, F. Pallas, J. Horneber, and P. Smith. 2013. An architectural model for deploying critical infrastructure services in the cloud. In *Proceedings of the IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom'13)*, Vol. 1. 458–466. DOI: https://doi.org/10.1109/CloudCom.2013.67

[105] S. Secci and S. Murugesan. 2014. Cloud networks: Enhancing performance and resiliency. *Computer* 47, 10 (Oct. 2014), 82–85. DOI: https://doi.org/10.1109/MC.2014.277

[106] Vibhu Saujanya Sharma and Aravindan Santharam. 2013. Implementing a resilient application architecture for state management on a PaaS cloud. In *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science (CLOUDCOM'13)*, Vol. 1. IEEE Computer Society, Washington, DC, 142–147.

[107] Noor-ul-hassan Shirazi, Steven Simpson, Simon Oechsner, Andreas Mauthe, and David Hutchison. 2015. A framework for resilience management in the cloud. *Elekt. Inf.* 132, 2 (1 Mar. 2015), 122–132. DOI: https://doi.org/10.1007/s00502-015-0290-9

[108] Bruno Sousa, Kostas Pentikousis, and Marilia Curado. 2014. MeTHODICAL: Towards the next generation of multihomed applications. *Comput. Netw.* 65 (2014), 21–40.

[109] B. Sousa, K. Pentikousis, and M. Curado. 2014. Optimizing quality of resilience in the cloud. In *Proceedings of the Global Communications Conference (GLOBECOM'14)*. 1133–1138. DOI: https://doi.org/10.1109/GLOCOM.2014.7036961

[110] R. Souza Couto, S. Secci, M. Mitre Campista, and L. M. Kosmalski Costa. 2014. Network design requirements for disaster resilience in IaaS clouds. *IEEE Commun. Mag.* 52, 10 (Oct. 2014), 52–58. DOI: https://doi.org/10.1109/MCOM.2014.6917402

[111] J. P. G. Sterbenz and P. Kulkarni. 2013. Diverse infrastructure and architecture for datacenter and cloud resilience. In *Proceedings of the 22nd International Conference on Computer Communications and Networks (ICCCN'13)*. 1–7. DOI: https://doi.org/10.1109/ICCCN.2013.6614125

[112] James P. G. Sterbenz, David Hutchison, Egemen K. Çetinkaya, Abdul Jabbar, Justin P. Rohrer, Marcus Schöller, and Paul Smith. 2010. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Comput. Netw.* 54, 8 (June 2010), 1245–1265.

[113] G. Suciu, C. Cernat, G. Todoran, V. Suciu, V. Poenaru, T. Militaru, and S. Halunga. 2012. A solution for implementing resilience in open source cloud platforms. In *Proceedings of the 9th International Conference on Communications (COMM'12)*. 335–338. DOI: https://doi.org/10.1109/ICComm.2012.6262565

[114] J. Suzuki, Y. Hidaka, J. Higuchi, Y. Hayashi, M. Kan, and T. Yoshikawa. 2016. Disaggregation and sharing of I/O devices in cloud data centers. *IEEE Trans. Comput.* 65, 10 (Oct. 2016), 3013–3026. DOI: https://doi.org/10.1109/TC.2015.2513759

[115] A. Tchana, L. Broto, and D. Hagimont. 2012. Approaches to cloud computing fault tolerance. In *Proceedings of the International Conference on Computer, Information and Telecommunication Systems (CITS'12)*. 1–6. DOI: https://doi.org/10.1109/CITS.2012.6220386

[116] M. H. C. Torres and T. Holvoet. 2014. Self-adaptive resilient service composition. In *Proceedings of the International Conference on Cloud and Autonomic Computing (ICCAC'14)*. 141–150. DOI: https://doi.org/10.1109/ICCAC.2014.33

[117] Phuoc Nguyen Tran and Nadia Boukhatem. 2008. The distance to the ideal alternative (DiA) algorithm for interface selection in heterogeneous wireless networks. In *Proceedings of the 6th ACM International Symposium on Mobility Management and Wireless Access (MobiWac'08)*. ACM, New York, NY, 61–68.

[118] Manghui Tu and Dianxiang Xu. 2013. System resilience modeling and enhancement for the cloud. In *Proceedings of the International Conference on Computing, Networking and Communications (ICNC'13)*. 1021–1025. DOI: https://doi.org/10.1109/ICCNC.2013.6504231

[119] D. Vasconcelos, V. Severino, J. Neuman, R. Andrade, and M. Maia. 2018. Bio-inspired model for data distribution in fog and mist computing. In *Proceedings of the IEEE 42nd Computer Software and Applications Conference (COMPSAC'18)*, Vol. 02. 777–782. DOI: https://doi.org/10.1109/COMPSAC.2018.10336

[120] P. Verissimo, A. Bessani, and M. Pasin. 2012. The TClouds architecture: Open and resilient cloud-of-clouds computing. In *Proceedings of the IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops (DSN-W'12)*. 1–6. DOI: https://doi.org/10.1109/DSNW.2012.6264686

[121] Alexandre Viejo and David Sánchez. 2019. Secure and privacy-preserving orchestration and delivery of fog-enabled IoT services. *Ad Hoc Netw.* 82 (2019), 113–125. DOI: https://doi.org/10.1016/j.adhoc.2018.08.002

[122] M. Villarreal-Vasquez, B. Bhargava, P. Angin, N. Ahmed, D. Goodwin, K. Brin, and J. Kobes. 2017. An MTD-based self-adaptive resilience approach for cloud systems. In *Proceedings of the IEEE 10th International Conference on Cloud Computing (CLOUD'17)*. 723–726. DOI: https://doi.org/10.1109/CLOUD.2017.101

[123] C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou. 2012. Toward secure and dependable storage services in cloud computing. *IEEE Trans. Serv. Comput.* 5, 2 (Apr. 2012), 220–232. DOI: https://doi.org/10.1109/TSC.2011.24

[124] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang. 2017. A survey on mobile edge networks: Convergence of computing, caching, and communications. *IEEE Access* 5 (2017), 6757–6779. DOI: https://doi.org/10.1109/ACCESS.2017.2685434

[125] T. Welsh and E. Benkhelifa. 2017. Perspectives on resilience in cloud computing: Review and trends. In *Proceedings of the IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA'17)*. 696–703. DOI: https://doi.org/10.1109/AICCSA.2017.221

[126] V. R. Westmark. 2004. A definition for information system survivability. In *Proceedings of the 37th Hawaii International Conference on System Sciences*. DOI: https://doi.org/10.1109/HICSS.2004.1265710

[127] Xin Xu and H. H. Huang. 2015. DualVisor: Redundant hypervisor execution for achieving hardware error resilience in datacenters. In *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'15)*. 485–494. DOI: https://doi.org/10.1109/CCGrid.2015.30

[128] J. Yanez-Sierra, A. Diaz-Perez, V. Sosa-Sosa, and J. L. Gonzalez. 2015. Towards secure and dependable cloud storage based on user-defined workflows. In *Proceedings of the IEEE 2nd International Conference on Cyber Security and Cloud Computing*. 405–410. DOI: https://doi.org/10.1109/CSCloud.2015.28

[129] J. Yao, P. Lu, and Z. Zhu. 2014. Minimizing disaster backup window for geo-distributed multi-datacenter cloud systems. In *Proceedings of the IEEE International Conference on Communications (ICC'14)*. 3631–3635. DOI: https://doi.org/10.1109/ICC.2014.6883885

[130] Q. Zhang, Q. She, Y. Zhu, X. Wang, P. Palacharla, and M. Sekiya. 2013. Survivable resource orchestration for optically interconnected data center networks. In *Proceedings of the 39th European Conference and Exhibition on Optical Communication (ECOC'13)*. 1–3. DOI: https://doi.org/10.1049/cp.2013.1291

[131] W. Zhao, P. M. Melliar-Smith, and L. E. Moser. 2010. Fault tolerance middleware for cloud computing. In *Proceedings of the IEEE 3rd International Conference on Cloud Computing*. 67–74. DOI: https://doi.org/10.1109/CLOUD.2010.26

[132] Z. Zheng, T. C. Zhou, M. R. Lyu, and I. King. 2010. FTCloud: A component ranking framework for fault-tolerant cloud applications. In *Proceedings of the IEEE 21st International Symposium on Software Reliability Engineering*. 398–407. DOI: https://doi.org/10.1109/ISSRE.2010.28

[133] Yun Zhou, Yuguang Fang, and Yanchao Zhang. 2008. Securing wireless sensor networks: A survey. *IEEE Commun. Surv. Tutor.* 10, 3 (2008), 6–28.

Received September 2019; revised February 2020; accepted March 2020

2018 Fifth International Conference on Software Defined Systems (SDS)

Embyronic Model for Highly Resilient PaaS

Thomas Welsh and Elhadj Benkhelifa
Cloud Computing and Applications Research Lab
School of Computing and Digital Technologies, Staffordshire University, Stoke on Trent, UK
Email: { thomas.welsh, e.benkhelifa}@staffs.ac.uk

***Abstract*—The resilience of distributed information systems has become a greater necessity as a support for both critical infrastructure and everyday use. This work focuses on ensuring the continued resilience of these systems through inspiration from resilient biological processes. In particular, the embryonic developmental processes are modelled and an architecture for self-healing cloud computing PaaS is developed in its likeness. This architecture is resilient through a purely distributed nature and autonomic attributes of self-healing and self-organisation.**

I. Introduction and Motivation

Cloud computing is concerned with providing information system infrastructure on a service-driven basis. Society having now transgressed from the industrial to the information age, has driven information to become a vital utility alongside those such as water, energy and sanitation. When citizens lose access to these traditional utilities the result is detrimental and even catastrophic to life. Whilst reduction of information access (e.g. to the internet) may not always have a direct effect upon physical well being, although given the mass interconnectedness of society it may certainly have an effect upon psychological well being. Further to this, information processing and communication systems now underpin the vast majority of these traditional utilities and even support critical infrastructures, whose operation is deemed essential to the stability of society. A large majority of information services now operate upon cloud environments, therefore ensuring the operational persistence of these information systems should be considered vital.

Threats to the operation of cloud and general information service systems having become more widespread, complex and varied. Traditionally one might design systems to be dependable in the face of hardware/software faults, e.g. failing components or faulty software. Also, is the continued threat of violent, varying and non-deterministic environmental conditions such as earth quakes, storms, or solar activity. There is also the threat of intentional/malicious cyber-attacks from criminal activity or military adversaries.

The constant evolution of these threats necessitates that research into their mitigation evolve alongside them. The increased complexity of systems, being composed and operating upon heterogeneous hardware/software platforms and multiple component types causes the mitigations of these threats to become more difficult . These subsets of threats tend to be examined in an atomic manner. This lack of an holistic approach can cause the solutions to these threats to become disjoint and lacking in interoperability. As a solution, this work focuses on developing systems that are *resilient* against a wide number of threats. In particular the focus is upon general faults within the system, as opposed to the threats which cause them.

Resilience in the context of computer systems and networks, is known by a number of terms, often the exact description of which differs depending on the context and who is defining it. For some it is considered synonymous with, or a measure of, fault-tolerance [1]. [2] gives two descriptions of resilience: ”the persistence of dependability when facing changes” and ”the persistence of service delivery that can justifiably be trusted, when facing changes”. Both definitions describe the persistence of dependability, although the second may be more open to interpretation. They are in fact describing a number of other desirable features of the system, as dependability contains a number of sub fields, and is also often considered a subset of trustworthiness. where they define ”changes” as any ”failures, attacks or accidents”. The author in [3] explains that this definition of resilience describes anything outside the system boundary, whereas dependability metrics often describe those within the boundary.A similar definition is given by the authors in [4], where they describe resilience as ”the ability of the network to provide and maintain an acceptable level of service in the face of various faults and challenges to normal operation”. Within this work we follow a combination of all of these definitions, stating that resilience is: ”The persistence of service delivery which can justifiably be trusted within the face of internal and external changes which challenge its nominal operation.”

Operating upon economies of scale, the infrastructures used to support cloud environments are inherently complex, dynamically linking heterogeneous components which may scale to extreme ranges. In order to minimise the threat of poor resilience within cloud and other distributed systems there is an active drive to enable them to operate autonomously. Proposed initially by IBM [5], autonomic computing is the desirable trait of *self-managing* systems which may take upon a variety of self-* states such as self-configuring, self-healing, self-optimising and self-protection.

In particular, the focus of this work is upon developing highly resilient cloud platforms which encompass autonomic attributes of self-healing and self-organisation to enable them to persist in their service delivery against these varied and evolving threats. The concept is based upon the autonomic nervous system within the animal body, which subconsciously manages a variety of heterogeneous systems which aggregate into the complex animal.

978-1-5386-5900-7/18/$31.00 ©2018 IEEE

Inspiration for the development of these highly resilient systems is taken from biological processes and their architecture. Inherently, biological systems are also complex and show considerable resilience whilst maintaining and operating at a high level of complexity through autonomic processes. Therefore, the study of biologically inspired systems seeks to model and apply the successful characteristics of these systems in order to engineer artificial ones in their likeness.

As such, the main contributions of this work is as follows:

- A high-level review of embryonic development characteristics which enable the modelling of autonomic attributes.
- A model for a cloud PaaS which is developed through a mapping of embryonic characteristics to cloud PaaS features

The rest of this paper is structured as follows: section II introduces embyronic systems and their development characteristics. Section III models necessary autonomic features of embryonic development relevant to resilience and provides an architectural model for a cloud PaaS in its likeness. Finally Section IV concludes the work.

II. Embryonic Systems and Development Characteristics

Embryonics is the field of developing systems which are based upon the characteristics of embryogensis [6]. Embryogensis refers to the development of a biological being from a mother cell, the zygote. This biological process provides a high degree of resilience through redundancy and self-healing. Embryonic inspired electronics have shown that resilience may be achieved by modelling a system on these self-healing capabilities [7], whilst embryonic software has shown similar self-healing properties for distributed systems[8]. This section will briefly discuss the basic process of embryonic development and cellular self-repair so as to derive some features to enable future modelling. This is a high-level abstraction and is by no means comprehensive.

Embryogenesis is a process of iterative *cellular-division*, whereby each cell will contain the information (DNA or genome) to create additional cells. The cells begin as stem-cells, which are *pluri-potent*, in that they can develop into any cell. Whilst all cells in an organism will share the same DNA, different cells will express different genes, which in turns causes differing proteins to be made and in effect will cause the cells to develop differently; also known as *cellular differentiation* [9] (Table I). This process enables extremely complex, multi-organ biological systems to develop from a single cell and with the process of self-check and cell-division, this organism has the ability to self-repair.

Cellular-differentiation may be instigated through a number of means which may be either *internal* or *external* to the cell. Internally the selection comes about through the presence of transcription factors, which are proteins found within the zygote. They are spatially-distributed according to the DNA and as the cells divide, those that remain in the same location denote the proteins which will be activated. This process is knows as *asymmetric segregation of protein determinants* (fig.1)[10] .

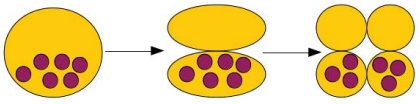


Fig. 1. Asymmetric Segregation of Protein Determinants

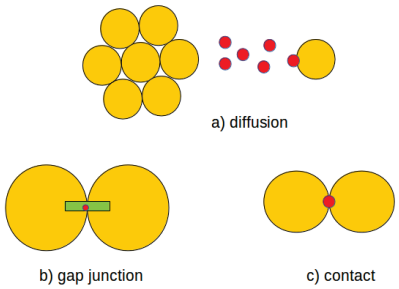


Fig. 2. Inductive Signalling

Externally, the cell can receive a prompt from other cells in a process known as *inductive signalling* or induction. This may occur in three forms: a) diffusion from a group of cells b) direct contact with another cell c) via a gap-junction between cells [10] (fig. 2). These cellular signalling methods are analysed in further detail in the next section.

Once the organism has developed, it maintains the ability to self-repair certain tissues through an application of stem-cells, also known as *somatic stem cells*. Typically, these cells, which are from an already developed animal, will be *multipotent* at best and thus only able to differentiate into a subset of the collection of possible cells. Pluri-potent stem-cells are less common although may be induced through artificial means [9]. Table 1 lists the coverage of different cell potencies.

TABLE I
Cell potency coverage

Potency	Coverage
Tutipotent	Embryonic
Pluripotent	Any
Multipotent	Multiple but similar
Oligopotent	A few
Unipotent	Self

Self-repair will occur due to cell-death, which in turn may be instigated in number of forms. Programmed Cell Death (PCD) or *apoptosis* which is cell-suicide and can occur due to an intrinsic prompt e.g stresses to the cell causing chemical changes, or damage to the DNA; or may be due to an extrinsic prompt from proteins binding externally to the cell.

An alternative method is *necrosis* which is due to an external prompt such as trauma or infection. Necrosis is considerably more traumatic to the processes and other cells within the body that apoptosis [11].

To summarise embryonic systems consist of the following key characteristics:

- **Genome** - all cells contain the same genetic material. The genes which are "switched on" denote the proteins which will be made and thus the function and form of the cell.
- **Division** - a cell may self-reproduce through division.
- **Differentiation** - cells becoming specialised to a number of different functions and through different means.
- **Self-repair** - an organism can repair through the application of stem-cells although the potency of each cell varies the type in which they can reproduce.

The aggregation of these characteristics will be the basis for the structure and architecture of the resilient cloud architecture.

A. Cellular Signalling

In addition to the development and self-repair functionality detailed previously, it is also necessary to understand the communication methods of multi-cellular systems, which is particularly relevant for development and self-healing.

Cells must have the right receptor to receive message (ligand) and thus not all cells will receive all messages. Therefore signals are only acted upon by a cell if the cell has the correct receptor and thus supports that signal type. Therefore signals are received and processed according to their purpose, as opposed to be directed towards a specific cell.

There are variety of different forms of cellular signalling. Predominately they differ in the distance and recipient of the message, although some usages and features vary. These are *autocrine* where a cell communicates with itself or same type of cell. *Paracrine* used for communication with cells within the immediate vicinity. *Endocrine* which is used for long distance/scale. Is based on hormones which provides global control such as encouraging growth and physiology. *Juxtacrine* which must be direct contact with an adjacent cell and could operate through either: gap junctions or direct contact via bind due to corresponding receptors [12]. A comparison of these signalling methods is presented in table II.

TABLE II
CELLULAR SIGNALLING METHODS

Type	**Propagation**	**Usage**	**Message Distance / Scale**
Autocrine	Self / Same	Development (reinforcement) / Pain / Infllamation / Self-destruction	Low
Juxtacrine Gap Junction	Local	Differentiation / State Info (coordination)	Very Low
Juxtacrine Contact	Local	Differentiation / Immune (safe non-safe)	Low
Paracrine	Vicinity	Differentiation / Behaviour	Diverse but Streamlined/Quick/ Degrades Rapidly/ High Concentration
Endocrine	Long Distance/Scale	Most common. Hormonal used for global control. Physiology and Growth	Slow and Long Lasting Low concentration

TABLE III
CELLULAR TO CLOUD FEATURE MAPPING

Requirement	**Represented as**	**Implementation**
Compute Resources	Cell	Network Node
Distributed Architecture	Multi-Cellular Organism	Multi-Nodes with overlay
Service Redundancy	Cell Division / Differentiation	Node/Network Replication
Service Diversity	Cell Division / Differentiation	Node/Service Replication
Network Redundancy/Diversity	Multi-Cellular Network	Graph-based resilience optimisations
Feature Selection	Differentiation	Software Libraries
Integrity Verification	Autocrine DNA Check	Cryptographic Node Hash
Self-healing	Cell Division, Self-check, Apoptsis and Necrosis	Cell SRA and Global Self-organisation
Self-organisation	Multi-Cellular Organism Cellular Signalling	P2P Overlay Network

III. HIGH-LEVEL ARCHITECTURE: PROPOSED MODEL

To accommodate the requirements given in the previous section an architecture inspired by embryonics for a resilient cloud PaaS is presented. To permit a high-level of resilience, the cloud will be purely distributed and therefore the architecture consists of only one component: the cell; which will form multi-cellular organisms in which the PaaS will operate upon. The physiology (network structure) of the architecture will self-organise and self-heal according to both external and internal stresses.

The natural resilience of molecular systems is indubitable. In order to correctly leverage these characteristics into a resilient architecture, the appropriate features must be mapped to the requirements of resilience and cloud features (table III).

The primary purpose of the multi-cellular organism is to execute applications it receives from its external environment and to pass communications to these applications via any external actors. The secondary purpose is to conduct this delivery such that the service is executed in a highly resilient manner i.e. that disruptions to the system will not prevent the application to continue to be delivered, whilst permitting the execution of features necessary for resilience.

The resilience is accomplished through: redundancy, the ability to replicate the application and other nodes, diversity by distributing the application across multiple nodes in multiple locations and through selection of resilience enabling features. Through the minimisation of architectural components, the attack surface is reduced. Subversion is discovered through

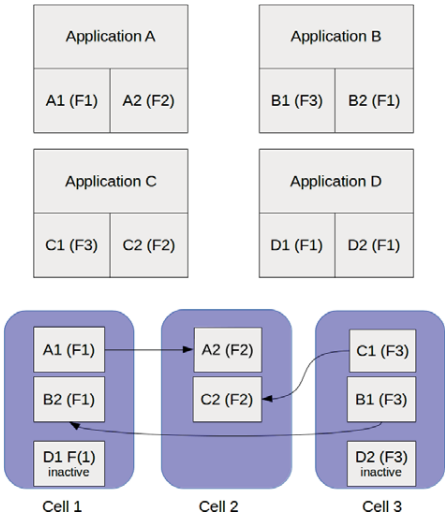


Fig. 3. Application resilience through redundancy, diversity viav cell differentiation. Applications are logically divided up according to functionality and are distributed to cells who have differentiated to that specialist functionality. Redundancy occurs via replication to additional cells which permit traffic to be re-routed according to need.

integrity self-checking, which causes it to be removed and a replacement created through division. Additionally the service resilience is enabled via dynamic self-organisation and self-healing within the network to adapt to challenges. Differentiation between individual cells decrees that each sell specialises in one or more particular functions. Therefore, logically each application will be divided into functions and distributed to cells which correspondingly specialise. Duplicates of functions will be distributed to other nodes but left inactive such that messages could be rerouted to this function in case an active node was unavailable. Cells will also have secondary specialisations in order to apply diversity and provide redundancy when needed (fig. 3).

PaaS was chosen over the other service delivery models as it more accurately aligns with the goals of resilience which are required of the cloud platform. A PaaS provides a container-based platform which enables service management, scheduling and orchestration. The application isolation may take many forms where the application may be executed at the OS level, Virtual Programming level or container level. [13].

Due to requirements for diversity across all levels within the system (including the underlying technology on which it sits), a PaaS is most suitable due to the wider ranging cross-compatibility across software and hardware. Due to the multi-platform nature of the MC architecture the platform is not constrained to just cloud architectures as its supporting infrastructure and may execute on a wide variety of devices

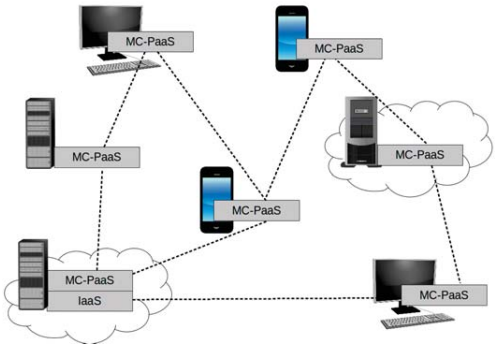


Fig. 4. Example multi-cellular PaaS distributed across diversified hardware and geographical locations.

and locations. In order to ensure maximum diversified resilience this is necessary. Figure 5 illustrates an example multi-cellular PaaS distributed across multiple diverse hardware configurations.

A. The Cell

The cell is the only architectural component (fig. 5). Cells differentiate and are networked together to provide platforms for the cloud architecture, so their computable functions vary, but their basic architecture does not.

The components of the cell are as follows:

- **Genome** - Contains the interpreter and its functionality that permits the cells code to function. Handles execution of application segments. It passes communication between the applications and the node for transmission to the organism.
- **Cell Function** - Functions are enabled through activating genes in the genome. Activated function's permit the cell to perform different activities.
- **Node** - handles the control functionality of the network. Receives, sends and forwards data to appropriate nodes, the genom and the external API.
- **Environmental Sensor** - provides interface to actors external to the system.
- **Signalling pathways** - logical communication paths between other cells.

1) Cell Functions: The cell consists of a number of concurrently/independently processing components. Algorithm 1 illustrates the inception of a cell and its communication loop.

Node - Organism Functions The node will listen from its peers for signals. If the signal received matches either an organism or current function then it is processed. If forwards messages to its vicinity according to the hop count or to everyone if they are for the organism.

Node - External Functions The node will listen for requests, and pass responses to, an external client interfacing

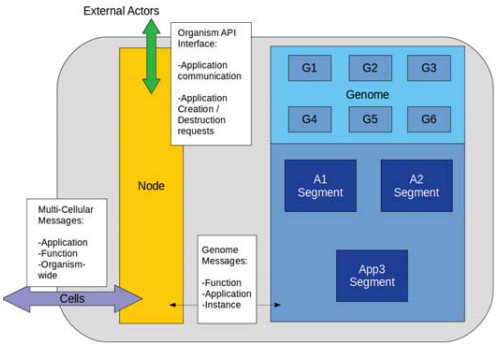


Fig. 5. Atomic cell architecture

with the organism API. These will be processed and be forwarded to other cells or to the global organism.

Inception Upon being created the cell must communicate with its neighbours. If not the first cell it will already have the address of its mother node. This node is then stored as its closest node and will use it to request further info. If it is the first cell it will divide.

Algorithm 1: Cell Inception and Communication Loop

```
Cell Division Request;
Mother Spawns Cell;
Cell Subscribes to mother;
Mother subscribes to cell;
while !Cell Death do
    if Genome Transmits Message then
        Node Publishes Message;
    end
    if Node receives message then
        if Is message global or match local hash then
            Process;
        else
            if TTL > 0 then
                Decrement TTL;
                Forward;
            else
                Drop message;
            end
        end
    end
end
```

Differentiation The differentiation process will occur at either start-up, through a prompt from the organism, or self-initiated due to spare capacity. The differentiation process will inform the cell as to what functionality it should and shouldn't have. Appropriate genes will be toggled which will enable the corresponding functionality. The cell will then respond to requests for this functionality from the organism as well as advertise when appropriate.

Application Segment Execution Application segments which fit the functionality of the genome will be requested for execution by the organism. If appropriate constraints are met then the application segment will be loaded onto the application and executed within its genome. Messages between the application segments and application control information will be passed between the node for the rest of the organism.

DNA Integrity Verification Periodically, or when prompted, the cell will perform a hash-check on its internal genome to verify its integrity. It will respond to challenges from adjacent nodes. If this check fails then the node will be cut off from the rest of the network or will self-destruct.

Reproduction Through a prompt from the organism the cell will fork its application and notify the new cell of its heritage to establish communication.

B. Multi-Cellular Organism

The multi-cellular organism is an autonomic network with self-organising and self-healing properties. It is a composition of an arbitrary and dynamic number of atomic cells with the purpose of executing numerous, scalable applications simultaneously and in a resilient manner. Therefore it only consists of one architectural component, the cell, yet maintains the resilient service delivery provision through collective decision making. Its functionality will be discussed below.

Communication As with biologically cell-based communication, cells will send message-oriented and not node-oriented communications. Such approaches have been shown to be successful in the past, as is the case with wireless sensor networks [14]. Although other cellular-signalling inspired techniques exist such as fraglet models [15], message-oriented networking has been selecting for this system due to its likeness to cellular signalling and the affinity of application to the logical division of applications and nodes.

Upon receipt of a message, each cell will assess whether it is relevant to them. If it is relevant only to their specific application instance it will be forwarded to other nodes, otherwise it will be transmitted on channels it was not received from according to remaining hop count. An exception to this is with juxtacrine messages which will be sent to only one connected neighbour. Nodes will keep a log of the "direction" and hop-count of a received message so that they may respond accordingly.

Similar to the different cellular signalling methods which vary in their distance and intensity, the differing forms of transmissions methods will vary in their propagation and hop-count; which will decree how fair the message will propagate. However the exact attributes and routing characteristics will be derived later through experimentation due to the uncertainty concerned with their stability with this particular use-case.

The proposed message categories include: Global, Function Specific, Application Specific, Application-Segment Specific or Application-Segment-Instance specific or local. Whilst the global messages will be concerned with overall platform

activities such as self-healing, resource scaling and the system diversity, the other message types will refer to specific data or applications and thus be hashed values.

Certain messages, such as with some global messages, will be transmitted but not read, until their final destination. This is similar to endocrine messages which are transmitted long distance to the next hormonal gland and then retransmitted.Other messages will merely be to nodes in the vicinity, and thus have only 1-hop count, other messages such as those intended to request or respond, will have varying and dynamic hop counts according to the exact service and request. These will be similar to paracrine messages as they will be sent to surrounding nodes.

TABLE IV

Message Type	Description	Examples	Hops	Cellular Signalling
Global	Organism wide information	Gene enable requests, State Requests, Divide	Many	Endocrine then Paracrine
Function	Specific to any node running that function	Spare function capacity request	Varied	Endocrine then Paracrine
App	To nodes running that application	Request die, Request reroute	Varied	Paracrine
App-Segment	To Nodes running that specific segment	Data, Keep alive,	Varied	Paracrine
App-Segment-Instance	To nodes running a very specific instance	Data, Keep-alive, Request reroute	Varied	Paracrine
Local	to only nodes in the vicinity	Integrity verify, divide	1	Juxtracrine

1) Application Management: **Application Request** An external actor will communicate with the organism (via any cell) and submit a request for their application to be loaded. The request will consist of the quantity and type of functions, in addition to any resource requirements such as storage etc. Upon receipt, requests for available function space will be sent. Capable nodes advertise reserved space. For all functions not with any potentially reserved space, new cells will be cloned and new functions enabled. The reserved space will need to be at least 2x the needed for redundancy purposes.

Application Creation Once the network has confirmed capacity the organism will request upload of the functions, which will then be propagated to the appropriate nodes. Once the functions have been successfully loaded, the given cells will establish links between the other nodes (fig. 6) Additional redundancy will occur through replication of functions across alternative cells which will be located according to resilience requirements.

Application Segment Communication Application segments will be identified by their application, segment and particular instance (fig. 7). Each segment will know the rough direction and hop count of the next segment for communication.

Application Destruction The application may end for a number of reasons, such as a prompt external to the organism,

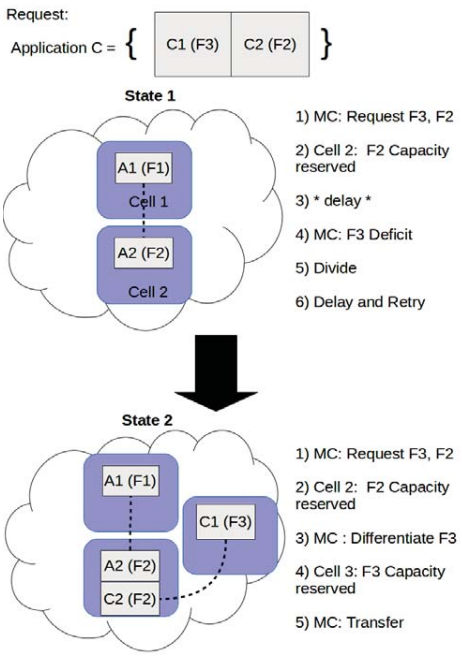


Fig. 6. Application Request Management

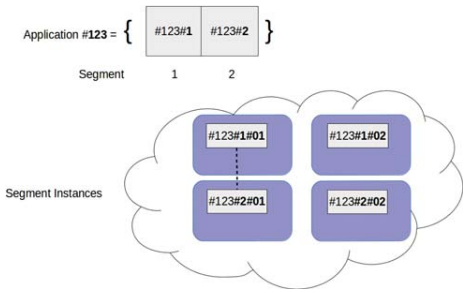


Fig. 7. Application Segment Identification

an internal prompt indicating the processing has completed, or the organism deciding that continued execution is detrimental. At this point the corresponding application will issue kill requests to its continued nodes and other replicas.

Additionally, nodes which die and contain the appropriate application will cause a break in the application path. At this point, any other application segments which cannot communicate with their corresponding segments will send a request for redundancy backups to initialise and will then re establish communication with these nodes.

Application Scaling If a greater number of requests for the application from an external source enter the network than it is possible to execute. The application will be given the command to duplicate. Likewise, if a predefined amount of time has passed and the capacity is greater than the request then the instances will be scaled down. This will be handled globally after node state indicates no activity. The overall process for application execution is given in fig. 8.

2) Self-Organisation: **Differentiation Process** As cells are created, the decision of differentiation will occur according to the current network need. Typically this will be concerned with a new or duplicate application, therefore it will likely be instigated by the dividing node, or nodes within the vicinity who are created simultaneously. Thus it will follow the paracrine or juxtacrine signal inducing methods. A node will first differentiate according to the need and then autocrine based self-check will be used until all nodes in the vicinity have confirmed that the correct type and number of nodes have been differentiated, at which point the application segments will be loaded appropriately.

Alternatively, a global request my decree that certain nodes are required to have an additional, secondary specialisation for redundancy and diversity purposes. At the targeted node types will use an autocrine based self-check until the relevant diversity/redundancy requirements are met.

Self-heal Self-healing process will largely be instigated globally, by the organism which will occur due to a lack of resources. A resource deficit will occur through the division and differentiation process as detailed previously. The resource deficit occurs mostly due to Cell death which occur for many reasons. A periodic internal hash check upon the genome which fails will cause a self-destruct and will therefore die cleanly, diving the applications and other cells time to re-organise (apoptsis).

Alternatively the cell may die in an unclean manner (necrosis) which will be harder to clean up and cause disruption to some applications. This may occur due to external reasons such as loss of the cells hardware, or it may occur due to the node misbehaving and thus being cut off from the network.

Application Distribution / Node Selection The method of application distribution across nodes will be dynamic according to the resilience requirements of redundancy and diversity. Should occur due to the differentiation process causing diversity in surrounding nodes and thus should distribute appropriately. This directly relates to service orchestration and placement.

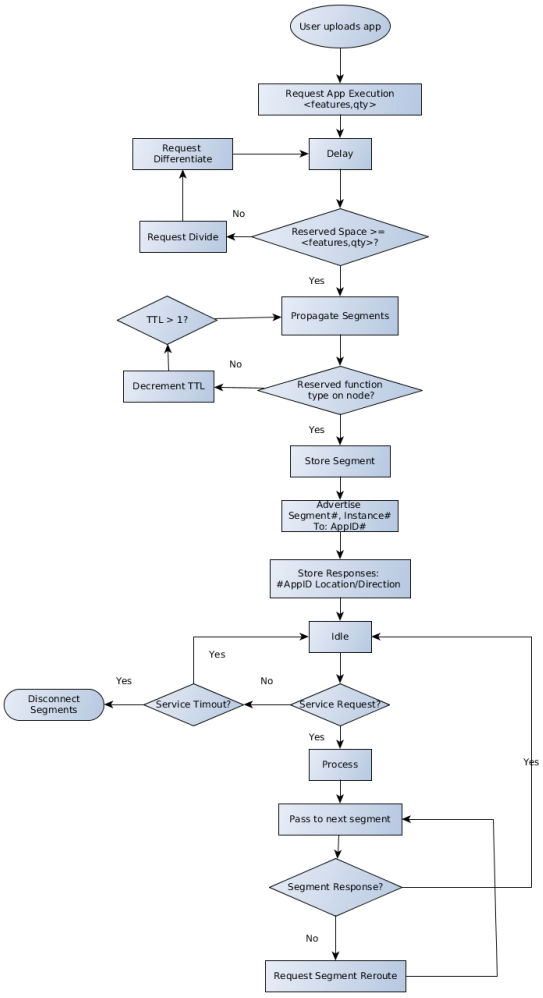


Fig. 8. Application Execution Process

3) Networking: The way in which nodes are networked will have a direct effect upon the ability of the network to scale,adapt,react, handle disruptions and its general communication performance. As mentioned previously, networking will consist of a message-oriented communication and not node/address oriented communications.

A careful selection of p2p network structures and routing methods will be necessary to select the most suitable for the required resilience characteristics. Specifically it is proposed that a hybrid communication method will likely result from the different message types as previously mapped. For example, global (organism-wide) messages which employ endocrine or endocrine-paracrine are most easily modelled as flooding or controlled flooding (multi-hop). Whereas instance specific communication between an application's segments would likely be routed. The suitable solution will be derived through investigation (variables to be investigated in table V) to understand more about the effects of these forms of message delivery upon the resilience and network requirements. One point of note of this embryonic architecture in contrast to other p2p networks is that due to the self-healing and division process, nodes will be created from other nodes and will therefore be placed by its mother node.

IV. Conclusion

This work presents a model for highly resilient cloud-computing PaaS. It is based on abstracted characteristics of embryonic development, in particular cellular differentiation and division, which enable the autonomic aspect of self-healing. Due to the architectures requirement for high-resilience, it operates in a purely distributed manner to avoid any single point-of-failure. Further research will involve investigating a variety of networking characteristics to find optimal solutions to enable the architectures ability to self-organise.

TABLE V
Networking Investigation Variables

Attribute	**Category**	**Possibilities**
Routing	Networking	Flooding, Controlled Flooding, Routed, Hybrids
Structure	Networking	Structured / Unstructured
Packet TTL	Communication	Network-dependent variable, Static, Dynamic
Message Size	Communication	Static, Dynamic
Retransmissions	Communication	None, Always, Variable
Keep-alives	Applications	Time, Distance, Traffic Dependent
Retransmissions	Applications	Requested, Responsive
Time outs	Applications	Static, Dynamic, Variable
Service Placement	Self-Organisation	Meeting Certain Metrics or Loose
Diversity Density	Self-Organisation	Forced, Loose Ratio of genome types for differentiation
Self-check period	Self-Organisation	Static, Random
Response Time outs	Self-Organisation	Static, Increasing, Network Size Dependent

References

[1] W. Najjar and J.-L. Gaudiot, "Network resilience: a measure of network fault tolerance," *Computers, IEEE Transactions on*, vol. 39, no. 2, pp. 174–181, Feb 1990.

[2] J.-C. Laprie, "Resilience for the scalability of dependability," in *Network Computing and Applications, Fourth IEEE International Symposium on*, July 2005, pp. 5–6.

[3] K. Trivedi, D. S. Kim, and R. Ghosh, "Resilience in computer systems and networks," in *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, Nov 2009, pp. 74–77.

[4] J. P. G. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Comput. Netw.*, vol. 54, no. 8, pp. 1245–1265, Jun. 2010.

[5] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[6] P. Marchal, P. Nussbaum, C. Piguet, S. Durand, D. Mange, E. Sanchez, A. Stauffer, and G. Tempesti, "Embryonics: The birth of synthetic life," in *Towards Evolvable Hardware*, E. Sanchez and M. Tomassini, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 166–196.

[7] E. Benkhelifa, A. Pipe, and A. Tiwari, "Evolvable embryonics: 2-in-1 approach to self-healing systems," *Procedia CIRP*, vol. 11, pp. 394 – 399, 2013, 2nd International Through-life Engineering Services Conference. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2212827113005027

[8] D. Miorandi, D. Lowe, and L. Yamamoto, "Embryonic models for self–healing distributed services," in *Bioinspired Models of Network, Information, and Computing Systems*, E. Altman, I. Carrera, R. El-Azouzi, E. Hart, and Y. Hayel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 152–166.

[9] V. K. Singh, A. Saini, M. Kalsan, N. Kumar, and R. Chandra, "Describing the stem cell potency: The various methods of functional assessment and in silico diagnostics," *Frontiers in Cell and Developmental Biology*, vol. 4, p. 134, 2016.

[10] D. Rudel and R. J. Sommer, "The evolution of developmental mechanisms," *Developmental Biology*, vol. 264, no. 1, pp. 15 – 37, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0012160603003531

[11] S. Y. Proskuryakov, A. G. Konoplyannikov, and V. L. Gabai, "Necrosis: a specific form of programmed cell death?" *Experimental Cell Research*, vol. 283, no. 1, pp. 1 – 16, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0014482702000277

[12] L. J. e. a. Alberts B, Johnson A, *Molecular Biology of the Cell. 4th edition*. New York: Garland Science, 2002, ch. General Principles of Cell Communication.

[13] L. Rodero-Merino, L. M. Vaquero, E. Caron, A. Muresan, and F. Desprez, "Building safe paas clouds: A survey on security in multitenant software platforms," *Computers Security*, vol. 31, no. 1, pp. 96 – 108, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404811001313

[14] F. Dressler, I. Dietrich, R. German, and B. Krger, "A rule-based system for programming self-organized sensor and actor networks," *Computer Networks*, vol. 53, no. 10, pp. 1737 – 1750, 2009, autonomic and Self-Organising Systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128608002958

[15] C. Tschudin, "Fraglets-a metabolistic execution model for communication protocols," in *Proc. 2nd Annual Symposium on Autonomous Intelligent Networks and Systems (AINS), Menlo Park, USA*, vol. 6, no. 3, 2003, pp. 1–3.

Bio-Inspired Multi-agent Embryonic Architecture for Resilient Edge Networks

Thomas Welsh and Elhadj Benkhelifa Staffordshire University
School of Computing and Digital Technologies
Stoke-on-Trent, ST4 2DE, UK
Email:{thomas.welsh | e.benkhelifa} @staffs.ac.uk

***Abstract*—With the pervasive introduction of IoE technologies, use-cases are frequently being found operating within harsh environmental conditions. This decrees the need for solutions which permit service delivery to operate in a highly-resilient manner. This work presents an architecture for a novel cloud platform designed for resilient service delivery. It supports networks where poor communication links or high node failure will cause services to be delivered in an non-resilient manner. This could be the result of factors such as high-node mobility, poor environmental conditions, unreliable infrastructure from environment disaster or cyber-attack. This biologically inspired architecture uses a purely distributed multi-agent approach to provide self-healing and self-organising properties, modelled on the characteristics of embryonic development and biological cell communication. To permit high-levels of a node churn, this multi-agent approach uses local-only communication. Probabilistic Cellular Automata are used to simulate this architecture and evaluate the efficacy of this approach.**

***Index Terms*—IoE, Embryonic, Multi-agent, Resilience, Bio-Inspired, Cloud, Edge, Artificial Life, Artificial Intelligence, Cybersecurity, Cellular Automata.**

I. Introduction

In common environments where networked devices are frequently found and relied upon: the home, work-place, public areas etc, persistent service delivery is taken for granted. Minor breaks in service may subjectively appear poor but this is largely due to their infrequence. Objectively, these networks deliver their service persistently due to the high-determinism of the environments they operate within.

We refer to the term resilience as a network's ability to persistently deliver it's service within the face of various internal and external changes [1]. If these commonly found networks are not robust enough to continue operating during changes in their internal or external environment then they do not operate resiliently.

Conversely there are many types of networks which must persist in their service delivery yet their underlying environmental conditions are non-deterministic, resulting in Intermittently Connected Networks. Examples of these include: Mobile Ad-hoc networks such as with mobile devices or vehicle communications, wireless sensor networks used for environmental monitoring often in extreme conditions and Exotic Media Networks in highly disruptive locations such as space. Most solutions rely on employing delay or disruption tolerant networking approaches such as store and forward, parallel routing, error correction etc. [2].

A number of use-cases are highlighted within the application of Industrial IoE where a large number of nodes collect data on the edge [3]. Mining [4], transportation, crime [5] and agriculture [6] are examples where large scale IoE systems may operate in environments with changing and non-determinstic environmental conditions in addition to device mobility.

Networking solutions are not compatible with all types of harsh environments. Some use-cases involve the destruction of nodes such as adversarial attacks in warfare [7] or node subversion during cyber-attack [8]. For these situations, novel architectures are often sought to enhance the network's environment. These take the form of distributed over centralised architectures, mass redundancy of resources, or diversity across hardware and software and location [9]. When seeking alternative architectures, a common source of inspiration lies within biological systems due to their inherently scalable and resilient nature. Hence this work contributes to the area of biologically inspired resilient computing architectures through the application of embyronics to the problem of IoE/ Fog operating within harsh environments. To accomplish this, a novel architecture which leverages the strong resilient characteristics of embryonic development is simulated using cellular automata in order to understand its autonomic characterstics.

The rest of this paper is as follows: Section 2 presents related work, section 3 describes the proposed embryonic platform as applied to fog networks, section 4 provides a description for the proposed Cellular Automata Model. Section 5 presents the simulation model with results. Section 6 presents an analysis of the results whilst section 7 concludes the work.

II. Related Work / Motivation

This section presents the related work for this research in the area of biologically inspired techniques for IoE/Fog resilience. Levering Fog computing nodes as a data processing platform for IoE is discussed in a number of places [10] with use-cases such as energy management [11], smart-cities [12] benefiting from simple data pre-processing on the edge nodes. However, the platform proposed in this paper aims to provide a more feature rich platform for data processing through the use of a micro-services architecture.

To the best of the authors' knowledge, the problems of resiliency within these platforms as a consequence of node mobility, hostile environments or threats to the architecture are

still open. A solution to this can be found within biological-inspired techniques, which have been applied to a whole host of problems within computer science, the Turing machine which was based upon a human clerk is arguably the first. In terms of resilient networks/systems, few works exist in the area of FoE due to it's novelty, however some can be seen within similar domains. For example cellular signalling models (which operate in a message and not node oriented fashion) have also been applied within wireless ad-hoc networks for resilient and efficient communication [13]. Whilst numerous bio-inspired techniques for resilient cloud computing and other distributed systems have been investigated [14]. These can involve architectural models similar to this work using cell-based models [15], networking approaches based upon the distributed processing capabilities of ant colonies [16], intrusion detection methods [17] or service orchestration methods [18] based upon artificial immune systems, alternatives to public key infrastructure based upon family genetics [19] and many more.

This work applies embryonics, the field of developing systems which are based upon the characteristics of embryogensis [20]. Embryogensis refers to the development of a biological being from a mother cell, the zygote. This biological process provides a high degree of resilience through redundancy and self-healing. Embryonic inspired electronics have shown that resilience may be achieved by modelling a system on these self-healing capabilities [21], whilst embryonic software has shown similar self-healing properties for distributed systems[22].

This work proposes an architecture which is deemed timely as a platform for combined Fog and IoE networks, also referred to as Fog-of-Everything (FoE) [10] [23]. Initially, cloud computing was deemed the supporting platform for IoE devices with it's inherent on-demand provisioning and elasticity for data storage, processing and UI, however a vast number of IoE use-cases are latency-sensitive and therefore long delays towards cloud providers are unsuitable. Medical applications often consist of devices with short transmission range and life-saving consequences [24][25] such as seizure detection [26]. Smart-cities provide a host of different applications which require low-latency and suffer from high device mobility, shared infrastructure/management [27] such as traffic management, public safety, [5] and emergency response. Industrial control Systems (as mentioned previously) not only require low-latency but may also operate in hostile environments[4], V2V etc. [27] [10].

This latency sensitivity has caused data processing to be pushed to the edge of the network, closer to its production. Fog computing is seen as combination of edge and cloud computing, providing the benefits of lower bandwidth and energy consumption whilst reducing greater QoS for IoE devices [27]. Fog is cited as a requirement for resilient IoE applications [28] Unfortunately fog nodes also suffer from fault-sensitivity due to their lack of scalable resources, typically wireless 1-hop device-to-fog communication and IoE device mobility [27]. To mitigate these issues, this work provides a highly resilient SaaS platform for fog networks, leveraging self-healing, self-organising and distributed processing capabilities.

III. Proposed Multi-cellular Embryonic Fog platform

This section describes the proposed multi-cellular embryonic platform and it's application to fog networks. This work extends a previous work in [29], where a conceptual model of an architecture based on the self-healing nature of embryonic cellular development was proposed as a solution to requiring a highly resilient architecture (figure 1) This architecture is purely distributed, having only a single component, the cell. The cell self-reproduces as needed which allows it to scale with demand and also self-heal. Each cell differentiates according to a particular software function and complex applications can be composed through passing messages between different cells. The cells self-organise to provide a SaaS for data processing which is considered the most suited platform for IoE applications due to the devices' constrained nature [27] [30]. This permits constrained data processing which can be offloaded to the cloud for more comprehensive processing when needed. The purpose of the architecture is to provide resilient service delivery within the face of external and internal changes, such as device mobility, loss of connection, long delays. The self-organisation architecture enables services to persist under varying conditions through software replication and self-organisation. When required functionality to complete an application is no longer available, the platform will self-organise to provide it

After further analysis of the networking characteristics of this architecture it was determined that too many communication aspects created a level of complexity which was not in line with the resilience requirements. To summarise, the ability to maintain any routing tables was incompatible with the high-levels of churn expected from environments which require high-resilience. Therefore a different, architectural approach was taken which involved local-only communication using multi-agent techniques inspired by embryonics. As a system for modelling discrete dynamical systems which operate using local-only communication, a Cellular Automata model was used to simulate the approach.

To summarise, this work investigates the application of embryonic techniques to a SaaS architecture. It aims to provide a combined Fog and IoE platform for highly resilient service delivery. Figure 2 illustrates the multicellular embryonic platform a top of the fog environment.

IV. Cellular Automata Model Simulation Description

This section describes the Cellular Automata (CA) based model for the embryonic architecture including the independent and dependent variables with motivations for their use. CA are discrete models of dynamical and complex systems. They are employed for modelling and simulating a variety of discrete scenarios. Within this simulation a stochastic CA is employed, due to having varying factors, such as the decision to spawn, the function to spawn and the simulated failure rate being dependent upon a random distribution. These stochastic cellular automata are sometimes referred to as Probabilistic Cellular Automata(PCA) and these characteristics makes them suitable for simulation of the new model[31].

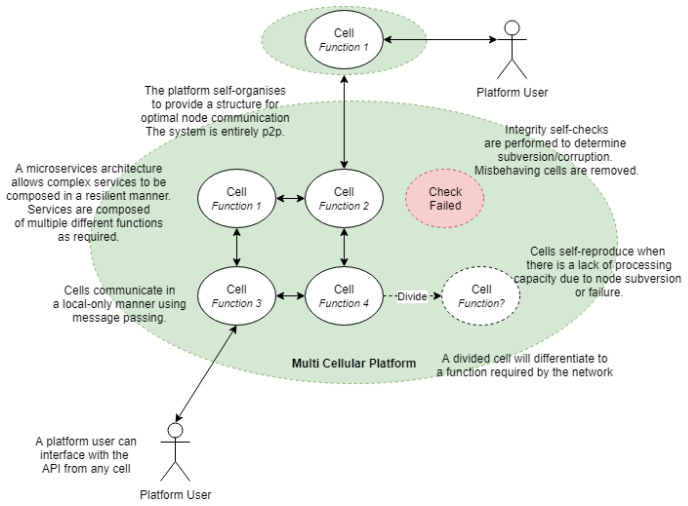


Fig. 1. The SaaS architecture inspired by animal embryonic development, composed of Multi-Cellular Platforms (MC). It provides resilient service delivery in hostile environments through leveraging the concepts of self-healing, cell division and differentiation.

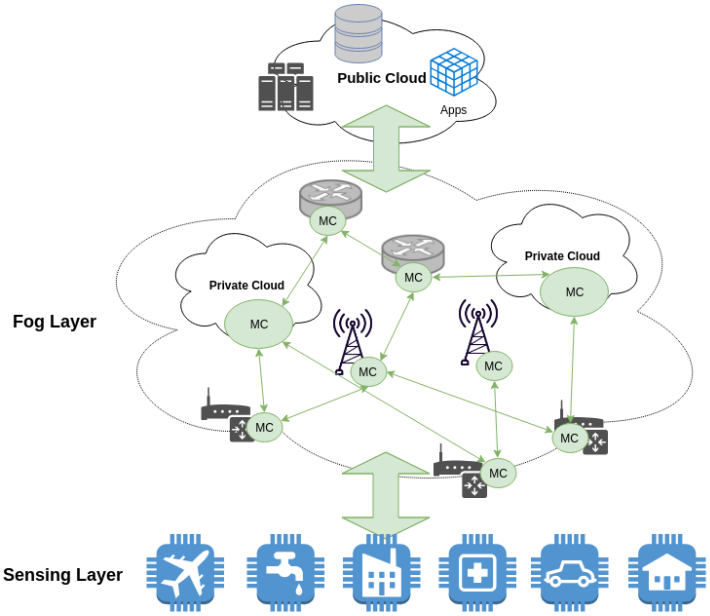


Fig. 2. The proposed Embryonic architecture situated within the Fog environment. MultiCellular platforms (MC) are distributed across the fog devices to provide data processing SaaS for the sensing layer of the SMART applications. This data is then offloaded to full cloud environments as required. The distinction between the cloud and fog layers is that the fog provides more constrained data processing. As the data is moved to the cloud, full applications can be executed in a standard IaaS environment.

A. Definitions

The following definitions should be considered within the context of this model.

- **Cell** - the node of each network and the cell in the cellular automata. Also represents the biological cell within the model. It differentiates to a particular software function and will execute code of that function. Or it will be dead (state 0)
- **Function** - A logical software function e.g. cryptography, web service
- **Application** - Composed of multiple functions to provide a specific service. An application's functions must be able to reach each other in a consecutive manner.
- **Update Function** - Probabilistic function which determines the state of each cell within the simulation. Executed upon each cell. Updated Asynchronously and randomly to account for similarities to real world networking.
- **Multi-Cellular Organism (MC)** - a collection of multiple cells which allows 1 or more applications to be executed via functions distributed across the cells.
- **Connectedness** - the quantity of fully connected applications. I.e. those where all functions can be reached consecutively in order to fully process the data.

B. Variables

The purpose of the simulation is to understand more about the constrained model's ability to resiliently deliver services. The external or internal changes to the system will be represented by the failure of nodes, according to a given stochastic variable. Whilst the resilience of the system will be measured through the number of services that are fully connected at any step within the simulation. This therefore will illustrate to what degree the network is still able to deliver its service under varying changes.

The following dependent variables (connectedness values) will be used to measure this resilience.

- **Connected0** - % of fully connected services with 0 networking hops allowed
- **Connected1** - % of fully connected services with 1 networking hop allowed

Connectedness is calculated as follows: at each step in the simulation, all starting nodes (state 1) are checked to see if they can reach the next consecutive function (state 2) and so on until the next function cannot be found, or if the final function is found creating a fully connected application. The quantity of fully connected networks is then recorded for that step in the simulation. Figure 3 illustrates two sub networks in a test network. One where the starting node can create a fully connected network and the other where it cannot.

With connected1, the search for each function is increased by one hop, as in figure 4. Neighbourhoods are von-neuman, where r=1 is a neighbourhood size of 5 inclusive of the central cell and r=2 is a neighbourhood size of 13, inclusive of the central cell. Von-neuman neighbourhoods were chosen over larger neighbourhood sizes, such as moore neighbourhoods, as it essential to reduce the number of communications being broadcast to nodes due to the flooding routing present in the network.

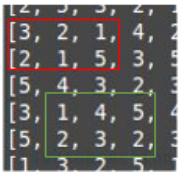


Fig. 3. The nodes in the red box are examples of a network which is not fully connected within the connected0 tests. The two starting nodes (state 1) can reach upto node 3 using local only communication but not to the final required function (state 5). In contrast the networking with a starting node highlighted in the green box can reach up to node 5.

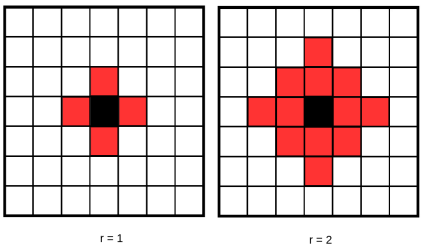


Fig. 4. Connected0 and Connected1 modelled as von-neuman neighbourhoods r=1 and r=2 respectfully. The red squares indicate the nodes in the central squares neighbourhood, the nodes which will receive communications from the central, black node.

The following application related independent variables are used within the simulation to understand more about the self-healing characteristics:

- **Spawn Rate (SR)** - % chance a node will grow another cell
- **Death Rate (DR)** - % change a node will fail
- **Quantity of Functions (Q)** - the variety of different function types
- **Neighbourhood Size (N)** - quantity of adjacent nodes to each cell.

The following independent variables are employed to understand more about the complexity characteristics of the system and their effect upon the resilience of the network.

- **Starting Location** - the starting location of the cellular architecture in the grid. e.g. central, top-left, bottom-left etc.
- **Neighbour Start Size** - the initial size of the neighbourhood.

The initial cell update function is as follows: first a random variable will decree (according to a pre-set probability of failure) if the node fails. If not the node checks its local neighbourhood. If a node is found to contain the same function as the current node, then the current node will differentiate according to an under-represented function.

TABLE I
SIMULATION PARAMETERS

Parameter	Description	Value
Time Steps	The quantity of discrete increments per test, chosen through experimentation	500
Grid Size	Discrete area of the simulation test.	10x10
Nodes	Total number of possible network nodes in each test.	100
Independent Variables	The number of variables which are varied	5
Test Runs	Number of test runs per each variable	10
Total runs	Total number of tests which were executed	27000

V. SIMULATION MODEL

This section presents the simulation of the CA-based model presented in the previous section. Including implementation characteristics, validation and results. The model was implemented in pure python using test driven development and executed on an i5 Linux system with 4GB of ram. The grid of cells was represented as a 2D array, where the state of each cell is in the range 0 to function quantity. At each step in the simulation the cells were updated with the cell update algorithm.

A. Validation

1) Graphical Validation: The simulation model was validated through a number of different yet complimentary means.

Two graphical representations were used to validate the model simulation. The independent variables related to resilience could be adjusted prior to the simulation. The interface would then permit stepping through the simulation (up to the maximum 500 time steps) The graphical representations would then enable stepping through varying configurations. This was used to examine the system operating under varying levels of strain. For example, operating under a maximum spawn rate and zero death rate allowed easily validation that the software was representing the model correctly. Conversely, using a high failure rate and low spawn rate caused the system to be completely inactive. Both of these cases illustrate validation of the model through extreme condition testing.

2) Extreme Condition Validation (Quantitative): Within the empirical experimentation dataset (presented in the next section) unstressed networks (zero failure rate) were tested. Verification occured by examining only those tests where the $DR == 0$ and $SR == 1$, with the means of each quantity of function being approximately Grid size / Function quantity. Which illustrates that in a non hostile environment where the software is aggressively reproducing the random selection algorithm evenly distributes the function types. However as there is no cell destruction this network will converge to a solution with minimal dynamism and informs very little about the selection process effect upon resilience.

3) Internal Validity: The empirical experimentation consisted of 27000 simulation test. These consisted of 10 runs of all independent variable permutations. The averages were then taken and due to the low variability and large amount of consistency (presented in the next section), these results seek to validate the simulation model.

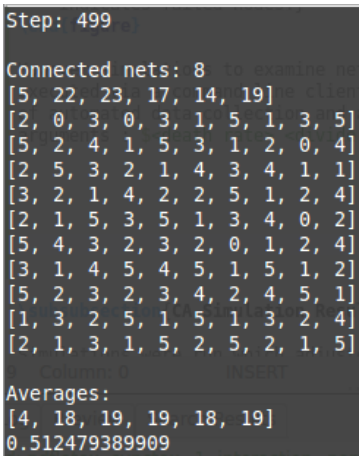


Fig. 5. ASCII Graphical interface. The grid is composed of a number of cells which can be in states 0 to the number of functions. A fully connected network is one in which the functions can be reached consecutively.

B. Results and Analysis

The purpose of the simulation was to verify that the constrained system with no real routing could remain resilient i.e. continue to deliver services under varying network changes. As the services (applications) will successfully execute when all functions of each application are fully connected, the goal is to examine the quantity of connected applications at each time step. This connectedness of each network is then compared against the independent variables (application and complexity characteristics) mentioned in the previous section.

The groups of tests examined in this section are simulated for a MC under stress, where $DR > 0$. We are seeking to determine how well the applications can still communicate using local only-interaction when under varying levels of network stress (death rate) and application complexity (function types 2-6). Variables such as spawnrate and cell differentiation process were examined in order to optimise the performance of the MC.

1) SpawnRate: During an initial analysis, it was shown that through aggressive spawning a performance improvement could be made. Conversely, reducing the spawnrate always caused negative performance. Figures 6 and 7 show the connected0 tests where $SR ==$ All and $SR == 1$, respectfully. The smoother declines indicates that this small optimisation can increase the ability for the applications to be connected, where higher failure rates benefit more over lower failure rates. This is due to the aggressive spawning increases the likelihood of nodes filling gaps where previous nodes had failed (state 0) i.e there is a greater number of functional nodes and a reduced

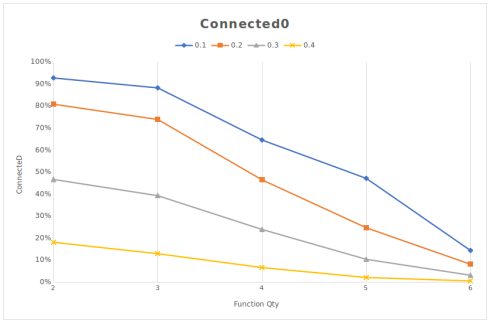


Fig. 6. Results of average connectedness of applications with 0 hop communication allowed.

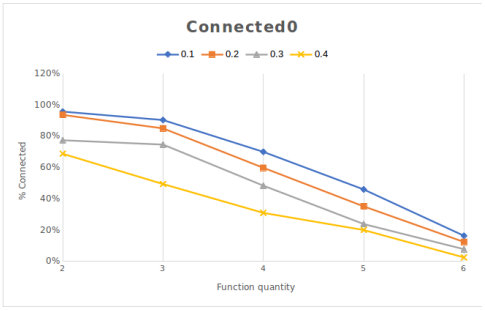


Fig. 7. Results of average connectedness of applications with 0 hop communication allowed where spawnrate==1

number of failed nodes. At lower failure rates this has less of an effect due to lower quantity of failed nodes.

2) Cell Update Function: Another method of performance optimisation made within the CA is the update function, which may choose to differentiate cells in order to optimise the MC. Therefore this section presents the results of 2 different groups of tests, each with different update functions. The update functions are as follows:

- **No Differentiation Optimisation (NoDif)** - Cells do not attempt to optimise the distribution of cells except to remove redundancy. The cell firstly examines the states of all adjacent cells. If it's state is the same as an adjacent cell and the quantity of functions is greater than the neighourhood size, it will attempt to differentiate to to an unseen state. The cell will still divide to a random function when there is adjacent space (Algorithm 1).
- **Differentiate according to weighting (Dif2)** - As above the cell first examines the adjacent node states and determines the least represented functions out of the global set of functions. It will then differentiate to the least represented function or if there are many of the same weighting, a random function from this subset(Algorithm 3).

Algorithm 1: Cell Update - No differentiation Optimisation

```
Data: NeighbourhoodState, CellState, Functions
Result: CellState, DivideCellState
Check Neighbourhood States;
if Cell is alive then
    if Cell does not die then
        if CellState is in NeighbourhoodState and N > 4 then
            CellState = random from Functions not in
              NeighbourhoodState;
        end
        if Cell should divide then
            Check Functions in NeighbourhoodState;
            for Function in Functions do
                if Function not in NeighbourhoodState then
                    DivideCellState = Function;
                else
                    DivideCellState = random from
                      Functions;
                end
            end
            Divide with DivideCellState;
        end
    else
        CellState = dead;
    end
end
```

Figure 8 presents the results of network connectedness for the connected0 tests. A clear trend can be seen that as the quantity of functions increases the ability for the applications to remain connected decreases. This is caused by the complexity of the application ensuring the likelihood of one function being adjacent to another to reduce. In the currently measured von-neumann neighbourhood where r=1, a cell will be able to communicate with 4 other cells. Therefore if $Q > 4$ the chance of finding the next node is < 100. This explains the considerable performance drop at functions 4 or greater. Regarding the difference between NoDif and Dif2, the trend illustrates dif2 providing increased connectivity over NoDif where $Q >= N$. This performance increase appears to be strong at medium DR of 0.2 and 0.3 but less favourable at 0.4. This evidences that attempting to optimise the distribution of cells does increase performance but only to a certain failure rate.

Figure 9 presents the results of network connectedness for the connected1 tests. The same trend can be seen as in connected0, however this time the decay is slower and the decline is a lot smoother, there are no sudden jumps (such as when $Q > N$). This would suggest that the issue of reduced probability is mitigated as the size of the search space is now increased. If it is assumed that the self-organisation of the MC will force differentiation of under-represented cells within the same time scale as the communication to next

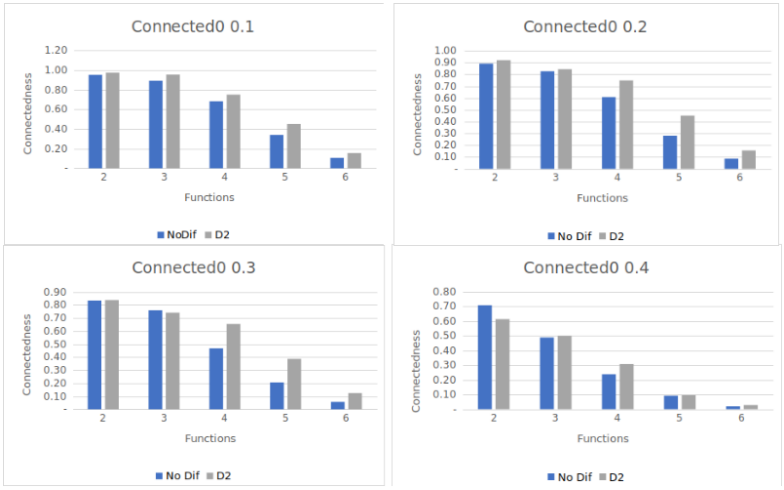


Fig. 8. Average connectedness comparing differentiation methods for connected0 tests

Algorithm 2: Cell Update - Weighted Differentiation

```
Data: NeighbourhoodState, CellState, Functions
Result: CellState, DivideCellState
Check Neighbourhood States;
if Cell is alive then
    if Cell does not die then
        for Function in Functions do
            Count Function in NeighbourhoodState;
        end
        minFunctions = Calculate min(Function in
          NeighbourhoodSate);
        CellState = random(minFunctions)
        if Cell should divide then
            Check Functions in NeighbourhoodState;
            for Function in Functions do
                if Function not in NeighbourhoodState
                  then
                    DivideCellState = Function;
                else
                    DivideCellState = random from
                      Functions;
                end
            end
            Divide with DivideCellState;
        end
    else
        CellState = dead;
    end
end
```

node, then the search space at each step will be $N * N$, else it will be $N * (N - 1)$. A considerable increase over simply N. The function distribution algorithm dif2 follows a similar performance improvement trend as with connected0 and therefore highlights the efficacy of this approach. As with the connected0 tests, high failure rates and tests where $Q > N$ still have poor performance.

C. Organism Start Size and Location

Starting location was examined for its relationship to complex systems, in case it had an effect upon the resilience of the MC. However due to a correlation of < 0.009 it was determined to have no effect.

A characteristic of the MC which is internally adjustable and is also concerned with complex systems is its starting size, i.e. the number of starting cells. Test groups were run with starting cells of 1, 5 and 9. For all tests, as start size increases, as does the connectedness. Which is an intuitive finding. At high failure rates, this can be quite a considerable performance increase of 50%, where $Q < N$.

Figures 10 and 11 show the distribution of tests which ended up with either an early failure or successfully survived, with the corresponding connectedness. In figure 10, the results for a starting MC of 1 show that a greater number of tests would fail than succeed. However figure 11 shows the same distribution but for a starting neighbourhood of 9 where only a small number of tests failed. This illustrates the effect of starting size upon success.

These graphs highlight some interesting aspects relating to the efficacy of different intelligent dynamic differentiation algorithms. At the higher start sizes the algorithms appear to be almost on par. However where start size == 1, the results don't fit a clear trend. Closer examination of these groups

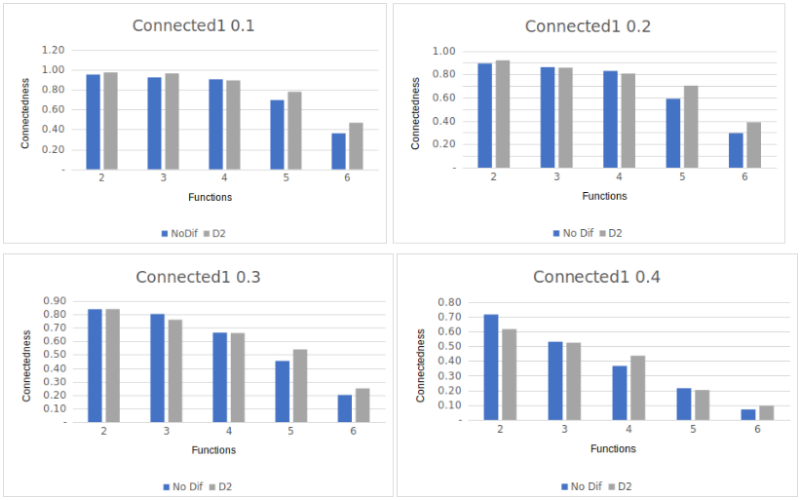


Fig. 9. Average connectedness comparing differentiation methods for connected1 tests

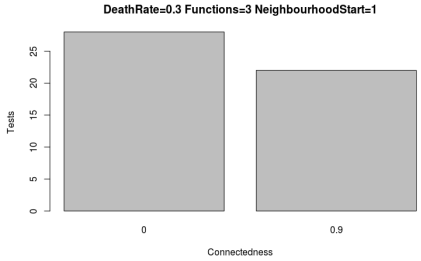


Fig. 10. 50 test runs with death rate 0.3, functions 3 and Neighbourhood startsize 1. Showing just under 50% of tests were successful whilst the rest were unsuccessful.

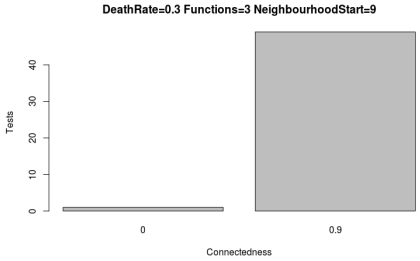


Fig. 11. 50 test runs with deathrate 0.3, functions 3 and Neighbourhood startsize 9. Where the majority of tests were highly successful.

of tests indicates something interesting. That the test results were polarised where they either tend to be highly connected, ($> 0.9\%$) or not at all which explains the poor performance where $DR = 0.3$ in Dif1. This is a trend that is persistent throughout the tests where those that did not fail had consistent connectedness where the performance improvement could be gained by increasing the start size.

Examining this at the highest failure rate of 0.4 illustrates still good performance, however this only holds true where $N > 4$. This strongly illustrates that the initial stages of an MC are vital to its ongoing survival.

VI. Analysis

This section presents an analysis of the results provided in the previous section. The results of all tests have indicated a number of points concerning factors which affect the connectedness of applications, within the embryonic model. All of which highlight methods to optimise the resilience of the MC dependent upon its characteristics. The independent factors (those variables which can be internally adjustable by the MC) are: the application complexity (quantity of functions per application), the level of division aggressiveness, starting MC size and the algorithm to determine differentiation. All of which are discussed in further detail below.

A. Application Complexity

The quantity of functions within an application undoubtedly has a direct effect upon its connectedness. This is by the inherent nature of needing to connect more cells within an application, leaving more points for failure. However, dividing application functionality is a prime characteristic of the MC architecture and a requirement of resilience, in order to distribute

risk. Therefore any technique which increases connectivity under increased application complexity is a positive benefit to the resilience of the embryonic architecture. A clear point is that adjusting the complexity of an application can enable it to survive during periods of particular stress although at an obvious reduction of functionality. This can pave the way for networking algorithms that downgrade according to perceived performance.

If if is not feasible to manage complexity through reduction of functionality, applications could be divided into sub sections or "organs" following the biological model. This would cause the overall likelihood of the application to still execute successfully albeit with a largely increased resource usage.

Whilst the application complexity would be reduced, communication protocols and network management complexity would increase due to the additional network overlay. Therefore finding an appropriate balance would be necessary.

B. Aggressive Spawning

In the previous experiments it was quickly determined that given the option, dividing to create a new cell increased the connectedness within the MC. The performance increase was considerable at higher failure rates but also decreased proportionally to application complexity. This is intuitive as a greater number of cells permits a higher chance of finding the next required function. This is directly through the required function being adjacent to the cell and indirectly through enhanced communication. However, during practical testing, the time period for spawning will need to be determined appropriately so as to not delay execution of other components with the cell, e.g. function execution.

C. Complex System Characteristics

Two characteristics of complex systems were also investigated for their effect upon connectedness, the cell's initial starting size and location. The start location showed little benefit as varying it made no difference to the connectedness results with a almost zero correlation. However the starting size correlated with an increase in connectedness, particularly at higher failure rates. The reason for this is the same as the aggressive spawning point previously mentioned. This is a strong finding considering this could give networks starting in highly hostile conditions an increased chance of survival. As it was noted that networks either survived entirely or failed early on, this highlighted the necessity for ensuring strong survival in early stages, where increased MC start size is a strong enabling factor.

D. Increasing Search Space

Throughout all the results, a clear improvement can be seen with the connected1 tests over the connected0. As explained previously, this is due to increasing the potential search space through the increased neighbourhood size, which can also be understood as von-neuman networks where r=1 and r=2, respectfully. If the quantity of functions is greater than the neighbourhood size (which is decreased further by the failure rate) then the probability of connectedness decreases and therefore increasing this search space can mitigate this issue and thus improve the connectedness of the service. However due to the application of flooding based routing, larger sized neighbourhoods will have a detrimental affect upon MC performance. Through permitting 1 hop communication, the search space can be considerably increased and thus the likelihood of remaining connected increases. Optimisations will need be determined once more is known about the effect of communication upon the MC in later stages of the research.

Therefore, we can deduce from the previous analysis that the relationship between the neighbourhood size and function quantity (service complexity) has a direct effect upon the connectedness of that application. The probabilistic model presented in section 4.1 indicates that during a best case scenario, the most optimal selection for function quantity is any which is less than the neighbourhood size. This can be explained as follows:

If $Q > N$ then the chance of the application to remaining connected is $< 100\%$, which reduces considerably with larger service sizes. Therefore increasing the neighbourhood size, and in particular ensuring it is larger than the application size including failure rate DR, such that is $N > Q$ or $N > (Q * DR)$ will cause a considerably increased chance for the application to remain connected.

E. Function spread through differentiation algorithms

Another point of optimisation is the method used by cells to differentiate according to the needed functions. The probabilistic model illustrates a best case scenario where function types are evenly distributed. However, as a result of node failures and possible differentiation methods the CA is incredibly dynamic, causing cell state to be in a constant state of flux, potentially changing at each time step. Therefore through deriving intelligent mechanisms which differentiate to force a given spread of functions, the connectedness can be improved. This can be seen with the performance improvement between the two differentiation algorithms, particularly where $Q > N$ and at higher failure rates. However this performance does not reach that of the probabilistic model although it does help to confirm its findings. This is largely due to the highly dynamic nature of the CA which also lacks many features making it less representative of the real-world system. The next stage of this research will develop a prototype implementation to permit investigation of the model with the currently lacking communication and processing performance attributes.

VII. Conclusion

In the authors' previous work [29], modelling of the embryonic characteristics against the required functionality highlighted a disparity between goals. Whilst the networking functionality decreed an array of features such as routing tables; the resilience and complexity characteristics were at odds, due to the increase in attack surface. Therefore this paper provided models and corresponding simulations to understand the efficacy of the embryonic model yet with the negative networking aspects removed: i.e. local-only communication.

This enabled an investigation into the architecture structure of the model without skewing the results with complexity of the communication.

Overall the results of the CA simulations have highlighted a number of key points relevant to the next (practical) stage of the investigation: Structural characteristics can be varied in order to increase resilience under varying levels of stress. These will provide the baseline categories for the next stages of tests. Increasing structural characteristics tend to be resource intensive in all manners, so their usefulness in a practical context will be quantified when using real-world systems. Further methods of intelligent decision making relating to optimising the function spread when choosing to differentiate can be investigated in a greater manner, as the discrete nature of the CA simulation prevented this but did highlight its effectiveness.

REFERENCES

[1] J. P. G. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Comput. Netw.*, vol. 54, no. 8, pp. 1245–1265, Jun. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2010.03.005

[2] M. J. Khabbaz, C. M. Assi, and W. F. Fawaz, "Disruption-tolerant networking: A comprehensive survey on recent developments and persisting challenges," *IEEE Communications Surveys Tutorials*, vol. 14, no. 2, pp. 607–640, Second 2012.

[3] M. Aazam, S. Zeadally, and K. A. Harras, "Deploying fog computing in industrial internet of things and industry 4.0," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4674–4682, Oct 2018.

[4] A. Singh, U. K. Singh, and D. Kumar, "Iot in mining for sensing, monitoring and prediction of underground mines roof support," in *2018 4th International Conference on Recent Advances in Information Technology (RAIT)*, March 2018, pp. 1–5.

[5] A. J. V. Neto, Z. Zhao, J. J. P. C. Rodrigues, H. B. Camboim, and T. Braun, "Fog-based crime-assistance in smart iot transportation system," *IEEE Access*, vol. 6, pp. 11 101–11 111, 2018.

[6] S. Heble, A. Kumar, K. V. V. D. Prasad, S. Samirana, P. Rajalakshmi, and U. B. Desai, "A low power iot network for smart agriculture," in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, Feb 2018, pp. 609–614.

[7] R. Amin, D. Ripplinger, D. Mehta, and B. Cheng, "Design considerations in applying disruption tolerant networking to tactical edge networks," *IEEE Communications Magazine*, vol. 53, no. 10, pp. 32–38, October 2015.

[8] I. Makhdoom, M. Abolhasan, J. Lipman, R. P. Liu, and W. Ni, "Anatomy of threats to the internet of things," *IEEE Communications Surveys Tutorials*, pp. 1–1, 2018.

[9] T. Welsh and E. Benkhelifa, "Perspectives on resilience in cloud computing: Review and trends," in *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, Oct 2017, pp. 696–703.

[10] E. Baccarelli, P. G. V. Naranjo, M. Scarpiniti, M. Shojafar, and J. H. Abawajy, "Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study," *IEEE Access*, vol. 5, pp. 9882–9910, 2017.

[11] M. A. A. Faruque and K. Vatanparvar, "Energy management-as-a-service over fog computing platform," *IEEE Internet of Things Journal*, vol. 3, no. 2, pp. 161–169, April 2016.

[12] L. Lyu, K. Nandakumar, B. Rubinstein, J. Jin, J. Bedo, and M. Palaniswami, "Ppfa: Privacy preserving fog-enabled aggregation in smart grid," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3733–3744, Aug 2018.

[13] F. Dressler and O. B. Akan, "A survey on bio-inspired networking," *Computer Networks*, vol. 54, no. 6, pp. 881 – 900, 2010, new Network Paradigms. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128610000241

[14] S. N. Mthunzi, E. BENKHELIFA, T. Bosakowski, and S. Hariri, "A bio-inspired approach to cyber security," in *Computer and Cyber Security: Principles, Algorithm, Applications, and Perspectives*, November 2018. [Online]. Available: http://eprints.staffs.ac.uk/5068/

[15] S. Hariri, M. Eltoweissy, and Y. Al-Nashif, "Biorac: Biologically inspired resilient autonomic cloud," in *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*, ser. CSIIRW '11. New York, NY, USA: ACM, 2011, pp. 80:1–80:1. [Online]. Available: http://doi.acm.org/10.1145/2179298.2179389

[16] B. Baran and R. Sosa, "A new approach for antnet routing," in *Proceedings Ninth International Conference on Computer Communications and Networks (Cat.No.00EX440)*, Oct 2000, pp. 303–308.

[17] D. Wang, L. He, Y. Xue, and Y. Dong, "Exploiting artificial immune systems to detect unknown dos attacks in real-time," in *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*, vol. 02, Oct 2012, pp. 646–650.

[18] W. Ha, "Cloud service selection with fuzzy c-means artificial immune network memory classifier," in *2018 14th International Conference on Computational Intelligence and Security (CIS)*, Nov 2018, pp. 264–268.

[19] T. Wang, B. Ye, Y. Li, and Y. Yang, "Family gene based cloud trust model," in *2010 International Conference on Educational and Network Technology*. IEEE, 2010, pp. 540–544.

[20] P. Marchal, P. Nussbaum, C. Piguet, S. Durand, D. Mange, E. Sanchez, A. Stauffer, and G. Tempesti, "Embryonics: The birth of synthetic life," in *Towards Evolvable Hardware*, E. Sanchez and M. Tomassini, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 166–196.

[21] E. Benkhelifa, A. Pipe, and A. Tiwari, "Evolvable embryonics: 2-in-1 approach to self-healing systems," *Procedia CIRP*, vol. 11, pp. 394 – 399, 2013, 2nd International Through-life Engineering Services Conference. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2212827113005027

[22] D. Miorandi, D. Lowe, and L. Yamamoto, "Embryonic models for self–healing distributed services," in *Bioinspired Models of Network, Information, and Computing Systems*, E. Altman, I. Carrera, R. El-Azouzi, E. Hart, and Y. Hayel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 152–166.

[23] K. Velasquez, D. P. Abreu, M. R. M. Assis, C. Senna, D. F. Aranha, L. F. Bittencourt, N. Laranjeiro, M. Curado, M. Vieira, E. Monteiro, and E. Madeira, "Fog orchestration for the internet of everything: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 14, Jul 2018. [Online]. Available: https://doi.org/10.1186/s13174-018-0086-3

[24] B. Farahani, F. Firouzi, V. Chang, M. Badaroglu, N. Constant, and K. Mankodiya, "Towards fog-driven iot ehealth: Promises and challenges of iot in medicine and healthcare," *Future Generation Computer Systems*, vol. 78, pp. 659–676, 2018.

[25] F. A. Kraemer, A. E. Braten, N. Tamkittikhun, and D. Palma, "Fog computing in healthcarea review and discussion," *IEEE Access*, vol. 5, pp. 9206–9222, 2017.

[26] M.-P. Hosseini, A. Hajisami, and D. Pompili, "Real-time epileptic seizure detection from eeg signals via random subspace ensemble learning," in *Autonomic Computing (ICAC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 209–218.

[27] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *Computer*, vol. 49, pp. 112–116, 2016.

[28] C. C. Byers and P. Wetterwald, "Fog computing distributing data and intelligence for resiliency and scale necessary for iot: The internet of things (ubiquity symposium)," *Ubiquity*, vol. 2015, no. November, pp. 4:1–4:12, Nov. 2015. [Online]. Available: http://doi.acm.org/10.1145/2822875

[29] T. Welsh and E. Benkhelifa", "Embryonic model for highly resilient paas," in *2018 Fifth International Conference on Software Defined Systems (SDS)*, April 2018, pp. 197–204.

[30] S. V. Vandebroek, "1.2 three pillars enabling the internet of everything: Smart everyday objects, information-centric networks, and automated real-time insights," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, Jan 2016, pp. 14–20.

[31] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan 2003.

Embyronic-Inspired Resilient Service Delivery at the Hostile Mobile Edge

Thomas Welsh
Computer Science and Information Systems
University of Limerick
Limerick, Ireland
thomas.welsh@ul.ie

Elhadj Benkhelifa
School of Computing and Digital Technologies
Staffordshire University
Stoke-on-Trent, UK
e.ebenkhelifa@Staffs.ac.uk

***Abstract*—The rapid uptake of smart systems integrating with the novel and emerging industry 4.0 ensures that Cyber Physical Systems (CPS) are being found operating in hostile environmental conditions. Conditions which may instigate considerable churn due to the failure of nodes or partial failure due to link removal. The industrial and time-critical usage of these systems ensures that operational failure can result in costly and unsafe conditions. This work presents a solution to this problem, inspired by the self-healing processes inherent to animal embryonic development, a platform for highly resilient service delivery at the hostile mobile edge is presented and evaluated.**

***Index Terms*—fog, edge, resilience, service, embryonic**

I. Introduction

The rise of rich data processing at the edge is undoubted and potentially unstoppable. Novel solutions to providing data processing services in these constrained environments Whilst the increase in use-cases integrated with Cyber Physical Systems (CPS) ensures that breaks in service could be catastrophic, with the potential for huge financial loss, or even loss of life. This safety-critical motivation and complexity of security issues has now driven a focus upon services to be delivered *resiliently*. Where resilience can be defined as encompassing a wide number of characteristics, including security, but focuses strongly on the persistence of service delivery in the face of internal and external challenges to the system. A number of use-cases are highlighted within the application of Industrial IoT where a large number of nodes collect data on the edge [1]. Mining [2], transportation, crime [3] and agriculture [4] are examples where large scale IoT systems may operate in environments with changing and non-deterministic environmental conditions in addition to device mobility.

Networking solutions are not compatible with all types of harsh environments. Some use-cases involve the destruction of nodes such as adversarial attacks in warfare [5] or node subversion during cyber-attack [6]. For these situations, novel architectures are often sought to enhance the network's environment. These take the form of distributed over centralised architectures, mass redundancy of resources, or diversity across hardware and software and location [7]. When seeking alternative architectures, a common source of inspiration lies within biological systems due to their inherently scalable and resilient nature.

A solution to this can be found within biological-inspired techniques, which have been applied to a whole host of problems within computer science, the Turing machine which was based upon a human clerk is arguably the first. In terms of resilient networks/systems, few works exist in the area of FoE due to it's novelty, however some can be seen within similar domains. For example cellular signalling models (which operate in a message and not node oriented fashion) have also been applied within wireless ad-hoc networks for resilient and efficient communication [8]. Whilst numerous bio-inspired techniques for resilient cloud computing and other distributed systems have been investigated [9]. These can involve architectural models similar to this work using cell-based models [10], networking approaches based upon the distributed processing capabilities of ant colonies [11], intrusion detection methods [12] or service orchestration methods [13] based upon artificial immune systems, alternatives to public key infrastructure based upon family genetics [14] and many more.

Levering Fog computing nodes as a data processing platform for IoE is discussed in a number of places [15] with use-cases such as energy management [16], smart-cities [17] benefiting from data pre-processing on the edge nodes. However, the platform presented in this paper aims to provide a more feature rich platform for data processing through the use of a micro-services architecture. In our previous work [18] an architectural solution to this problem was proposed. We modelled the self-healing characteristics of embyronic development to develop a self-healing, resilient data processing platform. Through simulation we illustrated the success of this approach. In this work we present and discuss a preliminary implementation of this work.

The rest of this paper is as follows: Section 2 describes the embryonic platform architecture, characteristics, and application processing practicalities. Section 3 discusses the implementation specifics and the methodology for analysing application and resilience performance. Section IV presents and discusses results of tests where there were no failures induced. whilst section V discusses the performance of the network under stress with induced failures. Finally, section VI concludes this paper.

II. Multi-cellular Embryonic Platform

This section presents the conceptual architecture and design of the Multi-Cellular Embyronic Platform (MC) which is a multi-agent complex adaptive system. This architecture is the practical realisation of the Cellular Automata-based model which was simulated in our previous work [18] illustrating the efficacy of the approach. The goal of the system is to provide resilient service delivery upon edge nodes in a microservices-like fashion. To achieve this, the system takes inspiration from embryogensis. Embryogenesis inspired electronics (or *embryonics*) have shown that resilience may be achieved by modelling a system on these self-healing capabilities [19], whilst embryonic software has shown similar self-healing properties for distributed systems [20] and is therefore proven in it's application for the management of complexity for resilience. In this application, the concepts of *cellular differentiation* and *cellular division* of animal embryonic development are employed to provide self-healing functionality in a multi-agent system. The cells (which compose the system) employ the MAPE-K loop to permit self-organising and adaptive behaviour according to internal and external events with an overall goal of enabling the execution of services in the presence of partial or full node failures.

The MC architecture is purely P2P so that it suffers from no signal point of failure which adheres to the characteristics of a complex system and resilience. Therefore it consists of only one component - the *cell*. In the likeness of embyronic characteristics an arbitrary quantity of cells will autonomously *divide* and *differentiate* according to both intrinsic information incepted about the ideal environmental state and the current state according to information from neighbourhood cells. For these reasons, this architecture is said to be *self-reproducing*, *self-healing* and *self-organising*. Additionally the local-only communication (NF2 and NF4) ensures the properties of this system are *emergent*.

A. The Cell

The Cell has two primary purposes. Firstly it should collaboratively maintain the P2P overlay network with it's neighbours. This involves dividing (self-reproducing) as required in order to self-heal the network and also differentiate to a function type which is underrepresented locally. It accomplishes this through continuously gossiping to it's neighbours about its known view of the network so that all cells have an accurate local view and can differentiate accordingly.

The second purpose is to process data which it is intended to. The chosen software function, which the cell has differentiated to, denotes the messages it can process. Messages it receives which contain its specific software function will be processed and the result forwarded on to it's neighbours. An application is therefore executed through this process of iterative data processing by individual cells.

The cell consists of the following sub-components which are illustrated in figure 1. The description of each component can be found below:

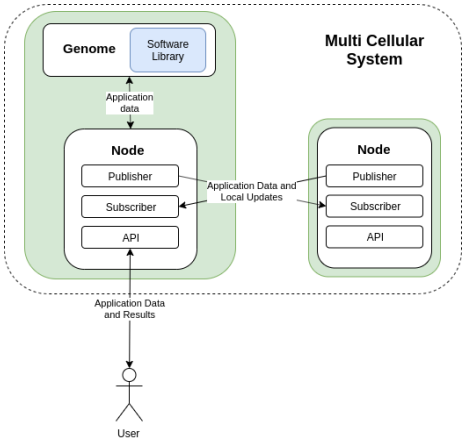


Fig. 1. Architectural model of the cell components

- **Genome** - will differentiate to a particular software function, as required by the global network. It will only process data of its function type, ignoring other messages. Once the data has been processed it passes the output to the node, for distribution to other cells or the end user.
- **Node** - uses the publisher/subscribe communication pattern to pass messages between other cells. There are two types of messages, data messages to be processed by the corresponding function and organisational messages. Organisational messages involve broadcasting known local node addresses, the current function types the cell can see, and requests for differentiation.
- **API** - is a sub-component oif the node and exposed to the end-user/devices. It is used to push data to be processed onto the MC platform and also to return results.

The cell is firstly initialised as follows:

1) Thresholds and initial values are set, these include the range of possible functions, the number of times the cell should divide and numerous timeout thresholds to control performance.
2) Next the cell instructs the genome to differentiate according to a random function within the range allowed.
3) The network socket in which messages will be published from is started.
4) A subscriber execution thread is started which loops a pool of sockets listening for messages from neighbours and then processes them as appropriate
5) An API loop is initiated which polls for messages from end-users.
6) The cell will divide according to the number of times given to it by it's mother cell.
7) The cell subscribes to it's mother cell and begins polling for messages.

After initialisation the cell enters a main loop in a further execution thread. The following activities are continuously looped within the thread:

1) Keep alives are published to all known subscribers. These consist of the cell's current known state of the neighbourhood. I.e. the set of all cell values it can see and communicate with.
2) The cell publishes lists of known peers so that it's subscriber cells can
3) Last messages from all subscribed to nodes are checked against timeout threshold. Those which exceed are removed from the pool. If the cell removed was a child cell it will divide.
4) The cell will check the current state of its neighbourhood and differentiate to an underrepresented function in order to ensure function distribution is even.

B. Self-organisation through optimal differentiation

In our previous work [18], it was shown that optimisation the differentiation could have a beneficial effect upon application connectivity compared to a random baseline. The algorithm used in these tests is only one of many examples and was chosen after successful evaluation

Algorithm 1: Cell Update - Weighted Differentiation

```
Data: NeighbourhoodState, CellState, Functions
Result: CellState, DivideCellState
Check Neighbourhood States;
if Cell is alive then
    if Cell does not die then
        for Function in Functions do
            Count Function in NeighbourhoodState;
        end
        minFunctions = Calculate min(Function in
          NeighbourhoodState);
        CellState = random(minFunctions)
    else
end
```

C. Applications

Applications are composed of a stack of distinct software functions and must be designed for the platform but operate using traditional vitual machines (e.g. Python Interpreter). The application data resides in the passing of messages and therefore no crucial data persists on any cell, ensuring that if either a full or partial failure occurs then the execution of the application can persist. At each stage of the application, the current function and data to be processed is propagated to neighbourhoods of cells, if a particular cell has differentiated to that particular function module then it will pop that function from the application stack, process the data with the appropriate function, and then propagate the resulting data with the rest of the stack. Full execution will occur in micro-services like fashion as messages are passed to loosely-coupled services.

This form of iterative data processing is designed to suit edge environments which were shown to need rich data processing functionality in a SaaS and PaaS manner yet are currently too constrained to do so due to their constrained nature [21] [22]. Therefore this method distributes the data processing in a resilient manner across multiple nodes.

D. Communication and Message Registry

Messaging is a crucial component of the system. Nodes broadcast messages as this operates without the requirement for maintaining routing tables, a costly procedure for both time and computation in networks with high degrees of node churn. In order to minimise network contention, cells limit their neighbourhood size (messages are not forwarded) and therefore the size of their neighbourhood directly affects network contention. Cells will broadcast two types of messages, firstly they *gossip* about their local environment, which permits the self-organisation of the system, secondly they broadcast the result of any processed data to their peers for further processing. The majority of functionality occurs within the single keep alive message - OKA. This employs the concept of local communication where nodes will continuously gossip to their neighbours about their known information. Table I presents the finalised messages implemented in this proof-of-concept.

TABLE I
REGISTER OF MESSAGES

Type	Code	Function
Organism	OKA	Keep Alive –– also sends known state of local nodes
Organism	OPAR	Advertise all known peers
Function	FDPR	Application function traffic in
Function	FAOT	Application function traffic out
API	PAPP	Push application to MC from an external user

III. Implementation and Evaluation

Following from the description of the architecture in the previous section, this section presents the implementation details and the performance valuation of the architecture. The code is heavily supported by the ZeroMQ (zero message queue) library [23] which is an extremely fast and efficient high speed, brokerless message passing library written in C. The publish/subscribe communication model employed permits the self-healing, apoptosis model of cellular death whereby nodes will cease communication with misbehaving nodes through simply unsubscribing.

The MC architecture can be modelled as a Finite State Machine (FSM), figure 3 illustrates these distinct states. Where the first cell will spawn with the ideal conditions incepted into it. The network will then converge to its ideal network structure by dividing according to its given environment. As cells die it will divide to create more and therefore cycle between degraded and acceptable delivery.

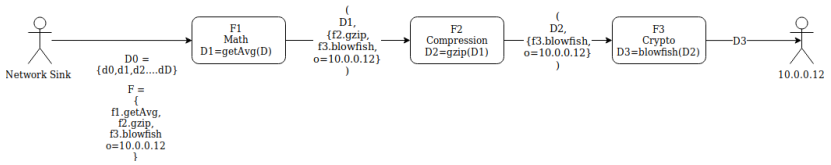


Fig. 2. This diagram illustrates the constrained application execution. The application starts as a tuple of data (D) and functionality F). The data is passed to the top function of the list which is removed as the data is passed to the next cell. The current state of the application therefore resides in the message.

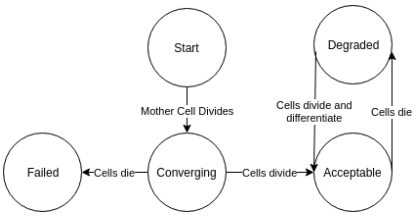


Fig. 3. The full MC architecture modelled as a Finite State Machine.

A. Methodology

All tests were run on a 64bit i7 quad core Laptop running Debian Linux where all cells used localhost addresses and randomised port numbers. All communication was conducted over the operating system's internal network stack and therefore was minimally affected by any external conditions. Additionally, all activity on the computer including background services was kept to the bare minimum to provide fair and consistent test results.

Tests were autonomously executed in batches using a suite of scripts to provide empirical consistency. Messages both sent and received by all cells were pushed to the debug server as per the setup in the previous chapter where they were stored in a MYSQL database. The mother cell was spawned with the configuration options for that test. A loop then iterates every second up until the time limit of 5 minutes (300 seconds). The events which occur during the loops typically consists of pushing applications to be processed and instigating artificial faults by killing nodes according to a pseudorandom threshold. Post-test run, all of the messages would be retrieved from the back end database and parsed to create a timeline of events. When a change in the network structure occurs, a graph of the network is constructed using the networkx python library. A history of all graphs is then stored (with a corresponding PNG image) for further analysis. This consists of a number of different algorithms which examine the resilience of the network at each stage. Statistics about the application processing time and messages sent and received are also recorded. Note: that graphs at this stage, particularly for those tests with node failures, were pruned of isolates. Therefore the graphs do not show all nodes that may be active, only those that are connected to other nodes. This is to ensure more accurate reporting of performance and resilience statistics.

B. Variables

1) Dependent Variables: These metrics are employed to evaluate the performance of application processing upon the architecture. They seek to determine whether the embryonic architecture can successfully process the application in it's current state. For the purposes of this testing, the applications used are only representative of what is considered to be a real-world data processing task discussed previously. This is accomplished as it permits the timeline of activity within each test to be examined easily from an external viewpoint. Determining if the application has been processed correctly requires the same logic as processing it upon the collection of cells. Moreover a simplistic data processing application still permits investigation and evaluation of the success of the platform as the focus is in connecting the distinct function types together to form a complete application. Therefore the example services consist of a simplistic function which merely multiplies the data it receives by it's function number and then pushes it to the next function until the data cannot be processed anymore due to all functions being exhausted. The application values are generated according to a maximum value and cells will differentiate according to their local information driven self-organisation as detailed in Algorithm 1. Table II presents an example data processing application for 5 functions and input data of 2. As the data is received by each function it is multiplied by that value and then pushed to the next. Although simplistic, this method is chosen as it minimises any effects upon the system performance that may result from more complex applications. Additionally the ability for software environments to execute arbitrary code is not in question and therefore does not need to be evaluated. This test focuses upon the networking architecture and self-organisation which permits the passing of messages between arbitrary software functions.

Applications are pushed to the architecture from the API every 10 seconds to allow time for the other messages to clear for more accurate reporting. The data is incremented each time to ensure different results to allow for ease of measurement. As the timeline is generated from processing the messages after the experiment has concluded, performance related values are recorded next to each application which are presented in table

2) Independent Variables: In order to evaluate the previously defined dependent variables whilst the system is under

TABLE II
EXAMPLE DATA PROCESSING APPLICATION MESSAGES FOR PERFORMANCE TESTING

Parameter	Data
Functions	5
Data	2
Stage 0	[[1,2,3,4,5], 2]
Stage 1	[[1,2,3,4], 10]
Stage 2	[[1,2,3], 40]
Stage 3	[[1,2], 120]
Stage 4	[[1],240]
Stage 5	[240]

TABLE III
VARIABLES RECORDED PER EACH TEST TO RECORD NETWORK PERFORMANCE

Variable	Description
Application Data (*Data*)	The output of the processed data.
Total Process Time (*TotalProc*)	The time in seconds from the first message to the last.
Shortest Process Time (*ShortProc*)	The time in seconds from the first message to the first output message.
Longest Receive Time (*LongRecv*)	The time in seconds from the first message to the last received message.
Process Messages (*Proc*)	The total number of data processing messages
Output Messages (*Out*)	The total number of data result messages.
Receive Messages (*Recv*)	The total number of messages received by nodes.

different states a number of parameters were varied (table IV). These are related to those which were previously investigated within chapter 4 using CA although variations exist due to the continuous nature of these tests contrasting with the discrete nature of the CA simulations.

TABLE IV
INDEPENDENT VARIABLES CHOSEN FOR EXPERIMENTATION

Variable	Description	Range
Functions ($Func$)	The quantity of possible function types.	2 - 7
Division (Div)	The extent to which nodes will divide. This value is halved at each division	2 – 6
Subscriptions (Sub)	The maximum number of other cells each cell can Subscribe to	2 – 8

IV. RESULTS - 0 FAILURE RATES

The first group of tests do not induce any artificial failures within the nodes, therefore the failure rate of any node was 0%. Despite this they still permit self-organisation and may self-heal if a timeout occurs. These tests were conducted for two reasons: the first is to identify the lower and upper bounds for application execution given different network parameters. The second is to understand how the independent variables affect the structure of the network with no failure rates. This therefore provides baseline values for network size and structure according to the different independent variables. These tests examine the effect of the independent variables upon the structure and size of the network. Insights are gained from examining the quantitative network data and visually inspecting graphs.

A. Application Performance

These tests evaluate the effectiveness of the platform to deliver the applications. To provide context for the performance, table V shows the correlation coefficients for application performance versus the independent variables. *Out* could be argued to be the most relevant as a greater number of output messages means more successful data processing. The strong negative correlation between $Func$ and *Out* indicates that as application complexity increases the probability of a successful application decreases. Noting that this trend does not stay true for the *Proc* and *Recv* indicates that in many cases the applications are still being executed although just not to completion. An increase in both Div and Sub coincide with better application processing success overall.

Table VI shows the correlation coefficients again but only for rows where $Out > 0$. These values show that for successfully executed applications, those tests at lower success rates were less strongly correlated with greater values of Sub. $Func$ is strongly correlated with $Proc$ but this is likely due to the increase in quantity of messages being sent rather than any performance increase. Finally Div is positively correlated with Out. This is both intuitive due to the increase in N but also informative as it ensures that despite some network structures being conventionally non-resilient (hierarchical) the data is still processed successfully. $Proc$ values also increase with Div. With the shortest $Proc$ increasing with $Func$ but not the total, whilst $Proc$ does not increase with Sub. This is a very informative point, as it means that greater values of Sub can be used to enable resilience yet without the extra overhead caused by increasing numbers of nodes as Div increases.

TABLE V
CORRELATION COEFFICIENTS FOR APPLICATION PERFORMANCE

–	Div	Func	Sub
Proc	0.56	0.13	0.23
Out	0.30	-0.42	0.26
Recv	0.47	-0.10	0.37

TABLE VI
CORRELATION COEFFICIENTS FOR APPLICATION PERFORMANCE FOR SUCCESSFUL APPLICATIONS

-	Div	Func	Sub
Proc	0.65	0.47	0.23
Out	0.43	-0.22	0.21
Recv	0.59	0.18	0.39
ShortProc	0.27	0.46	0.12
LongRecv	0.39	0.00	0.00
TotalProc	0.43	0.09	0.09

B. Application Execution Success

Tests from all divisions where the division was highest and $subs > funcs$ executed the most successfully. Figure 4 shows

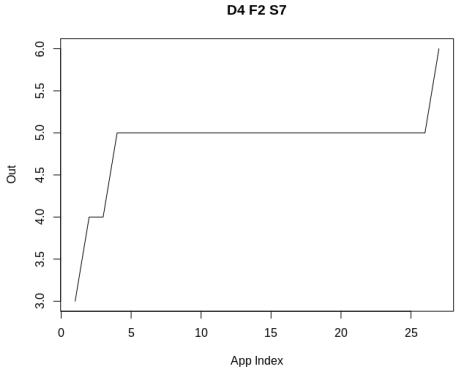


Fig. 4. $D4$ $F2$ $S7$ - Stable States

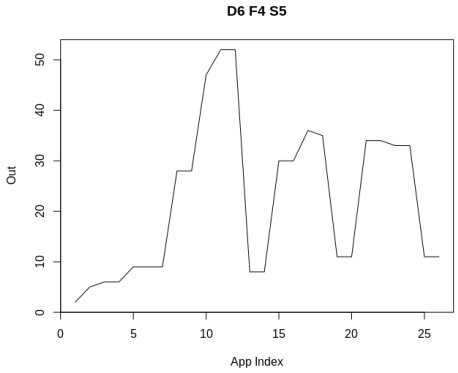


Fig. 5. $D6$ $F4$ $S5$ Output - Cyclic States

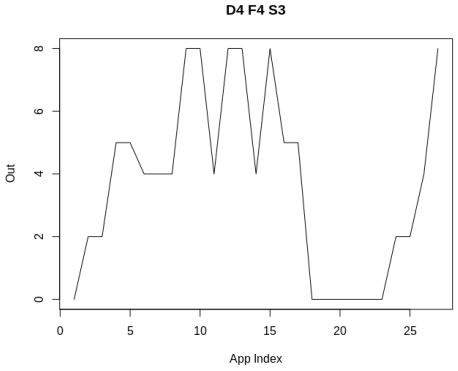


Fig. 6. D4 F4 S3 illustrating performance cycling between acceptable and degraded.

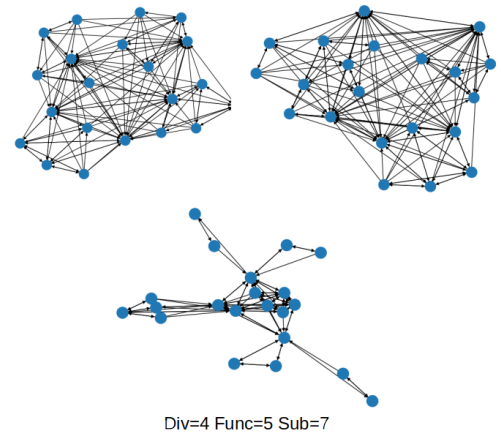


Fig. 7. 3 runs of the same test parameters D4 F5 S7 illustrating varying network convergence and chaotic conditions.

an example case of stable and successful execution output - a baseline. Figure 5 shows an example where applications are executed successfully but the performance cycles and presents a glimpse of the system states.

Tests configurations where the division is close or equal to the Subscription performed variably. The volatility in these tests provide insight into the network due to the variability in data which are discussed to provide insight into why the performance varied. Figure 6 shows the output message rate over time for a test where the system cycles between processing well (acceptable) and processing poorly (degraded). Neither of the other two test runs of this variant were as successful. This illustrates the importance of initial starting conditions.

The effect of starting conditions is highlighted in figure 7 where 3 runs produced two converged networks which did not execute applications successfully and one which did. This indicates the *chaotic* nature of this system. Due to the lack of node failures, network wills remain in this structure.

C. Unsuccessful Applications

A number of parameter sets produced no outputs over all test-cases. Particularly where $sub = 3$ or 4, and $func = 5$ or 6. Figures 8 and 9 shows the $Proc$ curves for $D4$, $F6$ and $S3$ and test $D6$, $F7$, $S3$. Process messages have a greater chance than output messages of varying in quantity during each peak

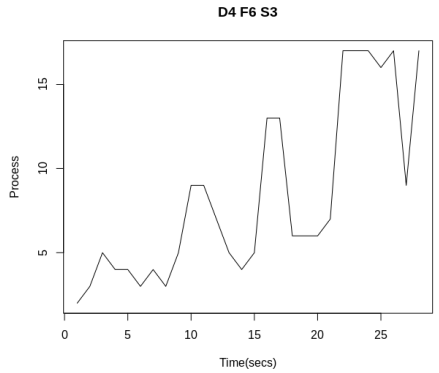


Fig. 8. Process Curve for D4 F6 S3 which had no successful application output

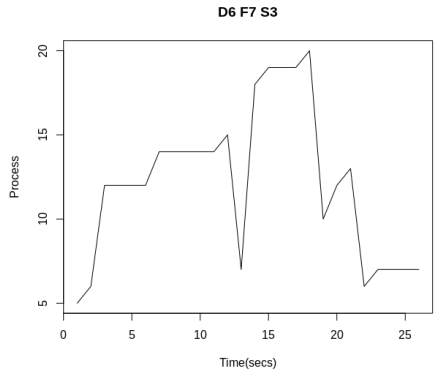


Fig. 9. Process Curve for D6 F7 S3 which had no successful application output

as they may be received by a different nodes depending upon the network structure and cell self-organisation. However the cycling nature of the peaks can still be seen in these graphs. The constraining factor here is the divisions and predominately the function size. A clear rule is that outputs will successfully occur when the Subscription size is equal to or greater than the function size.

V. Results - Failure Rate Tests

These tests follow from those conducted previously although with a fundamental addition of an artificial failure rate. At every 2 seconds, for every currently running child cell a pseudorandom variable is produced. If the variable is less than the given failure rate then the cell will be killed. The purpose of the failure rates is to evaluate the performance of the system under a high rate of churn. Therefore, these tests will evaluate the self-healing capabilities of the system and the application execution success alongside it.

The results in the previous section were concerned with the independent variable effect upon the structure of the network. Analysing the network from the perspective of it being converged is not suitable in this instance as the failure rates cause the network to be dynamic. Therefore the failure rate was set at a specific level to further understand the limits of the system under different parameters as a higher failure rate causes a greater perturbation of the system. In this instance, 0.01 was determined as a suitable value after manual experimentation. During preliminary testing, a slightly lower value of 0.001 causes the network to remain stable and high, evidence of which is provided later in this section.

A. Application Performance

To evaluate the application performance whilst the system is self-healing, tables VII and VIII present the correlation values for the independent variables against application performance variables for all tests and tests where the application was successfully executed, respectfully. In the values of all tests, the effect upon the quantity of messages processed performs less well in across all values compared to the 0 failure rate tests. In this instance as functions increase the quantity of messages processed actually decrease slightly, whereas in the 0 failure rate tests they increased ever so slightly. This illustrates the stronger relationship for tests which include failures between a high application complexity and the ability of the services to remain connected. For both out and $recv$, the performance is slightly worse than the 0 failure rate tests. These results are intuitive overall as the network is under stress in comparison to the 0 failure rates.

In terms of the successful applications, the relationships between the independent variables and the application performance on the whole follow the same trend as 0 failure rate tests yet with worse performance. Some times the performance degradation is to such a degree that the relationship is insignificant such as with $TotalProc$ and $LongRecv$ with $func$. This is interesting as it indicates that in this instance the time taken to complete applications is unrelated to their complexity.

TABLE VII
Correlation Coefficients for Application Performance in the Failure Tests

.	Div	Func	Sub
Proc	0.43	-0.11	0.04
Out	0.23	-0.54	0.10
Recv	0.41	-0.25	0.14

Whilst a higher quantity of output messages generally means the application has been executed successfully, it is important to note that a network with a higher output relative to another does not necessarily equate to better performance overall. For example if the ratio of messages ($recv : output$) is skewed towards $Recv$ then this may not be operating at

TABLE VIII
CORRELATION COEFFICIENTS FOR APPLICATION PERFORMANCE IN THE FAILURE TESTS FOR SUCCESSFUL APPLICATIONS

.	Div	Func	Sub
Proc	0.47	0.48	0.09
Out	0.39	-0.27	0.09
Recv	0.48	0.22	0.21
ShortProc	0.07	0.21	-0.02
TotalProc	0.25	0.00	-0.08
LongRecv	0.23	0.00	-0.07

optimal performance, considering resource cost. Additionally, the larger value of div which equates to a larger network will use considerably more resources. In terms of temporal performance, some interesting performance variations were seen. Particularly at the lower output performing tests and higher division rates. For example, D6 F5 S6 is 1.45 seconds faster than D6 F6 S5. Whilst D6 F4 S5 is a whole second slower again with 1 less function. This illustrates the potential non-linear speed increases through reduction of application complexity. These tests provide a comparison from the baseline presented for the 0 failure rate tests. Comparing the results there is a general trend across all values with worse performance than the 0 failure rates. This serves to validate the results as the failure rate is likely to worsen performance overall. Unlike in the 0 failure tests, sub appears to have no signification relationship with application performance. Out increases with div, which is again intuitive due to the increase in N. Finally the results of functions follow the same trend as in the previous tests, which is intuitive. The results indicate that application processing can still occur within the face of node failures within the correct parameters.

VI. CONCLUSION

This work presented the design of the embryonic data processing platform for resilient mobile edge networks. It then presented results and analysis to evaluate its efficacy. It illustrated that applications could be successfully executed under continued failures within the network. It also illustrated that the system had cyclic states which related to it's performance delivery. This highlights future areas of work to determine a metric to measure it's state and resilience.

REFERENCES

[1] M. Aazam, S. Zeadally, and K. A. Harras, "Deploying fog computing in industrial internet of things and industry 4.0," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4674–4682, Oct 2018.

[2] A. Singh, U. K. Singh, and D. Kumar, "Iot in mining for sensing, monitoring and prediction of underground mines roof support," in *2018 4th International Conference on Recent Advances in Information Technology (RAIT)*, March 2018, pp. 1–5.

[3] A. J. V. Neto, Z. Zhao, J. J. P. C. Rodrigues, H. B. Camboim, and T. Braun, "Fog-based crime-assistance in smart iot transportation system," *IEEE Access*, vol. 6, pp. 11 101–11 111, 2018.

[4] S. Heble, A. Kumar, K. V. V. D. Prasad, S. Samirana, P. Rajalakshmi, and U. B. Desai, "A low power iot network for smart agriculture," in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, Feb 2018, pp. 609–614.

[5] R. Amin, D. Ripplinger, D. Mehta, and B. Cheng, "Design considerations in applying disruption tolerant networking to tactical edge networks," *IEEE Communications Magazine*, vol. 53, no. 10, pp. 32–38, October 2015.

[6] I. Makhdoom, M. Abolhasan, J. Lipman, R. P. Liu, and W. Ni, "Anatomy of threats to the internet of things," *IEEE Communications Surveys Tutorials*, pp. 1–1, 2018.

[7] T. Welsh and E. Benkhelifa, "Perspectives on resilience in cloud computing: Review and trends," in *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, Oct 2017, pp. 696–703.

[8] F. Dressler and O. B. Akan, "A survey on bio-inspired networking," *Computer Networks*, vol. 54, no. 6, pp. 881 – 900, 2010, new Network Paradigms. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128610000241

[9] S. N. Mthunzi, E. BENKHELIFA, T. Bosakowski, and S. Hariri, "A bio-inspired approach to cyber security," in *Computer and Cyber Security: Principles, Algorithm, Applications, and Perspectives*, November 2018. [Online]. Available: http://eprints.staffs.ac.uk/5068/

[10] S. Hariri, M. Eltoweissy, and Y. Al-Nashif, "Biorac: Biologically inspired resilient autonomic cloud," in *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*, ser. CSIIRW '11. New York, NY, USA: ACM, 2011, pp. 80:1–80:1. [Online]. Available: http://doi.acm.org/10.1145/2179298.2179389

[11] B. Baran and R. Sosa, "A new approach for antnet routing," in *Proceedings Ninth International Conference on Computer Communications and Networks (Cat.No.00EX440)*, Oct 2000, pp. 303–308.

[12] D. Wang, L. He, Y. Xue, and Y. Dong, "Exploiting artificial immune systems to detect unknown dos attacks in real-time," in *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*, vol. 02, Oct 2012, pp. 646–650.

[13] W. Ha, "Cloud service selection with fuzzy c-means artificial immune network memory classifier," in *2018 14th International Conference on Computational Intelligence and Security (CIS)*, Nov 2018, pp. 264–268.

[14] T. Wang, B. Ye, Y. Li, and Y. Yang, "Family gene based cloud trust model," in *2010 International Conference on Educational and Network Technology*. IEEE, 2010, pp. 540–544.

[15] E. Baccarelli, P. G. V. Naranjo, M. Scarpiniti, M. Shojafar, and J. H. Abawajy, "Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study," *IEEE Access*, vol. 5, pp. 9882–9910, 2017.

[16] M. A. A. Faruque and K. Vatanparvar, "Energy management-as-a-service over fog computing platform," *IEEE Internet of Things Journal*, vol. 3, no. 2, pp. 161–169, April 2016.

[17] L. Lyu, K. Nandakumar, B. Rubinstein, J. Jin, J. Bedo, and M. Palaniswami, "Ppfa: Privacy preserving fog-enabled aggregation in smart grid," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3733–3744, Aug 2018.

[18] T. Welsh and E. Benkhelifa, "Bio-inspired multi-agent embryonic architecture for resilient edge networks," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2019.

[19] E. Benkhelifa, A. Pipe, and A. Tiwari, "Evolvable embryonics: 2-in-1 approach to self-healing systems," *Procedia CIRP*, vol. 11, pp. 394 – 399, 2013, 2nd International Through-life Engineering Services Conference. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2212827113005027

[20] D. Miorandi, D. Lowe, and L. Yamamoto, "Embryonic models for self–healing distributed services," in *Bioinspired Models of Network, Information, and Computing Systems*, E. Altman, I. Carrera, R. El-Azouzi, E. Hart, and Y. Hayel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 152–166.

[21] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *Computer*, vol. 49, pp. 112–116, 2016.

[22] S. V. Vandebroek, "1.2 three pillars enabling the internet of everything: Smart everyday objects, information-centric networks, and automated real-time insights," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, Jan 2016, pp. 14–20.

[23] P. Hintjens, *ZeroMQ: messaging for many applications*. " O'Reilly Media, Inc.", 2013.