

Loughborough University Institutional Repository

Emergence in active networks

This item was submitted to Loughborough University's Institutional Repository by the/an author.

Additional Information:

- A Doctoral Thesis. Submitted in partial fulfillment of the requirements for the award of Doctor of Philosophy of Loughborough University.

Metadata Record: <https://dspace.lboro.ac.uk/2134/14704>

Publisher: © Marcelline Shirantha de Silva

Please cite the published version.

This item was submitted to Loughborough University as a PhD thesis by the author and is made available in the Institutional Repository (<https://dspace.lboro.ac.uk/>) under the following Creative Commons Licence conditions.



For the full text of this licence, please go to:
<http://creativecommons.org/licenses/by-nc-nd/2.5/>

University Library

Author/Filing Title DE SILVA, M.S.

Class Mark T

Please note that fines are charged on ALL
overdue items.

FOR REFERENCE ONLY

0403116252



Emergence in Active Networks

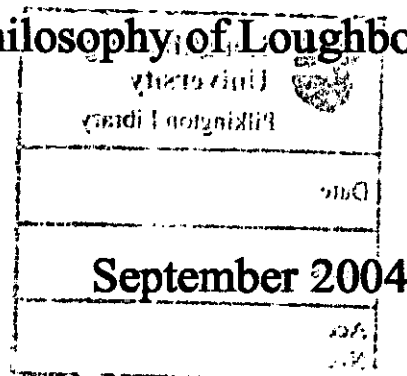
by

Marcelline Shirantha de Silva


A Doctoral Thesis

**Submitted in partial fulfilment of the requirements
for the award of**

Doctor of Philosophy of Loughborough University



© by Marcelline Shirantha de Silva 2004

	Loughborough University Pilkington Library
Date	SEPT. 2005
Class	T
Acc No.	0403116252

Dedicated to my wife Stephanie

A B S T R A C T

Abstract

Any complex system may potentially exhibit unpredicted and undesirable behaviour as a result of certain combinations of input stimuli. An Active Network, being a communication network in which user requested operations are undertaken in the network nodes themselves, is a candidate to exhibit such behaviour. For example, resource utilisation will be influenced by the specific combination of activities triggered by the users and may develop undesirable characteristics such as a self-sustaining profile. Conventional simulation tools do not detect such characteristics.

This thesis proposes a solution based on a Petri-Net model in which the resource utilisation of the Active Network is abstracted above the link level communication element. It is then suggested that a certain type of Emergence in resource utilisation may manifest itself as Self-Similarity. The Hurst Parameter (H) of the resource utilisation profile for each node in the network can then be used to identify the presence of this characteristic. The R/S Statistic is used to estimate sets of H values for a range of different Active Application scenarios. It is subsequently seen that a self-sustaining resource utilisation profile (termed a "Cascading Effect") occurs when a significant subset of the nodes display high values of H .

This thesis takes the view that Emergence in Active Networks is a problem that has to be approached with a global comprehension of the system as opposed to the conventional approach of a piecemeal development of solutions. This view is reinforced by the hypothesis that an Active Network is a Complex System and Emergence is non-complex self-organisation within it. It proposes that the high-level abstraction of the Active Network forms a view by which global comprehension can be obtained and is used for the detection of anomalous behaviour (i.e. Emergence). The key enabler for self-organisation is proposed to be 'the resources' within the Active Network nodes and hence the detection technique was focused on the utilisation characteristics of these.

A C K N O W L E D G M E N T S

Acknowledgements

I would like to thank Prof. David Parish for his continued support. His contribution of knowledge and advice, especially in the key initial stages of the project, has helped me focus on my objectives. Without his encouragement and support this work would not have been possible.

I would also like to thank, Mark Sandford, Pete Sandford, Andy Larkum and Les Dorrell (a former Active Networks expert) for their practical advice, support and prayers.

Many thanks go out to the rest of the HSN group for being patient with me in a noisy working environment.

Finally, I would like to thank my wife for her support, especially during the write-up stage. She contributed to this work by proof reading this entire document and made sure my 'i's were dotted and my 't's were crossed. She also restrained my obsession with commas, semi-colons and long sentences.

T A B L E O F C O N T E N T S

Table of Contents

Abstract	i
Acknowledgements	ii
Table of Contents	iii
List of Figures	ix
Background	xi
1 Introduction	1
1.1 Research Scope	6
1.2 The Project Strategy	9
1.2.1 Investigate Active networks and produce a suitable generic model	9
1.2.2 Investigate Emergent Properties and characteristics	9
1.2.3 Investigate suitable methodologies for the high-level abstract modelling of Active Networks	10
1.2.4 Development of suitable test scenarios for simulation	10
1.2.5 Obtaining results and identification of Emergence through established mathematical techniques	11
1.3 Original work	12
1.4 Summary of the thesis	13
1.4.1 Background	13
1.4.2 Chapter 1	13
1.4.3 Chapter 2	13
1.4.4 Chapter 3	13
1.4.5 Chapter 4	14
1.4.6 Chapter 5	14
1.4.7 Chapter 6	14
1.4.8 Chapter 7	15
1.4.9 Chapter 8	15

1.4.10 References	15
1.4.11 Appendices	15
2 Active Networks	16
2.1 Applications of Active Networks	20
2.1.1 Video-conferencing and Internet telephony	20
2.1.2 Mobile Internet	20
2.1.3 Caching and Load Distribution	21
2.1.4 Network management functions	21
2.1.5 Distributed sensors	21
2.1.6 Security	22
2.2 Active Network Research	23
2.3 Summary of chapter	27
3 Emergent Properties and Complex Systems	29
3.1 Characteristics of Emergent Properties	30
3.1.1 Feedback	31
3.1.2 Domino Effect (positive feedback)	33
3.1.3 Meta-Balance	34
3.1.4 Survival and Sameness	34
3.1.5 Vortex	35
3.1.6 Resonance	36
3.2 Emergence Research	38
3.3 Characteristics of Complex Systems	40
3.3.1 A Complex System consists of numerous independent components	41
3.3.2 Components interact locally and the interactions are numerous	42
3.3.3 Overall global behaviour is independent of the internal structure of the components	42
3.3.4 Overall behaviour of the system is well defined	42
3.3.5 Evolution in Complex Systems	43
3.4 Active Networks as Complex Systems	44

3.4.1 Distributed Processing Architecture	44
3.4.2 In-built Intelligence and Self Awareness	45
3.4.3 Local Network Awareness	45
3.4.4 Lack of Central Management Control	45
3.4.5 Application Level Organisation	46
3.4.6 Adaptation and Evolution	46
3.4.7 Memory	47
3.4.8 Limited Resources and Competition	47
3.5 Summary of Chapter	48
4 Modelling and Simulation	50
4.1 Simulation/Modelling Strategy	51
4.2 Static-Node simulation	53
4.3 Defining Characteristics of the Active Networks model	55
4.4 Defining Generic Model Applications and Primitive Functional Components	59
4.4.1 Possible Active Applications	59
4.4.2 Primitive Functional Components	60
4.5 Mathematical Solutions verses Simulation	61
4.6 Summary	64
5 Petri-Net Simulation	66
5.1 Modelling set scenarios with Petri-Nets	69
5.1.1 Sequential Actions	69
5.1.2 Cycles	69
5.1.3 Dependency	69
5.1.4 Concurrent processes	70
5.1.5 Synchronisation	70
5.1.6 Decision-making /conflict	71
5.2 Extensions of Petri-Nets	73
5.3 Petri-Net Simulation of Active Networks	74

5.4 Design/CPN	76
5.5 Model flow diagrams/layouts	78
5.5.1 Model hierarchy	78
5.5.2 Active Network (Ohira-Sawatari lattice structure)	79
5.5.3 Active Node Model	83
5.5.4 Local Storage Component	85
5.5.5 Merge Packet Component	87
5.5.6 Replicate Packet Component	89
5.5.7 Direction Solver Component	91
5.5.8 Other components: Data Logging	92
5.6 Summary	95
6 Detection of Emergence	96
6.1 Overview	97
6.1.1 Emergence as the loss of complexity in a system	98
6.1.2 Emergence through the measure of self organisation	98
6.1.3 Emergence as patterns in a system	99
6.2 Detection technique suitability testing	103
6.3 Self-Similarity	105
6.3.1 Calculation of the Hurst parameter – Rescaled Range statistic (R/S statistic)	105
6.3.2 Approximation and Stability of the R/S statistic	112
6.3.3 Cascading Effects and Self-Similarity	114
6.4 Summary	116
7 Results	118
7.1 Overview	119
7.2 Results: Case Study Analysis	121
7.2.1 Case Study 1	122
7.2.2 Case Study 2	124

7.2.3 Case Study 3	126
7.2.4 Case Study 4	128
7.2.5 Case Study 5	130
7.2.6 Case Study 6	133
7.3 Results Discussion	136
7.4 Summary	139
8 Conclusions and Future Work	140
8.1 Virtual Node Simulation	144
References	148
Appendices	
i. MATLAB Algorithm	158
ii. Design/CPN Petri Net Diagrams	161
ii.1. Declarations	161
ii.2. Model Hierarchy	169
ii.3. Top-level Active Network	170
ii.4. Active Node (section 1)	173
ii.5. Active Node (section 2)	176
ii.6. Active Node (section 3)	178
ii.7. Active Node (section 4)	186
ii.8. Active Node (section 5)	188
ii.9. Local Storage Component in detail (section 1)	190
ii.10. Local Storage Component in detail (section 2)	193
ii.11. Merge Packet Component in detail (section 1)	197
ii.12. Merge Packet Component in detail (section 2)	201
ii.13. Replicate Packet Component in detail	205

ii.14. Direction Solver Component in detail	207
ii.15. Data logging component	209
iii. Petri-Net Simulators Reviewed	216
iv. Case Study Tabulated Results	221
iv.1. Case Study 1	221
iv.2. Case Study 2	222
iv.3. Case Study 3	223
iv.4. Case Study 4	224
iv.5. Case Study 5	225
iv.6. Case Study 6	226

LIST OF FIGURES

List of Figures

Figure 1.1.a: The Cloud model overview of an Active Network	7
Figure 3.1.1.a: Examples of Positive Feedback	31
Figure 3.1.1.b: Dynamics of Positive Feedback	31
Figure 3.1.1.c: System representation of Negative Feedback	32
Figure 3.1.1.d: Dynamics of Negative Feedback	32
Figure 3.1.1.e: Two distinct views of Feedback	33
Figure 3.1.2.a: The Domino Effect	34
Figure 3.1.5.a: Computer representation of a Vortex	35
Figure 3.1.6.a: Example of a Resonant system	36
Figure 3.1.6.b: Resonance	36
Figure 3.2.a: Mathematical and scientific roots of emergence (Jeffery Goldstein)	38
Figure 3.3.a: The Global Emergence in Complex Systems	41
Figure 4.2.a: Ohira-Sawatari model adapted to Active Networks	53
Figure 5.a: Petri-Net terminology	67
Figure 5.b: The Transition with the red border is 'enabled'	68
Figure 5.c: The Transition with the double red border has 'fired'	68
Figure 5.1.1.a: Sequential action	69
Figure 5.1.2.a: Cycles	69
Figure 5.1.3.a: Dependency	70
Figure 5.1.4.a: Concurrent processes	70
Figure 5.1.5.a: Synchronisation of concurrent flows	71
Figure 5.1.5.b: Two-level deep buffer	71
Figure 5.1.5.c: Common resource store	71
Figure 5.1.6.a: Decision-making/conflict	72
Figure 5.5.1.a: Model Layout: Active Network Model Hierarchy	78
Figure 5.5.2.a: Model Layout: Active Network	80
Figure 5.5.2.b: Sample Source File ("src.txt")	81
Figure 5.5.2.c: Active Packet/Application Structure and Direction Indicator values	82
Figure 5.5.3.a: Model Layout: Active Node	83

Figure 5.5.4.a: Model Layout: Local Storage Component	85
Figure 5.5.5.a: Model Layout: Merge Packet Component	87
Figure 5.5.6.a: Model Layout: Replicate Packet Component	89
Figure 5.5.6.b: Active Packet Replication Scheme	90
Figure 5.5.7.a: Model Layout: Direction Solver Component	91
Figure 5.5.8.a: Model Layout: Data Logging Component	92
Figure 5.5.8.b: Sample Output log file "out.txt"	93
Figure 6.3.1.a: Examples of varying goodness-of-fit values (r^2) for regression lines	107
Figure 6.3.1.b: The Vonkoch self-similar curve	108
Figure 6.3.1.c: R/S statistic plot for the Vonkoch curve. Hurst value = 1.0519	109
Figure 6.3.1.d: Random trace	110
Figure 6.3.1.e: R/S statistic plot for the random trace. Hurst value = 0.5154	111
Figure 6.3.3.a: Process of experimentation and the definition of the "Cascading Effect" threshold of 0.9	115
Figure 7.2.1.a: Case Study 1: Hurst analysis of Active Network	122
Figure 7.2.2.a: Case Study 2: Hurst analysis of Active Network	124
Figure 7.2.3.a: Case Study 3: Hurst analysis of Active Network	126
Figure 7.2.4.a: Case Study 4: Hurst analysis of Active Network	128
Figure 7.2.5.a: Case Study 5: Hurst analysis of Active Network	131
Figure 7.2.6.a: Case Study 6: Hurst analysis of Active Network	134
Figure 8.1.a: Virtual Node Simulation of an Active Network with a uniform lattice of cells	145

B A C K G R O U N D

Background

Multimedia is a collection of text, speech, graphics, audio and video formed into numerous applications meeting the needs of corporate and domestic clients alike. The progress of the Internet is towards the provision of these enriched services over IP, which is the defacto standard for fixed infrastructure networks. In conjunction, the recent rise of wireless data communications has provided significant impetus to the research of viable technologies providing guaranteed Quality-of-Service (QoS) and efficient resource utilisation [Kul99]. The intrinsic nature of wireless networks mean that bandwidth is at a premium in heavily congested airways. Currently, IP struggles with the legacy of having been originally designed for text-based communications and with the problem of maintaining backward compatibility with existing hardware.

As the size of the Internet expands from millions of nodes to billions of nodes, the information richness and capabilities of the current TCP/IP protocol will see its limits. A current topic of discussion in improving the Internet is the focus on adding Quality-of-Service (QoS) controls on Routers (to support real-time communications, reliable distributed multicasting and multi-party interactive communications) [Mar99]. However, as more services are added and deployed on a large-scale, the cost (the major sink for funds would be the extension of infrastructure and network management) and complexity of the system will grow rapidly.

C H A P T E R 1
I N T R O D U C T I O N

1 Introduction

This chapter will present:

- A brief description of Active Networks.
- A brief explanation of Emergence.
- The contributions made by this work to further the understanding of Active Network behaviour in Emergent situations.
- The strategy undertaken to develop a detection technique for Emergence in Active Networks.
- The idea that Active Node resource usage is linked with Emergent Behaviour.

The key concepts in this chapter are: Emergence, Active Networks, original work, Self-Similarity, resource usage fluctuations, detection technique, high-level abstract model

This research project primarily brings together two fields of study in order to present an interesting problem that would pose significant barriers to the implementation of the discussed technology.

The first is a new networking concept called Active Networks; envisaged and promoted by the networking community as a radical alternative to IP (Internet Protocol) networks. It is an advanced internetworking technology that would provide increased throughput of multimedia and the efficient usage of bandwidth. Active Networks, however, do nothing to simplify the complexity of the current Internet structure. The addition of Active Networking components (as will be shown later in this thesis) would, in fact, raise levels of complexity.

The second is Emergence, which is the term given to the uncontrolled manifestation of system-wide structures (good or bad) through the dynamic interactions of individual system components. Emergence is a topic of high interest to many research communities including the systems, physics and mathematical communities. It is generally understood that the term Emergence is the collective definition for system behaviour over and above what can be practically understood in a complex system. The discovery of Emergence is proposed as a practical methodology to evaluate complex system performance without the need to understand every single facet of system behaviour [Gol99].

Both fields, even though thoroughly researched, lack a set of coherent principles, standards and definitions. The directions of research (in Active Networks and Emergence) are varied and changeable. In this environment this research project proposes to merge the two fields and find a practical solution to the detection of certain types of Emergence in Active Networks. The detection scheme proposed by this research is one part of the overall design and modelling process; undertaken as a stage in the development process of a stable complex system for the Internet.

Active Networks promises to be a more radical solution to current Internet woes and describes the implementation of a *'user-subscribed customisation of the data connection'*. In other words, the end-users could be given the capability to modify routers to specifically enhance the application throughput along it's connection path;

requesting resources based on the type of application, the required Quality-of-Service and the network state for the duration of the connection. Active Networks is the term given to this distributed processing environment that lacks any central management control.

The lack of central management control is not exclusive to Active Networks. The current IP protocol owes its popularity to the fact that it provides easy integration to the fabric of the Internet without the authorisation of any one supervisory body. Active Networks propose to go one step further and allow users (the clients and servers) the ability to customise the core of the Internet.

The extent to which the 'end-users' are allowed to control the network is a highly debated issue. Giving individuals Active Capability would open the network up to a whole host of security, integrity and 'political' problems (e.g. possible threats from hackers, erroneous software creating system wide crashes and users who will insist on priority for their applications). It is possible that Active Capability, in the future, be given to accountable organisations (e.g. Internet Service Providers, multi-media application servers, etc.) with an attached premium or be integrated to software packages designed for the Internet. Several authentication/integrity steps also need to be taken to ensure that Active packets are not erroneous or malicious [Bro01]. The disadvantage of this would be that it is a step away from the original concept of giving end-users total flexibility in the deployment of Active Applications. Requiring approval and authentication would also delay the deployment of new multi-media enriched services. However, many believe that a compromise is essential if it is to become a reality.

One factor in the success of any new technology is its rapid deployment onto the Internet. From its widespread use in the 1980's the Internet was a place where new technologies were expected to be in service in a couple of years from their initial inception. That margin of development time is constantly being reduced and designers are under pressure to get their products out onto the market before the technology becomes obsolete. The Active Network paradigm provides a significant step towards reducing this development time, through the use of its open-protocol customisable data

connection capability. It further facilitates the development of new services currently not envisaged.

The current process of thought for the development of Active Network applications is founded on improving existing Internet solutions (i.e. to provide certain levels of QoS and reliability for multimedia services). An example would be the Active Network equivalent of the Multicast Backbone (Mbone) - a current implementation of IP multicast protocols. IP multicast schemes, including Mbone, allow a user to 'broadcast' packets of information to selected groups of 'listeners' using a reduced set of traffic streams as opposed to multiple one-to-one (normal IP unicast) streams that would congest the network. Reliable and scalable multicasting would lead to a revolution in the publishing of data, audio and video across the Internet. Even today we see governments, businesses, television/radio networks, educational establishments, the music industry and many other multimedia developers using IP multicast to reach audiences throughout the world [Sav96a]. The Mbone is a software alternative to what should ideally be a hardware-based technology. It provides a scheme of moving multicast packets through the network by encapsulating them in unicast IP packets (a process called tunnelling). This 'workaround' implementation came about in order to provide backward compatibility with existing routers and servers [Sav96b]. A hardware solution is ideal as it would increase the speed and reliability of Multicast connections. Even with today's production of routers with added hardware-based multicasting, the goal of reliable and scalable IP multicasting is not yet reached. The consensus is that Active Networks will provide the strongest solution to the problem without sacrificing the flexibility to service other applications [Ten97].

The introduction of Active networks into the arena of emerging Internet technologies has opened up several issues including security and wide-scale integration with legacy devices. Few research organisations have concentrated on the little understood topic of 'Emergent Properties' and 'Emergent Behaviour' in relation to networked systems. Whilst there are many hypotheses and partial solutions to customised applications, no universally acceptable models and theories have been formed. It is apparent that while considerable research work does exist in the fields of Active Networks and Emergent

Properties, separately, the idea that both are linked has received little attention. This is understandable since the definition of Emergent Behaviour has various meanings under varying levels of abstraction. The complexity and broad scope of Emergent Behaviour (tantamount to the phrases “Emergent Properties”, “Emergent Phenomena” and “Emergence”, for the purpose of this thesis) have been difficult to breakdown and analyse in accordance with standard engineering practices. The ‘systems’ approach, of viewing the problem in distinct sets of sub-elements and providing a progressive set of solutions, is inadequate [Mar99]. One would require a holistic approach to Emergent Properties as opposed to a piecemeal systems analysis in order to gain an accurate result. This does not mean that all details of an Active Network have to be included in the modelling process. One can selectively include features suspected of contributing to Emergence in Active Networks [Yua02].

The solution to the problem is by no means a straightforward simulation exercise. The detection of Emergent Behaviour in Active Networks is complicated by the fact that there is no set model for this type of network; as major research organisations are in the process of developing and testing various topologies and distribution mechanisms, all under the umbrella of Active Networks. This thesis presents a method to overcome the lack of standardisation by forming a generic modelling/simulation scenario. There is some agreement as to what is possible through Active Networks and in the type of services that users want or seek to have in the near future. This forms the initial point of reference for a proposed Active Network modelling/simulation scenario with which to investigate the issue of Emergence in Active Networks.

1.1 Research Scope

There are conceptual differences between existing networks and Active Networks that are further highlighted by the amount of Complexity within each. Whilst classic networks can be modelled and analysed by equivalent equations or captured data traces, a highly complex Active Network requires additional methods in order to understand its behaviour, evolution and performance.

“Complexity is the property of a real world system that is manifest in the inability of any one formalism being adequate to capture all its properties. It requires that we find distinctly different ways of interacting with systems. Distinctly different in the sense that when we make successful models, the formal systems needed to describe each distinct aspect are NOT derivable from each other.” - D.C. Mikulecky, Professor of Physiology, Medical College of Virginia Commonwealth University.

One objective of this project is to confirm the existence of Emergent Properties in Active Networks and devise models that would facilitate the detection of them. Another is to describe an approach to detect Emergence and ultimately ascertain the practicality and viability of Active Networks in future networking solutions.

This constitutes a management process with certain complications. Given the fact that Internet usage is multi-fractal and Active Networks are programmable (they have no restrictions on functionality and therefore state), the number of possible states a large-scale network can be in is essentially infinite. Thus an information-modelling approach or a finite-state-machine (foundations in Control Theory and Systems Analysis) approach is not applicable [Mar99].

As a background to a possible solution, this thesis highlights key concepts of Emergent Properties and their identifying characteristics. It follows on to justify Active Networks as Complex Systems with the capability of producing Emergence, and draws insights from varied fields of study not necessarily network related:

- Artificial Life

- Complex Systems and Chaos Theory
- Biological Systems including human neurological and immune response systems
- Multi-Agent systems
- Genetic Algorithms and Artificial Intelligence
- Physical systems
- Mathematical equations

Techniques and theories related to Complexity Theory and Complex Systems research provide the most credible and complete solutions to the modelling of Active Networks and Emergence, and have directed the course of this research (a discussion of which is provided in this thesis).

From a System's perspective an Internet Environment consisting of Active capability can be viewed as a collection (or cloud) of nodes in between the server and client (i.e. all clients and servers are at the edge of the network and all the routing/transportation hardware is in the core). The cloud would provide additional processing of the connection between the server and client that is dynamic and locally aware. It is a high-level abstract view of the Internet with Active Network elements distributed within it. The author proposes that this model is sufficient for a **functional analysis** of Active capability and its side effects. It also does not promote any particular configuration of Active Nodes in order to further the progress of discovering Emergence.

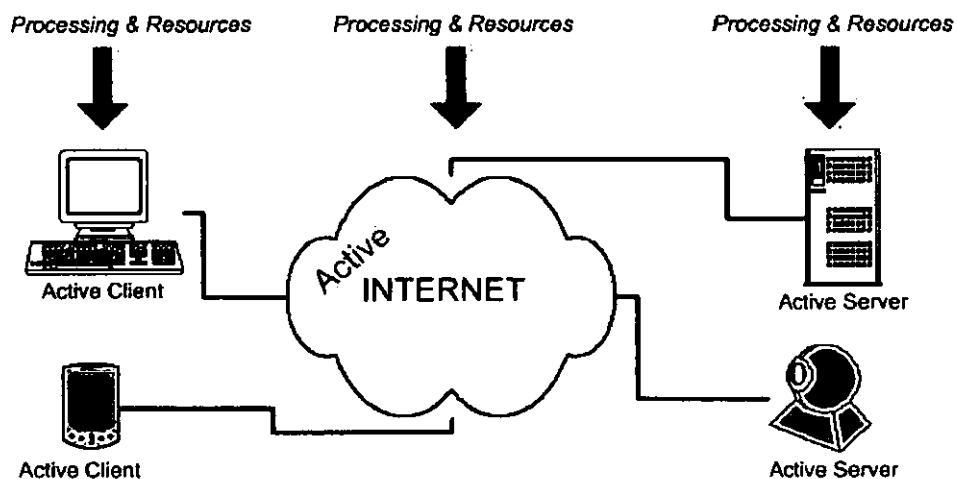


Figure 1.1.a: The Cloud model overview of an Active Network

The work contains an investigation of the manifestation of unusual (and possibly undesirable) phenomena commonly termed as “Emergent Behaviour” [Bar97] within generic models of an Active Network and was successful in identifying one in particular, which the author has termed a **“Cascading Effect” in node resource utilisation**. In brief, a “Cascading Effect” is a phenomenon linked to ‘replication’ (for the purposes of this work) that, once formed within the Active Nodes, continues to sustain itself (feedback) and grow; the detection of which can only be done post simulation. The phenomenon is distinguished through a set of empirical results and highlights the causal input conditions of the Active Network. It might be assumed that one can intuitively point out potential Active Applications that would always cause such a phenomenon (i.e. Active Packets that have a replicating function, built into them). However, as described in chapter 7, section 7.3, not all replication scenarios produce a “Cascading Effect”. A methodology is devised to analyse the potential of replicating packets to create “Cascading Effects” in the Active Network model.

The method, once verified, could then be used in future simulations of Active Networks (extraneous to this work) to identify Cascading Effects, even when there are no ‘apparent’ Active Applications with replicating elements. This would be tantamount to the detection of ‘Emergence’, or at least one type of Emergence. Detecting “Cascading Effects” is important for Active Network designers, as this is an uncontrolled and undesirable situation of resource fluctuations that can quite easily manifest itself through the code invoked by Active processes. The formal methodology will provide a generic test bed for legitimate Active processes and a suitable detection mechanism for undesirable culminations of Active Applications.

The underlying mechanism for the “Cascading Effect” is **“Self-Similarity”** [Naw95]. A trace is said to be self-similar when it roughly looks the same on various time-scales (i.e. scale invariant) and is synonymous with long-range dependence. A “Cascading Effect” by its very nature would generate feedback loops adding to the existing effect. This would result in similar patterns emerging when viewed at various time scales. To verify the “Cascading Effect” hypothesis, the author used the R/S technique to calculate the Hurst parameter (a well-known estimator value of Self-Similarity) of each data trace obtained from the simulation.

1.2 The Project Strategy

The research work strategy was based around the following key sections:

1.2.1 Investigate Active networks and produce a suitable generic model

The generic model, which forms the starting point for the analysis of Emergence, had to be sufficiently succinct and complete; taking into account the relevant features that have a possible link with anomalous behaviour. Hypothesising about all possible factors of Emergence would be an exhaustive process if it were not for the fact that the generic model, itself, provides a reduction in information/detail, and thus reducing the number of Emergence factors under consideration. Nevertheless, highlighting Emergence factors was an ongoing task, thus making the production of the generic model an iterative process.

A review of Active Networks research was performed in order to gain an understanding of the concepts and possible applications. These applications provided the functional descriptions for the generic model.

1.2.2 Investigate Emergent Properties and characteristics

The research delved into areas such as the natural clustering of organisms (e.g. flocking birds models, human traffic scenarios), Emergence in Complexity, road traffic modelling, cognitive science, aircraft systems, multi-agent systems and the Internet. Each field of science described a system with particular Emergent characteristics. The investigation was primarily directed to draw analogies from the above-mentioned fields that relate to Active Networks. Appropriate results and conclusions were found from past research that were used as foundations for the modelling of Active Networks. Further to this, the research work undertook a critical breakdown/analysis of Emergence and its various characteristics. The classic makeup of Emergence was identified as exhibiting several patterns endemic in system-wide behaviour.

1.2.3 Investigate suitable methodologies for the high-level abstract modelling of Active Networks

The Active Network was modelled as an abstract view in order to reduce the amount of detail it contained and to become a generic case for the different Active Network architectures currently being researched. The modelling process and simulation process were initially grouped as two separate tasks. However, as the research progressed it was found that a particular choice of modelling scenario also provided a list of recommended simulation solutions as part of the package. Thus, the consideration of any modelling technique had to take into account the availability and ease of use of the recommended simulators. Suitable modelling techniques included Petri Nets and Cellular Automata (both being simulation based techniques). It was decided that a direct simulation approach of the Active Network functionality was more appropriate than attempting to build a model comprised of mathematical equivalent equations of system dynamics. The latter technique, being more involved, would have required an unacceptable number of assumptions to be made and would have been inflexible to an iterative development process. Where there was more than one simulator for a particular modelling technique, an evaluation was performed based on the following:

- Availability
- Ease of use and a suitable level of modelling detail
- Industry recognition and support
- The ability to graphically layout the model
- Modular design capability and hierarchical decomposition
- Speed of simulations
- Ability to export results as text files

1.2.4 Development of suitable test scenarios for simulation

Once the iterative design process produced a credible example of an Active Network it was subjected to various input scenarios whereby it was hoped that at least one defining Emergent characteristic would manifest as a pattern within the system. Defining characteristics were initially identified and classified through a review of Emergence

research. The test scenarios contained parameters that threw the generic Active Network into unstable situations, from which it was hoped Emergence would be produced.

1.2.5 Obtaining results and identification of Emergence through established mathematical techniques

The use of a particular mathematical analysis technique is dependant on the type of data produced by the network and the selected input simulation conditions. Since Emergence manifests itself as patterns in the system and is reflected in the data, the mathematical techniques considered were based around pattern recognition. Some of the techniques evaluated included Fourier analysis, Wavelet analysis, Image Recognition/Enhancement algorithms and Self-Similarity algorithms. The 'type' of result returned from the simulation also merited careful consideration. For this research, the **Resource Usage Fluctuations** in Active Nodes proved to be the most suitable data for analysis, since they are **part of the evidential link between Emergence and the dynamic interactions within the Active system.**

1.3 Original work

The decomposition and classification of an Active Network based on higher-level functionality is a crucial concept and is believed to be unique to this project. With this process it is now possible to abstract-model an Active Network with minimum detail and to rapidly provide generic models designed as test-beds for Active Applications. Performance evaluation is further simplified with the use of a universal modelling theory known as Petri Nets.

The analysis of a large-scale system must contain a preliminary list of parameters/outputs to be observed; being a sub-set of all possible characteristics that can be analysed post simulation. Whilst there are methods in Systems Theory to formally identify these parameters based on preset evaluation criteria they do not extend to isolate the parameters that are of particular importance when considering Emergent Behaviour. A systematic approach to identifying parameters of interest is inadequate. A more 'systemic' approach, using techniques imported from non-systems related fields (e.g. Complex Theory), is needed. Using this philosophy the author has noted that Emergence could manifest itself, in a highly distributed and highly connected network of intelligent sub-systems (nodes), through the dynamic interactions of those sub-systems. Any dynamic interaction requires a quantity of resources be allocated to that event, and thus the key parameter for Emergence Behaviour analysis would be the resource usage fluctuations in nodes. From an Active Network perspective these nodes are individual Active routers or any piece of hardware with Active capability. The concepts and techniques are sufficiently universal to be used in the analysis of other highly distributed networked systems such as the current Internet, Adhoc Networks, Intelligent Networks, wireless networks and modular avionics systems for aircraft. The technique would provide an additional test for critical systems design.

1.4 Summary of the thesis

1.4.1 Background

1.4.2 Chapter 1

Introduction

1.4.3 Chapter 2

Chapter 2 of this thesis provides an in depth analysis of Active Networks including a brief history and the current stage of development of the area. Particular attention is given to the functionality of this new paradigm in networking; giving examples of instances where multimedia applications are improved through Active capability. The chapter then proceeds to review the current state of Active Network research, giving a list of organisations that are pioneering in this field.

1.4.4 Chapter 3

Chapter 3 provides a detailed analysis of Emergence including examples of systems displaying Emergent Behaviour. The chapter defines the terms Emergence, Emergent Behaviour and Emergent Properties in relation to this piece of research and makes the distinction between good and bad types (i.e. Emergent engineering and anomalous behaviour, respectively). A review of current Emergence research is included in this chapter in addition to any convergence of ideas, noted through the varied fields of study. The second section of this chapter concentrates on Complex Systems and Complex Systems research and provides an introduction - defines what a complex system is and its differences with other types such as chaotic systems. It proposes that Active Networks be considered as Complex Systems or at the very least as having complex elements. The chapter also proposes that (citing previous research) Complex Systems are likely to exhibit Emergence under certain conditions.

1.4.5 Chapter 4

Chapter 4 describes the modelling and simulation techniques used in this project. It describes in detail the modelling methodology used along with its advantages over other methods that were also considered. The chapter then proceeds to describe the development process of the Active Network model and includes all assumptions, estimations and configurations made during the iterative development process. The generic Active Network model is rigorously defined – the development process starts by the analysis, breakdown and classification of Active Applications into Primitive Functional Components (PFCs). The components form the foundation for a hierarchical, modular and ‘*top-down*’ design.

1.4.6 Chapter 5

Chapter 5 describes in detail the modelling paradigm used in this project - Petri Nets. It describes the advantages and provides generic modelling examples of system elements (used as templates in the modelling of an Active Network). The chapter also highlights the extended capabilities of *Coloured* Petri Nets; their suitability in describing Active Packets, the dynamic interactions within the system and the critical resource usage in Active Nodes. A review of Petri Net simulators is undertaken, which describes the advantages of Design CPN (the simulator of choice) over others.

The second section of this chapter provides detailed Petri-Net diagrams and a hierarchical layout of the Active Network model. Also included in this description are:

- A breakdown of the Active Packet header, which includes Active Packet types (based on Primitive Functional Components)
- Sample input and output files
- A mechanism for logging results

1.4.7 Chapter 6

Chapter 6 is dedicated to identifying Emergence within simulated results. It describes the strategy undertaken and the method ultimately used to detect Emergent Behaviour, which was preceded by a review of possible pattern detection techniques. Each

technique was tested for suitability for the Active Network modelling scenario. The advantages and shortcomings of the R/S technique (“ReScaled-range statistic”), for the purposes of detecting Self-Similarity in the simulation data, are discussed in detail. This is in relation to the applicability of the technique to Emergence and Active Networks. The straightforward procedure in applying this mathematical technique, through MATLAB software package, is also described. This chapter also establishes a link between the identification of an Emergent Behaviour and Self-Similarity.

1.4.8 Chapter 7

Chapter 7 provides the main body of results indicating the presence of Emergence. It describes all input test scenarios/cases and provides graphical representation of results (derived through the R/S statistic). These results are linked with the initial input conditions for the simulations, which then can be used in the discussion process. An analysis and discussion of the results are provided, which include a credible theory on the causal link between a particular Emergent output and its input conditions.

1.4.9 Chapter 8

Chapter 8 provides summative conclusions of the results achieved by this work. It contains the direction undertaken by the research work and provides a discussion of possible improvements to the detection of Emergence (e.g. additional techniques to detect other types of Emergence). The chapter also describes an alternative approach to the modelling methodology used by this project.

1.4.10 References

1.4.11 Appendices

The appendices contain the MATLAB algorithm code for the detection technique (i.e. R/S statistic calculation) and the detailed diagrams of the Petri-Net Active Network model.

C H A P T E R 2
A C T I V E N E T W O R K S

2 Active Networks

This chapter will present:

- A detailed description of Active Networks, their capabilities and potential problems.
- A review of developments in Active Network research.

The key concepts in this chapter are: Active Networks, DARPA definition of Active Networks, Active Network research

During 1995, The Defence Advanced Research Projects Agency (DARPA) sponsored an Information Science and Technology (ISAT) study, entitled "Virtual Infrastructure", from which a new research initiative, called Active Networks was born [Mau02]. Its mission was to develop "*networks that turn on a dime*" [DAR].

The specific goals of the DARPA programme were to achieve the following (as quoted in their information resources):

- *Quantifiable improvements in Network Services –*
 - *Audio/video synchronisation and full-rate video over multicast a reality*
 - *Fewer retransmitted packets, 100% increase in useful data rate to end applications*
 - *Architecture based solutions to Future Department of Defence (DoD) needs*
- *Fault-Tolerance Mechanisms based inside the network*
- *Multi-Tiered Mobile Security -*
 - *Authentication forms used for dynamic access control*
 - *Separate traffic and administrative functions based on types and policy*

Through the above-mentioned specifications it was hoped general wide-ranging improvements to networks would be made. For example:

- *The ability to quickly and safely deploy new services.*
- *Achieve widespread use without need for standardisation process.*
- *Be able to upgrade crucial network services to keep pace with network complexity.*
- *To develop new strategies for routing and service provisioning in large networks that have overlapping topologies and mobility requirements [DAR].*

Active networks are a multi-service Internet architecture, designed for the rapid deployment of new services and technologies over the Internet (this being the ultimate goal). An Active Network will consist of routers and switches that can be programmed to perform certain functions on packets flowing through them. In contrast, traditional networks simply provide a transport mechanism for data transferred between clients and servers with minimum computation (e.g. header processing). As a part of Active

functionality it is possible for users to inject programmes into Active hardware along a particular data traffic path in order to modify, store and redirect packets.

'Nodes' are the definition given to routers, switches and other network layer devices that carry Active capability (i.e. the nodes can perform computations on, and modify the packet contents on a per-user or per-application basis) [Ten97]. This results in a customisable network that could readily be incorporated into the current Internet infrastructure, thereby ensuring interoperability with legacy routers and switches [Ten97].

The fundamentals of customisation of Active Networks are to allow the users to inject programs into the Active Nodes of the network. There are many variations to this concept and programmability, which in its purest form is defined to augment the standard IP packet datagram with Active 'capsules' (programme fragments that are executed at each node that it traverses and have the capability to affect further packets). These capsules will be loaded onto Active Nodes based on the availability of the correct resources to open and run the programme, the amount and type of resources requested to service subsequent packets in an Active Application stream, signature/security verification, outward-link status and congestion. The Active capsule approach (also known as the 'integrated approach') has advantages of being able to rapidly deploy services, use generic Active hardware, be easily customised and have increased information throughput. The flexibility is such that Active Nodes can be programmed to perform multiple actions/computations on multiple packet streams flowing through them at any given time. The nodes would also be able to factor the local network state/environment in its decision process, thus making the actions dynamic.

The less flexible 'discrete approach' involves the use of Packet Header Options to invoke a pre-loaded set of service components (i.e. primitives) from an Active Node within its Execution Environment (EE). The development of an API styled structure is necessary to facilitate this execution process. The discrete approach allows system administrators to maintain control over the programmes that are run on their hardware, thus improving security (lacking in the integrated approach). Customisation, however, is limited to a pre-defined set of services that can be built through the primitive components; present at a particular node at a given time. This can be augmented,

however, using a “Downloading on Demand” strategy whereby an Active router can request any primitive programme component that it lacks from its neighbours.

Current Internet technologies will benefit from the new networking paradigm, as it replaces several ad-hoc techniques, already implemented to various degrees of success, with a common networking substrate with benefits of added flexibility and performance [Wet98a]. Historically, the array of services being offered on the Internet has been met with limited enthusiasm because the available infrastructure was not efficient. For example, the current TCP/IP protocol is inefficient for ‘real-time’ communication and in particular with video. Successful services provide features to enhance the applications at end-systems [Mar99]. This is in keeping with the “end-to-end argument” of a communication network – a rationale used in layered system design to place high-level functionality nearer to the application that uses it (i.e. further up the layered structure). A system design following the principles of the “end-to-end argument” will not place high-level functionality in lower layers of the system, especially if such functionality is only used by a select group of applications [Sal84].

Active Networks re-interprets the “end-to-end argument” to allow high-level functionality in lower-layered communication devices. This benefits services that can only be supported or enhanced inside the network [Bha97].

2.1 Applications of Active Networks

The 1995 DARPA objectives provided the foundation for the development of Active Applications. Some of the popular services, which will greatly benefit from an Active network layer within routers/switches (i.e. being researched as Active Applications), are mentioned below:

2.1.1 Video-conferencing and Internet telephony

These are two types of multimedia streaming applications that will use real-time and multicast services. The applications will be able to reserve resources and bandwidth through routers, so that streaming is contiguous. IP/Active multicasting will reduce the bandwidth needed to communicate (from one sender to multiple users) by having the routers/switches cache the data as well as process acknowledgement feedback [Wet98a]. Internet telephony has the means to be unimpeded by isolated packet losses within a communication session (e.g. Internet voice applications use sample based coding of the analogue signal and Adaptive Packetisation and Concealment schemes employed at end-stations are very good at maintaining a perceived quality for users). However, they are not capable of concealing ‘burst type’ losses of packets, which result in the significant degradation of speech quality. In such circumstances Active Nodes can be used to regenerate lost packets and inject them into the stream [Le00].

2.1.2 Mobile Internet

Notebooks and other mobile IP devices will benefit greatly by services deployed through Active routers; optimised for wireless transmission. For example, Audio and Video transcoding and compression routines running on Active base-stations will compensate for limited bandwidth of RF communications. Including Forward Error Correction (FEC) can compensate for lossy transmissions. Mobile IP devices can access the Internet at different sites without the need to reconfigure address information [Wet98a].

2.1.3 Caching and Load Distribution

Web servers would be the main beneficiaries of these services. They would reduce the amount of wide-area traffic by allowing the Routers to intercept and process repeated requests from multiple users. By distributing the requests over a number of cache servers, the network is capable of reducing the web traffic concentration of particular sections. The services, transparent to end-users, would minimise latency and bandwidth usage, compared with proxy agents. Today, specialised products such as Cisco's CacheDirector and LocalDirector perform these same functions as vendor promoted systems [Wet98a]. With Active networks, the caching can be taken a step further by having a significant proportion of the web pages dynamically generated within the caching nodes themselves. This would suggest a scheme of Active nodes that support the storing and execution of web generation programmes. Whilst in normal caching hierarchies the servers are fixed, an Active network can dynamically re-position the caches when necessary [Ten97].

2.1.4 Network management functions

A decentralised scheme of network management will come into being with user end-stations being the initiators of most of the management operations along with other applications (the operations themselves being executed on the nodes), thus reducing the overall network management costs [Mar99].

2.1.5 Distributed sensors

Effective viewing of a 'composite image' of a large distributed network of 'sensors' requires the fusion and storage of multi-type data within the network (the merging of data offers significant bandwidth reductions). Active Networks provide this capability along with the ability to provide multiple user access to each sensor. The viewed data can also be differentiated based on the customisable requirements of the user [Ten97].

2.1.6 Security

Active Networks can apply 'dynamic' security to the network, which would involve users and applications being able to apply highly customisable and situational security policies for each or groups of information transfer streams. Active Routers within the network can provide authentication, encryption and access control (e.g. of node resources) for Active Applications at optimal points along the transfer path, which also benefits the entire Active routing/processing system by ensuring the integrity of the Active programmes being run in it. Active packets can also be specifically designed/installed in order to route out and neutralise unwanted/malicious traffic, thus creating a dynamic response to security threats (i.e. dynamic firewall formation) [Cam00].

Security devices, such as firewalls, apply filters based on various fields in IP packets such as source/destination address, requested service, etc. Active Networks allow the dynamic programming/updating of Firewalls with filter modules that set permissions, thus new Active Applications from approved vendors can automatically authenticate themselves without the need for 'system administrator intervention' (i.e. a static security policy becomes a dynamic security policy) [Ten97]. The security policy can spread through the network on a 'need to know' basis and can 'actively' react to any invalid attempts to access data by changing its policy level [Liu00].

With current Internet schemes many of the innovative network services mentioned above are possible by using Agents at end-stations (e.g. allowing web servers and clients to exchange Java applets) and overlays (e.g. the MBone) [Wet98a] [Ten97]. However, deploying them as a network layer element within routers and switches offers considerable improvement in functionality and performance.

In general, some of the current requirements of multimedia-enriched applications are that they be adaptive (i.e. have the capability to change their functional behaviour based on environmental conditions such as network congestion). Active Networks would support and enhance these types of applications by allowing the optimal use of network resources; through the execution of Active code at strategic points within the network [Yam00].

2.2 *Active Network Research*

Various organisations have been spearheading the research into Active Networks; specifically defining the enabling technologies and dealing with issues such as node security, capsule and programmable switch architecture, and compatible applications.

The following is a list of mechanisms, execution environments, operating systems and test platforms developed, or currently being developed, to facilitate the transportation and execution of Active Packets through a network:

- ANTS (Active Node Transfer System) was originally devised by researchers at MIT (Massachusetts Institute of Technology). The MIT team was prototyping an architecture based on ‘capsules’ and studying the effects of such a system. The software platform for the architecture is Linux whereas the capsules are encoded using Java. The capsules use ‘programme language constructs’ to create a programme, which will invoke built-in primitives within the node. Some of the distinguishing characteristics of the programme/capsule method are ‘demand loading’ and ‘component caching’. The MIT system uses ‘demand loading’ to reference components built into the node, rather than issuing them as capsules. ‘Caching’ allows the use of recently accessed components without the need for re-loading and verification. All of this is designed to reduce the amount of overhead within a packet. Furthermore, the capsules have the capability to create a pre-defined state within the nodes (‘soft state’), which the subsequent packets of a particular stream can use [Ten97]. The Active Networks Project at MIT was funded through DARPA and ran from September 1996 to August 2000.

The second version of ANTS is under development at the University of Washington. The ANTS project (ver. 2) aims to design and construct a system in which clients, servers and capabilities (that are embedded in the network infrastructure) can rendezvous to provide Internet middle-ware services. The objective is achieved in three key stages:

- Investigation and development of software for self-configuring overlay networks

- Investigation and development of a network service model that accommodates a heterogeneous network in which nodes have different capabilities
 - Investigation and development of a network service model that allows different parties to combine their services [Wet98b].
-
- AIPv6 (Active IPv6) – a mechanism to merge Active Network capability with Internet Protocol version 6 (Ipv6) [Mur97]. The scheme combines the concept of Active Capsules with IP packets to provide interoperability in a network with Active and non-Active elements [Yue03].
 - The SwitchWare project is research aimed at developing a programmable switch that would allow digitally signed type-checked modules to be loaded into the node. The focus of the research group is the improvement of security on Active networks by using formal methodologies, identification of the underlying infrastructure and developing theorems. Security is supported at the programming language level (SML/NJ) without the need for high overheads, as would normally be encountered when protection is built into the operating systems of nodes [Smi97]. The SwitchWare project is a collaboration between the University of Pennsylvania and Bellcore Research Labs. The Bellcore group have defined and developed a prototype Active router that uses a small-scale multiprocessor and interconnections to an ATM network, using 10 and 100 Mbps Ethernets [Ten97]. Under the SwitchWare project several sub-projects were initiated:
 - PLAN (Programming Language for Active Networks) – devised by researchers at the University of Pennsylvania and is a development of software language constructs used to write executable Active code [Hic98].
 - ANEP (Active Network Encapsulation Protocol) – devised by researchers at the University of Pennsylvania, University of Kansas and MIT [ANE].
 - SNAP - Safe and Nimble Active Packets [Hic01].
 - SANE O/S - Secure Active Network Environment [SAN].

- ANON (Active Network Overlay Network) - In 1998 an Active Packet was sent around the world in 500 milliseconds [Tsc97].
- CANES (Components for Active Network Elements) – Georgia Tech. At the Georgia Institute of Technology researchers are applying Active network concepts to improve network congestion. The strategy is to give the applications the ability to request useful algorithms from Active nodes (e.g. lossless compression, selective discard and transcoding) during periods of congestion [Ten97] [Bha98].
- SPROCKET - Smart Packet development language [Sch99].
- MAUDE: A Wide-Spectrum Formal Language for Secure Active Networks – Stanford University [Mes99].
- JANOS: Java Based Active Network OS - University of Utah [Tul01].
- NetScript – a language designed to develop mobile agent programmes for Active Networks that can be dispatched as and when required. The software was developed at Columbia University and can be considered as the ‘third direction’ in Active Network research. The NetScript project, apart from developing a programming language, would provide the structure for Active Execution Environments (EE). The language would provide the means to script the processing of packet streams, routing, packet analysis, management functions and signalling [Ten97] [Yem96]. Netscript can be considered as another flavour to Active Networks (i.e. an Agent based approach as opposed to an Integrated approach or a Discrete approach).
- Liquid Software - the development of a suit of mobile code technologies by researchers at the University of Arizona [Har96].

- **ACTIVATE** - ACTIVE nets Test Environment by researchers at the University of Southern California, SRI International and Metanetworks Inc. [ACT].
- **ANCORS** (Adaptable Network Control and Reporting System) – a collaboration between SRI International, University of Southern California and Metanetworks Inc. [ANC].
- **ABONE** (Active Network Backbone) - a collaboration between SRI International, University of Southern California and Metanetworks Inc. [Ber00].
- **ASP** (Active Signaling Protocol) Execution Environment - collaboration between SRI International, University of Southern California and Metanetworks Inc. [Bra02].
- **Smart Packets** – a development of the BBN group who is investigating issues of programmability, data dictionaries and authentication mechanisms related to IP. Furthermore, they are seeking to improve management and diagnostic capabilities of IP systems [Ten97] [Sch00].
- **Adaptive Web Caching** - University of California, Los Angeles (UCLA) [AWC].
- **Building dynamic interoperable security architecture of Active networks** - University of Illinois [Cam00].
- **Design and demonstration of a scalable high-performance Active Network Node (ANN)** - Washington University in St. Louis [Dec99].
- **RCANE: A Resource Controlled Framework for Active Network Services** - University of Cambridge [Men99].
- **Tamanoir Execution Environment** - Claude Bernard University, Lyon [Gel00].

2.3 Summary of chapter

In this chapter we have discussed the objectives and research directions of Active Networks beginning with an initial set of targets being defined through DARPA. What is apparent is that the research has taken 3 distinct directions (i.e. integrated, discrete and agent-based) from which several sub-groups/competencies have been created to tackle problems-areas in Active Networks (e.g. the development of 'safe' programming languages, defining compatible transfer mechanisms, defining code execution environment, security and network management). The solutions proposed by research organisations may extend over several of these sub-groups and may rely on the work of others. What is also apparent is that there is not, as yet, a convergence of techniques by research establishments. Therefore any modelling and analysis techniques proposed by this thesis must be generic to be compatible with a majority of research strategies whilst satisfying Active Network objectives. The development of a generic model requires insight into Active Networks and their requirements. It does not, however, require a detailed critique of all Active Network research; hence the inclusion of only a brief review of prominent ones in this chapter. The examples given for Active Network applications are based on what is currently 'out there' as working technologies or what is proposed in the near future (the goals are set by user expectations for more cost-effective multimedia over the Internet). It can be seen that Active Networks provide a generic solution to a vast majority of proposed technologies, which can be installed and used immediately.

"I like the idea of taking network intelligence from the hardware and putting it in the packet. IP is rather passive, but it will take a lot to figure out how you bring order to something that is so democratic." - Virginia Brooks, an analyst with the Aberdeen Group, in Boston [Leo97].

From what has been discussed in this chapter, one can identify 4 possible drivers for Active Networks:

- The ability for application developers to rapidly deploy new services without the required process of standardisation between end-systems.

- The addition of services that can't be achieved without significant and fundamental changes to protocols and hardware.
- The enhancement of existing services through user customisation or with next generation Active Applications.
- The ability for network operators to replace existing 'intelligence' within the network with a flexible common technology.

The biggest issue in the deployment of Active Networks is security. Whilst there are many Active Network solutions to achieve superior network integrity and security, the inherent ability of users (and network administrators) to customise the network will generate several problems including the 'Emergence' of unusual phenomena. The large-scale deployment of such a system, which changes behaviour based on a large set of variables (pre-determined and unexpected), has management issues that require sophisticated solutions. These are difficult to solve until there is a consensus amongst the Active Network community.

C H A P T E R 3
E M E R G E N T
P R O P E R T I E S A N D
C O M P L E X S Y S T E M S

3 Emergent Properties and Complex Systems

This chapter will present:

- A detailed description of Emergence and its characteristics.
- An introduction to the concept of Complex Systems.
- A link between Emergence and Complex Systems.
- The justification that Active Networks are complex systems and therefore are likely to exhibit Emergence.

The key concepts in this chapter are: Emergence, characteristics of Emergence, characteristics of Complex Systems, Active Networks as Complex Systems

3.1 *Characteristics of Emergent Properties*

Emergence is a set of individual interactions that results in a coherent whole, which cannot be deduced from examining the properties of the individual [Bos99].

Emergent Properties are unexpected characteristics that might manifest themselves in distributed intelligent systems. Emergent Phenomena and Emergent Behaviour are essentially the same definition given to a system's behavioural anomalies that result from the above-mentioned characteristics.

The 'system' in this body of research is an Active Network. In an Active Network, for each application/task, the processing is distributed among several nodes. The nodes themselves have decision-making capabilities and are locally 'aware' of their surroundings. Thus we can define them as intelligent. Unexpected characteristics may show up as self-organised patterns, either within a small-scale (locally isolated) boundary or globally (i.e. system-wide). They may have the potential of increasing congestion, fluctuate resource usage within the nodes, fluctuate the smooth flow of packets and even lock-up the network (i.e. generally reduce the Quality of Service expected by the user).

Emergence is a theory, which describes the self-organisation of systems that form global order. This order appears to be well defined and different when compared with individual component definitions. It is also the mechanism to explain the 'survivability' of the system 'structure' in the midst of component replacement (i.e. a global behaviour is sustained even when the underlying components change throughout its lifetime). Emergence is global behaviour of systems that is 'non-deducible' from the underlying components as well as being 'irreducible' to those components [EME]. It is also dynamic and is a product of the evolution of a system (not a predetermined phenomena identified through the system characteristics).

Some of the following are generalised characteristics and examples of Emergent Properties. They are evident in a wide range of research fields and can be discovered through empirical analysis of systems. These characteristics are:

- Feedback –the circular effect

- Domino Effect
- Meta-Balance
- Survival and Sameness
- Vortex
- Resonance

3.1.1 Feedback

Feedback is a structure that flows in a loop in a system (i.e. a combination of cause-effect events that forms a re-iterative cycle). There are two types of simple feedback structures:

- Positive feedback – feedback that is self-reinforcing/self-amplifying; also known as the Snowball Effect [Dau00].

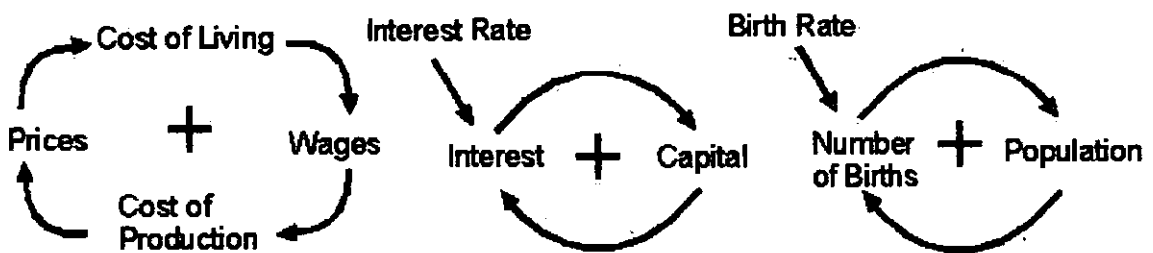


Figure 3.1.1.a: Examples of Positive Feedback

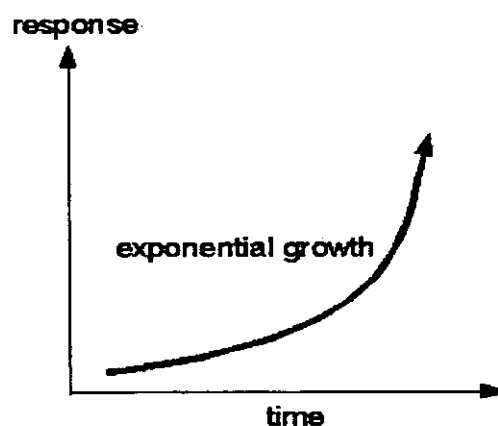


Figure 3.1.1.b: Dynamics of Positive Feedback

- Negative feedback - feedback that is self-regulating/compensating, leading to stable system behaviour [Dau00].

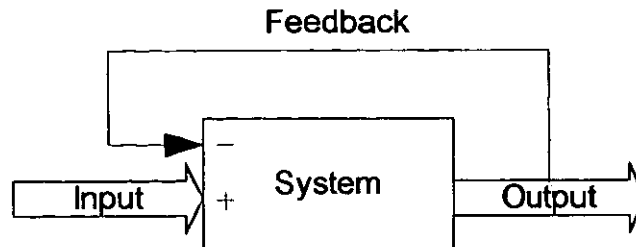


Figure 3.1.1.c: System representation of Negative Feedback

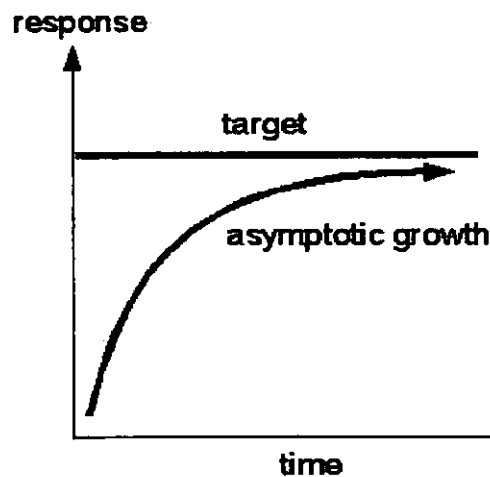


Figure 3.1.1.d: Dynamics of Negative Feedback

A system that forms feedback loops gives rise to new structures and emergent properties. Feedback is synonymous with the term 'adaptation', whereby the components have the ability to adapt to perturbations in the system, using the mechanisms of feedback.

Feedback can be viewed in two distinct ways. The first is a linear progression of the feedback cycle (i.e. we see the progress of the cycle in time - past, present and future). We can also see the cyclic movement of the outputs feeding into the inputs (not necessarily the same inputs that caused the outputs in the first place). In the linear view the 'causes' are always responsible for the 'effects'. The cyclic view has no such property. What is apparent is that both views are valid in any given phenomena and are

interrelated. “A feedback structure is a cyclic structure rolling through linear time.” [Am94].

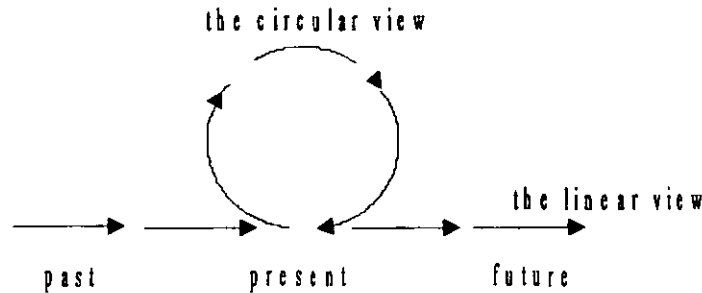


Figure 3.1.1.e: Two distinct views of Feedback

The linear view focuses on the ‘pattern’ that can be observed in the system while the circular view focuses on the ‘structure’ of the Emergent Property. The ‘structure’ is a concept. It is viewed at a higher level of abstraction using, as a guide, the interactions of the parameters in the system. Conversely, the ‘pattern’ is more concrete and is readily observable depending on the scale of the system. ‘Patterns’ are observable at lower levels of abstraction [Am94].

“Situations with observable ‘structures’ are indicative of Emergent Properties”

Systems theory states that both views of ‘structure’ and ‘pattern’ are valid for the analysis of a system, albeit being fundamentally different. The system’s structure is only apparent when (as mentioned above) viewed at a higher level of abstraction, which translates, in practical terms, to a model depicting the system with coarse detail. The same system must be modelled in fine detail at a lower level of abstraction in order to view the patterns [Am94].

3.1.2 Domino Effect (positive feedback)

Also known as a wave pattern, the Domino Effect originated from the way that dominoes fall creating a wave formation. In this scheme, there exists a circular ‘structure’ as well as the wave-like ‘pattern’. Whilst the underlying mechanism for the

self-sustained behaviour is positive feedback, the defining characteristic of the Domino Effect is the advancing wave front [Am94].

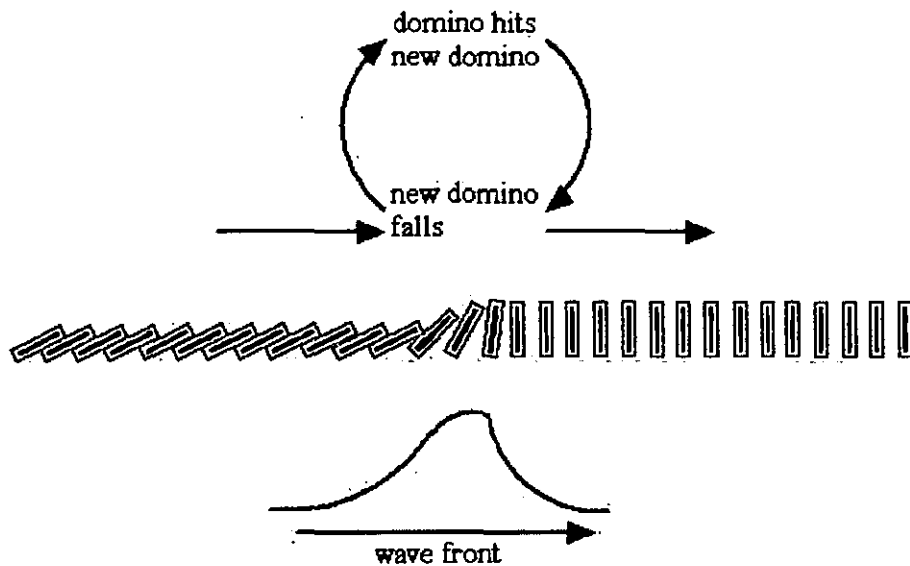


Figure 3.1.2.a: The Domino Effect

3.1.3 Meta-Balance

Complex Systems exhibit Emergent Phenomena as ordered structure from disorganised behaviour. In order for Emergence (i.e. a stable structure) to appear at higher levels of abstraction (as mentioned-above), the lower-level abstract model must be 'out-of-balance'. Thus Meta-Balance is the term given to the seemingly 'stable' structure at higher levels. Emergent Behaviour (or global order) only occurs in a system when it is pushed out of balance. Both the Snowball Effect and the Domino Effect are examples of systems in Meta-Balance [Am94].

3.1.4 Survival and Sameness

Survival and Sameness refers to the 'structure' of phenomena within a system. A feedback loop, for example, stays the same throughout its existence (structurally) even though new system components continuously replace the old ones (e.g. in order for the

wave front to continue, in the Domino system, it must be continuously fed with new dominoes). The replacement process keeps the structure 'alive' [Am94].

3.1.5 Vortex

A Vortex is generated from within the system. It is an active force that binds the system or a section of the system to an organised existence (an existence not recognised in the ordinary sense). The 'hyper-existence' of a Vortex has the following basic characteristics:

- The Vortex must be EMBODIED
- The components of the system need to be out of balance
- There must be feedback in the system
- A Vortex cannot be analysed by 'reductionism' (i.e. the analysis of individual components) [Am94].

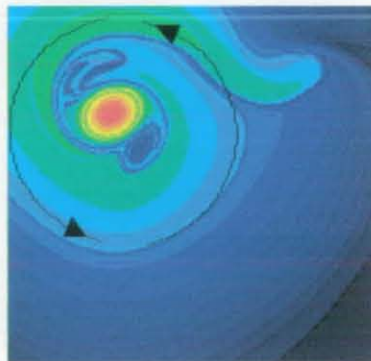


Figure 3.1.5.a: Computer representation of a Vortex

The difference between a Vortex and any other circular structure is that an active force is present at the centre of it holding the system in that structure (e.g. a tornado is a naturally occurring vortex. It appears to have a force at the centre sucking great masses towards it. This, however, is an illusion created by the circulating masses). Where there is the possibility of several Vortices, the system will be sucked into the strongest/closest one, and will remain there until perturbed by some force taking it to the next Vortex [Am94].

3.1.6 Resonance

Resonance is a repeating process, much like feedback, but with the added characteristic of 'information reduction'. The difference between Resonance and a Vortex is that a Vortex is an active force that a system will be sucked into, whilst Resonance is a simplified structure in which the system will be trapped.

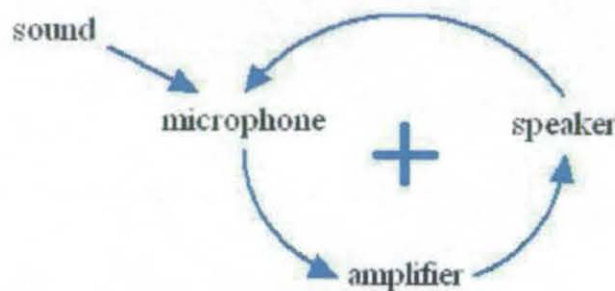


Figure 3.1.6.a: Example of a Resonant system

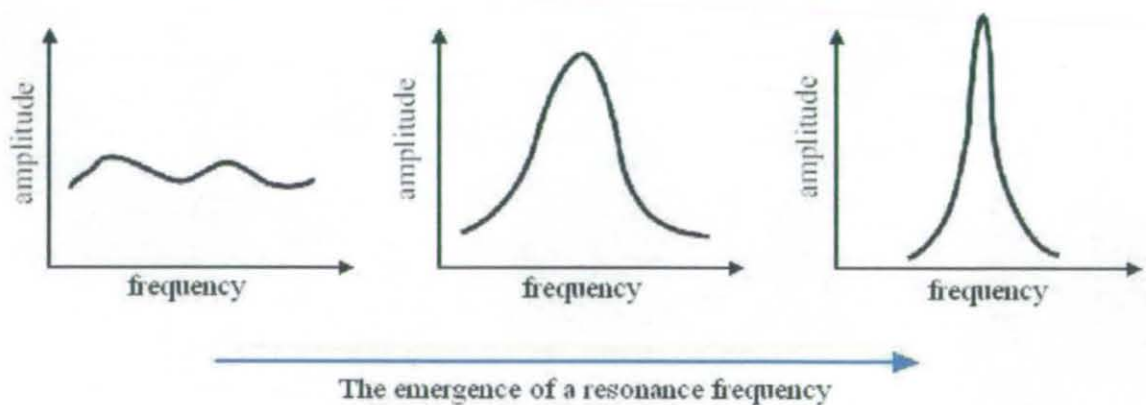


Figure 3.1.6.b: Resonance

Information reduction allows the Resonance to behave as an Emergent filter (i.e. limit the behavioural characteristics of the system to the boundaries of the resonant behaviour). With a Complex System, there may be millions of individual components behaving independently of each other to satisfy their individual objectives. Every single component is an undetermined variable or group of variables. This does not mean,

however, that any Emergent Property contains millions of parameters and detail. Most often, a Complex System will produce simple patterns in which it will be trapped. This would imply that an Emergent structure, which functions as a resonant behaviour, is also well defined [Am94].

3.2 Emergence Research

What is perhaps surprising is that known circular patterns in nature, physics, etc. exist that can be readily extended to other fields. For example, the characteristics of waves are near identical in the perspective of Radio waves and Ocean waves. The same food-chain structures emerge in widely different Eco-systems [Am94].

Emergent phenomena may appear differently in different systems, however, they share commonality through the above-mentioned characteristics.

Not all Emergent Properties are detrimental to the system. 'Emergent Computing' is one area where designed Emergent Behaviour adds value to the services offered by a Network [Bus01].

The following diagram depicts, succinctly, the broad range of research topics associated with Emergence.

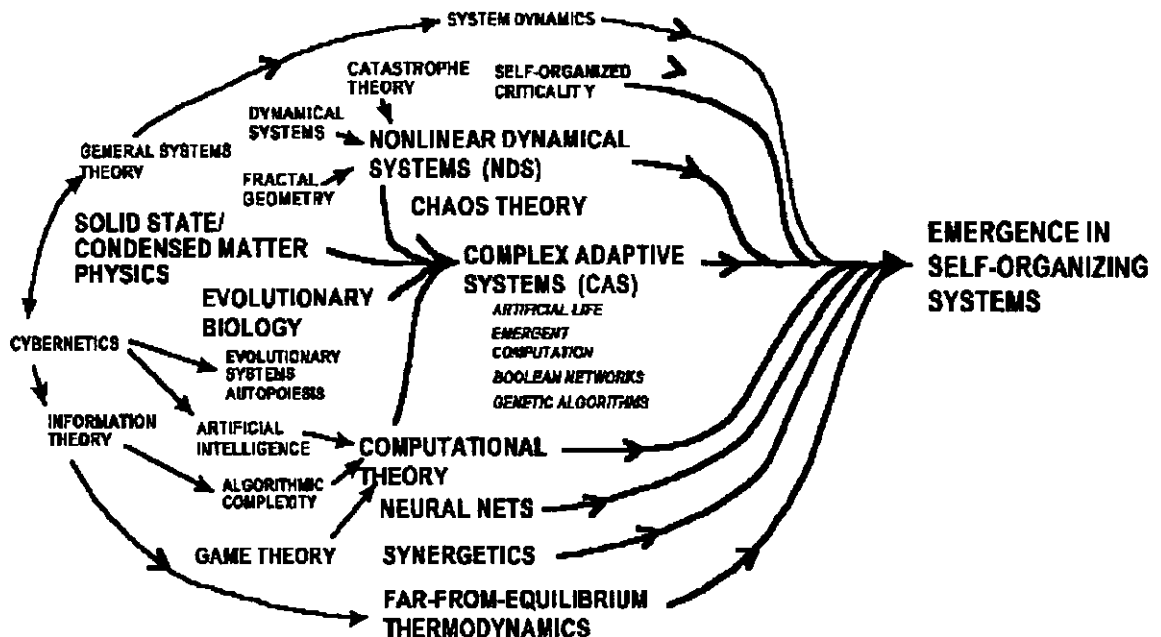


Figure 3.2.a: Mathematical and scientific roots of emergence (Jeffery Goldstein)

As can be seen from Figure 3.2.a the most focused field of study that links through to Emergence is Complex Adaptive Systems. The theory of 'Complexity' plays host to a large number of scientific and mathematical fields, each striving to identify the

characteristics of Emergence, thus acknowledging a link between Complexity Theory and Emergence Theory [Gol99].

3.3 *Characteristics of Complex Systems*

The definitions of ‘Complex Theory’, ‘Systems Theory’ (a relatively established field) and ‘Chaos Theory’ (less established in the way of applicable results) are closely interrelated, thus making any distinction between them difficult and only valid when considering a particular system (as is the case in this work). In a general sense, Systems Theory covers simple-systems exhibiting simple behaviour. Chaos Theory studies the ability of systems (simple or complex) to produce complex/chaotic (i.e. unpredictable) behaviour over long periods [Cru90]. It forms one part of Complex Theory (‘Complexity’). In contrast, the section crucial to Emergence is the study of Complex Systems with overall ‘simple’ behaviour. Complexity of this form is the ‘middle ground’ between ordered systems and Chaos [Cru02].

It is easy to understand why the author and others consider Complexity to form the obvious choice for the analysis of Active Networks. Active Networks (and any other large-scale network such as the Internet) have, or propose to have, millions of heterogeneous nodes, which would be impossible to analyse in terms of simple feedback cycles (a key sign of Emergence in systems modelled under Systems Theory).

The global structure of the system comprises of many local interactions between individual components. The global Emergent Behaviour resulting from the interactions exerts influence over the behaviour of the individual component in a circular manner. This keeps the system in that particular Emergent structure, whether it be a Vortex or Resonance [Am94].

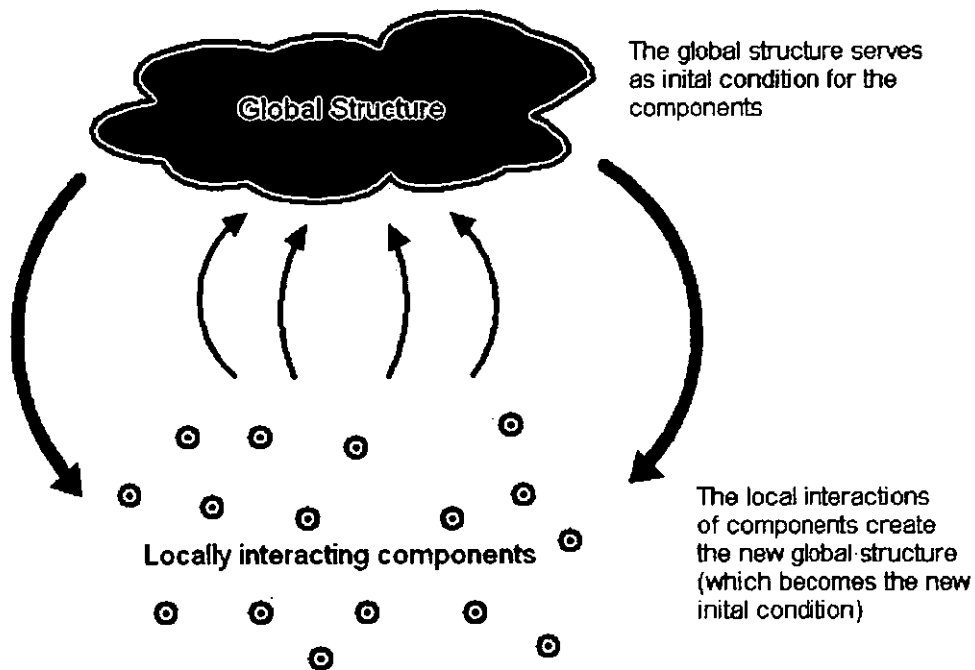


Figure 3.3.a: The Global Emergence in Complex Systems

A Complex System has:

- Numerous independent components
- Components that interact locally and numerously
- Overall global behaviour that is independent of the internal configuration of components
- Overall system behaviour that is well defined
- Evolutionary Behaviour

3.3.1 A Complex System consists of numerous independent components

An important feature of individual components is that they exist independently of each other and their behaviour is due to independent decision-making. The components are 'wholes' capable of existing on their own. Thus a Complex System is a whole built up of wholes.

This is partly true in the case of an Active Node, which can exist by itself but is functionless without the rest of the network [Am94]. However, the 'Active' element (i.e. the intelligence) of such a node may meet this criterion.

3.3.2 Components interact locally and the interactions are numerous

In order for Emergent Properties to exist in amounts that can be observable, there should be numerous local interactions. In complex communication networks this is accomplished by having a varied number of packet types, simultaneous packet flows, multiple routes to destinations and a general high interconnectedness.

In networks such as the Internet, direct node-node interactions are confined to 'local space'. Any other connection (e.g. source-destination connection) is achieved through the global structure and not by a direct link (i.e. a virtual connection is made using several nodes, traversing several sections of 'local space' within the network and with several possible routes).

High connectivity is also likely to exist in Active Networks. Emergent Behaviour would result if an event (anomalous or otherwise), created in local space, propagates to others as a ripple (a Domino effect). The ripple is facilitated and amplified through the high interconnectedness of the network and 'piggybacks' on the normal source-destination communication process [Am94].

3.3.3 Overall global behaviour is independent of the internal structure of the components

Similar Emergent Behaviours can be observed in a system that is independent of the processes that would be involved in achieving them. It is also possible that completely different systems will exhibit the same Emergent Property (e.g. waves) [Am94].

3.3.4 Overall behaviour of the system is well defined

Viewing the Emergent Phenomena by itself, disregarding the individual components and much of the low-level detail, it is possible to note that the global structure conforms to simple rules (possibly mathematical rules) and behaves in an exact manner [Am94].

3.3.5 Evolution in Complex Systems

Evolution in Complex Systems is based around Darwinian concepts. Complex Systems have the ability to apply 'Natural Selection' to processes (i.e. "survival of the fittest"). Darwin viewed organisms as perpetual machines (i.e. staying alive long enough to make a copy of themselves and die). Complex Systems, however, are more involved but components within the system do possess the Darwinian characteristics of surviving for the completion of their objectives. Any processes that fail to complete will be weeded out of the system (a simple filtering process), and an evolutionary landscape will develop as the system progresses. Complex Systems also have an interesting feature of competition among components for limited resources. This makes the evolutionary landscape dynamic and adds another dimension to the fitness criteria and objectives of components. This breeds new 'intelligence' and 'creativity' and exposes the system to Emergent Phenomena [Am94].

3.4 Active Networks as Complex Systems

In order to analyse Active Networks it is helpful to classify the system in broad terms so as to focus further research. The following text will seek to show that Active Networks fall into the category of Complex Systems with which Emergent Properties are commonly associated.

The criteria mentioned in the previous section apply to large-scale computer networks (Active or otherwise). Networks of an Active nature will add another dimension to the complexity because of their inherent programming capabilities.

The idea that Active Networks will exhibit Emergent Behaviour is further strengthened by the following factors (classified by the author using available research on Active Networks):

- Distributed processing architecture
- In-built intelligence and self-awareness
- Local network awareness/feedback loops
- Lack of central management control
- Application level organisation
- Adaptation and evolution
- Memory
- Limited resources and competition

3.4.1 Distributed Processing Architecture

Each Active Node would have the capability to process a unique/distinct task or a particular portion of an entire application - Active Applications may acquire several nodes on the source-destination path to process a particular task (e.g. for a reliable video stream). Thus the application acts as task manager; splitting, replicating and distributing programme components to various Active Nodes. Since a single node might be unaware of the full application programme/functionality, it will not have a sense of the final

outcome. Thus any Emergent Behaviour that can result, globally, cannot be foreseen or prevented by any one node.

3.4.2 In-built Intelligence and Self Awareness

All Active Nodes will have a processing core along with memory to store and run programmes. Furthermore, they have the capability to identify the processing required on a packet, based on the current local network conditions (e.g. network congestion), requirements of the packet and its own resource status. Intelligence and autonomy are strong contributors to Emergent Behaviour. In addition, each node would be likely to have self-diagnostic capabilities.

3.4.3 Local Network Awareness

It is proposed that an Active Node has the means to gather information about its surrounding nodes and links (e.g. resource usage, size of input/output queues, link congestion). Local network 'awareness' will manifest itself as information feeding back to Active Nodes through Active Packets (as a primary or secondary objective of an Active Packet). Active Nodes would then act on that information according to pre-set rules and procedures. As opposed to 'Intelligent Networks', Active Networks do not possess nodes that are capable of initiating intelligent processes. In practical terms, Active Packets/Applications form the creative force behind node 'awareness', which is highly dependent on the requirements of the application. Irrespective of the underlying mechanism, an Active Node will have the capability to store information and become 'aware' of its surroundings. It can take actions, based on this perception of the network, in the servicing of applications. A feature of this would be the formation of local feedback loops (affected by local network conditions), which would contribute to Emergent Behaviour [Am94].

3.4.4 Lack of Central Management Control

Control of Active Nodes is dependent on their internal instructions and on the code carried within Active Packets. The autonomous behaviour of individual nodes and the

lack of knowledge of the final outcomes will result in nodes behaving normally, in local space, whilst contributing to global Emergence.

3.4.5 Application Level Organisation

Even with the presence of autonomous behaviour within the nodes, groups of them could essentially co-operate on a level that corresponds to a specific application (i.e. Active Nodes may organise along the packet source-destination path to provide better Quality-of-Service). This type of co-operation is not visible to the individual nodes, but is visible to the end-users as improved service. The ability of nodes to organise, without being 'aware' of the fact, is an Emergent Property, irrespective of the existence of a controlling element (i.e. the application). The focus of this thesis is the **unexpected** collective self-organisation of nodes. In such a situation applications would still form as expected, however, the influencing factors are such that the combination is potentially detrimental and may not have co-operative behaviour.

3.4.6 Adaptation and Evolution

Active Networks can be defined as adaptable and evolutionary networks. This can be perceived in several ways. For example, the network space is dynamic and changing in terms of the addition and removal of nodes; a failure in sections of the system results in redundant components taking over. Active Nodes, with their local 'awareness', can easily adapt to the changes in the structure of the network. Further to this, an Active Node would have the capability to adapt the processing of packets depending on the network state (e.g. network congestion, node resources, etc.).

"The (Active network) programming abstraction provides a powerful platform for user-driven customisation of the infrastructure, allowing new services to be deployed at a faster pace than can be sustained by vendor-driven standardisation processes."
[Ten96].

The discrete approach of Active Networks achieve customisation through ‘plug-in extensibility’, which is a technique for the loading of code that is pre-defined and is an abstract prediction of future needs. This may not be sufficient in establishing a true evolutionary network, and in the future we may see the extension of Active Nodes through the dynamic loading of code, which is customised and updated on demand [Hic00]. Evolution would add a further element of unpredictability to the behaviour of nodes and would contribute to Emergent structures not envisaged originally.

3.4.7 Memory

As an Active Node is aware of its surroundings, itself and the application passing through it, it has the potential to retain details within node memory (e.g. previous packet type, packet number processed in a stream, congestion states, queue lengths, etc.). Any details retained from previous actions and events constitute ‘memory’, and will contribute to Emergence within the network. Furthermore, an Active Packet may be able to reserve node resources in order to service further packets (from a single application) traversing the node.

3.4.8 Limited Resources and Competition

The Internet has limits to the size and capabilities of resources (i.e. link bandwidth, node processing power, memory, etc.). It is possible that an Active Application will contain resource-usage maximisation algorithms. Applications using Active technology will compete for resources within the network based on a scheme of apportionment and priority. This competition will lead to a dynamic, evolutionary landscape (as mentioned previously), thereby affecting the ‘fitness’ of one application over another. Emergent Properties are likely to form as a result of this additional complexity.

3.5 *Summary of Chapter*

This chapter attempts to highlight the natural link between Emergent Phenomena and Complexity Theory by citing several exemplary points and references. It is proposed that the key features that would naturally manifest in Complex Systems are, in fact, Patterns and Behaviours of Emergence. This reduces further research work to the conceptual understanding of Complex Systems and key results. The chapter goes forward to explain why Active Networks should be considered as Complex Systems and therefore be a perfect candidate for modelling (for the purpose of observing Emergent Behaviour).

As can be seen from Figure 3.2.a, Emergence research is varied and highly topical. Different research institutions delve into Emergence analysis through different area of expertise and different modelling paradigms. It is not the intention of this chapter to provide a detailed description of individual Emergence research projects. However, this chapter attempts to extract some commonality by focusing on generic Emergent characteristics that might be encountered in research work.

Instances that can be identified as Emergence are varied (possibly infinite in number) and this chapter may not have included all forms. The examples given are distinct pattern manifestations within a particular system. These patterns are useful for global system analysis because they are observable and provide symptomatic evidence of system anomalies. Much of the focus for the Emergence detection process, introduced by this research (and by other research projects), is based on observing patterns in various system dimensions (i.e. various observable criteria) [Kul01].

The discovery of a system and its capabilities is possible through a broad range of analytical techniques and competencies. At one end, systems can be understood through the analysis of contributing components and localised interactions. At the other extreme, localised components, properties and interactions are insufficient to explain global behaviour. Most systems lie between these two extremes and require a balanced approach to the modelling process in order to comprehend system dynamics.

Emergence is sometimes seen as an important transition device for the heuristic explanation of system behaviour, until the knowledge of such a system is complete and the laws/principles governing anomalous behaviour are fully extracted. This view is questioned by several researchers who believe that Emergence is a unique facet of

complex system behaviour. System non-linearity would result in an unending 'Emergence of Emergents' [Gol99]. Therefore, a full comprehension of a system that includes all anomalous behaviour may not be possible.

It is worth noting that irrespective of the arguments, the importance of identifying Emergence is not diminished.

C H A P T E R 4
M O D E L L I N G A N D
S I M U L A T I O N

4 Modelling and Simulation

This chapter will present:

- A description of the strategy employed for the modelling and simulation of an Active Network.
- A detailed description of the high-level abstract Active Network model, including all assumptions made during the development process.
- A description of the Active Applications used to develop the model and the process by which they were incorporated – the development of core functional processes also known as Primitive Functional Components (PFCs).

The key concepts in this chapter are: high-level abstract model, Active Applications, modelling assumptions and features, PFCs

4.1 *Simulation/Modelling Strategy*

The current trend in research into Active Networks has been to define and implement several different topologies and communication protocols, independently or collaboratively, by various institutions. While this explores, in detail, all possible scenarios and manifestations of Active Networks, there exists little in the way of consensus and standardisation. Thus, the variation in Active Networks currently being considered (and to be considered in the future) presents itself as a problem when choosing a suitable network with established standards. In contrast, the current Internet uses TCP/IP, which is the established protocol, and hence the research work into various aspects of performance analysis is numerous and developed. To circumvent this problem it was decided that a suitable **high-level low-detail** model/simulation be created, in the hopes that it can be used in an efficient manner to discover Emergent Behaviour (albeit a unique example of Emergent Behaviour).

The basis of the high-level simulation is the development of a generic model. This model would contain little detail of the manner by which the communication process occurs (i.e. Active Packet transport mechanism) and of the specific network configuration. The lack of detail is advantageous since the resultant simulations serve as 'indicators' of network behaviour.

Developing a generic model serves several needs:

- Provides a template for future detailed modelling of Active Networks
- Summarily proves the existence of Emergence and Emergent Properties
- Locates possible trouble spots and gives direction to the future detailed investigation of an Emergent Property
- Avoids analysing results obtained from large complicated models of Active Networks, without an idea of the Emergent Behaviour present within it
- Provides a systematic approach to the problem
- Provides a framework for the incorporation of future Active Applications and viewing of their consequences on the network

- Acquire adequate generic behaviour to universally represent various Active Network research paradigms

The modelling strategy, created by the author, involves the discovery of a range of Active Applications that encompass the entire capability of an Active Network. Once these applications are noted, they are broken-down into 'Primitive Functional Components' (PFCs). An Active Node can be modelled using these primitive components (and not much more), whilst retaining node functionality allied to the servicing of applications. What are important are the functional qualities of these components and not their detailed execution processes within each node (which can be disregarded).

The advantages of this approach are:

- Top-down approach – can be started from simple user requirements of Active Applications
- All applications are a combination of one or more primitive operations/components
- Can incorporate future applications
- Independent of any specific technologies or protocols
- Suitable simulators are assessed and implemented quickly

4.2 Static-Node simulation

The static node simulation scenario used in this project comprises Active Nodes being placed in a pre-configured pattern to form a network. This method offers a straightforward approach and closely adheres to the application of Primitive Functional Components in generic modelling. An Active Node is depicted as an element pre-programmed with all the functional components, which are individually activated depending on the type of packet that passes through it. Note that in a high-level abstract simulation the details pertaining to the method of distribution and execution of the primitive programme components (i.e. as a part of the Active Packet stream or pre-loaded into Active Nodes) is inconsequential.

The overall network topology is based on the Ohira-Sawatari deterministic model, [Ohi98] which was used to describe Emergent Network Traffic dynamics [Sol01].

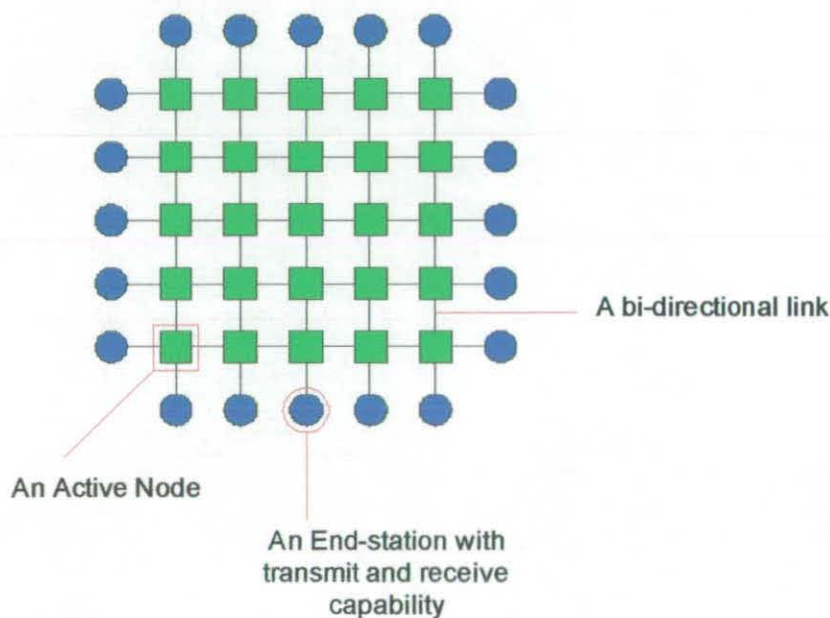


Figure 4.2.a: Ohira-Sawatari model adapted to Active Networks

The network topology is a simple, scalable representation of the Internet. It provides multiple connections from end-station to end-station (i.e. multiple paths for packets to take when travelling from source to destination). Whilst not technically an accurate

representation, it provides a comparable structure to the Internet (with Active Nodes) when viewed at a high-level abstraction. It is expected that the high degree of connectivity would give rise to Emergence; in particular as structures formed by feedback.

4.3 *Defining Characteristics of the Active Networks model*

General observations on Active Networks were made as a pre-cursor to the high-level model-building task. The following list details these observations, which function as bounds for the modelling paradigm. The list consists of known truths about Active Networks, assumptions made of the model and the justifications for these assumptions. The list also mentions the modelling considerations made with respect to there being Emergent Behaviour within the system.

4.3.1 There are two fundamental types of Active Networks which are being considered by this thesis:

- Capsules - networks with Active packets that carry programme code ('capsules') to be executed in Active nodes.
- Programmable Switches - networks with Active packets that carry 'trigger bits', which call and execute pre-loaded code within Active nodes.

The generic high-level model will inherently cater for both schemes. The features of the generic design will consider only the type of code being executed in a node, at a given time and location, irrespective of how this code-block came into use.

4.3.2 All possible Active services and functions can be broken-down into Primitive Functional Components. The generic high-level model will seek to model Active Applications as these components or as combinations of these components - a substitute to modelling Active Applications with formal definitions. This gives independence from end-user requirements, applications and services.

4.3.3 All Active Packets will be typed based on the primitive functional components they invoke.

4.3.4 Packets have a limited lifetime. Hence, a sustained pattern within the network has an abstract view, and involves several Active Packets/streams during its

lifetime (i.e. the abstract nature allows it to maintain structure and travel throughout the network, however the underlying Active Packets may not remain the same).

- 4.3.5 All 'intelligence' and decision-making activities will be focused within Active Nodes.
- 4.3.6 Packets are responsible for the dynamic progression of the network and the development of Emergent Phenomena.
- 4.3.7 Packets do not interact with each other directly. Indirect interaction is only possible through an Active Node.
- 4.3.8 The movements of the single packet are deterministic. Several factors influence the path of an Active Packet. These are:
 - Source and Destination end-stations; as applicable with standard routing of packets within a network.
 - Congestion on the outward link.
 - Lack of proper resources at current node – the node would re-route to another containing the proper code modules and/or adequate resources to process the Active Packet.
 - Packets dropped due to the expiry of a 'Time-to-live' counter.
 - Last node's address – not allowing the packets to reverse course without being modified by an Active process.
- 4.3.9 The simulation process would initially consider a network made solely of Active Nodes, as it would ease the development of the simulation environment. Emergent Behaviour would be in its purest form. It is assumed that Emergent Behaviour is just as likely to occur in hybrid systems containing Active Nodes and normal routers. Emergent Phenomena are independent of the underlying nodes (i.e. the same 'structures' can emerge from different network

configurations), are capable of traversing the network and can encompass small or large sections of the network without the loss of their characteristic 'structure'. It is therefore feasible to assume that hybrid systems are equally capable of developing the same Emergent Behaviour. However, the Emergent Phenomena indigenous to hybrid systems are not considered for this research.

- 4.3.10 For large-scale simulations of a particular network, the packet flows can be considered as random. It is assumed that for a high-level abstract simulation the routing information of individual packets is less important when trying to view large networks globally. Thus, only the functional capabilities need be included in the packets. The random path of packets is based on the analogy drawn between Active Networks and Complexity (simple patterns and structures occur, as Emergence, in complex and highly random systems).
- 4.3.11 An Active Node has limited resources (i.e. limited input/output queues, memory, processing capacity and the maximum number of concurrent processes). Furthermore, the node may limit the amount of resources allocated to each process depending on the dynamic conditions at the time of allocation.
- 4.3.12 Active Packets, depending on the application, can retain resources in the current node as it passes through (the concept being referred to as 'history', 'imprint', 'memory' and 'trace'). This process is characteristic of lead-packets reserving and conditioning part of the node to process follower-packets (all part of a homogeneous stream). An Active Packet can have an 'imprint' on a node depending on:
- The resources it calls for
 - The time limit for the resource allocation
- 4.3.13 For the purpose of modelling Active functionality with ease, an Active Packet is considered as an entire Active Application session, consisting of a combination of Primitive Functional Components. One can think of an Active Packet as an

application stream (e.g. an Active video/audio stream). For the purpose of this project, the high level model combines all actual packets of an application session into a high-level abstract 'Active Packet' without loss of functionality.

- 4.3.14 Each End-station injects one Active packet into the network for each simulation run (i.e. the analysis is of the simulation of a 'single shot' of Active Packets interacting to form global patterns and effects). Note that an Active Packet is a complete application. Thus the model simulation considers the interaction of only one set of applications simultaneously injected into the network. This simplifies the simulation and reduces the number of possible factors influencing a possible Emergent effect. The data analysis would therefore be able to offer a clearer understanding of the underlying permutations that caused such an effect.
- 4.3.15 In order to accurately depict resource usage fluctuations the Active Network model contains 3 resource types: "MEMORY", "PROCESSOR" and "BUFFER". An Active Packet requires all 3 types of resources. MEMORY and PROCESSOR resources are fairly obvious needs of an Active Packet. BUFFER resource is the term given for the Active Packet's input queue resource requirement.

4.4 Defining Generic Model Applications and Primitive Functional Components

4.4.1 Possible Active Applications

As part of the modelling strategy mentioned above, this work attempts to define a set of primitives as the foundations for the development of Active Network model. The initial step in this process is the collection and analysis of a representative sample of Active Network Applications (both predicted and/or implemented). These applications are listed below:

- Reliable Scalable Multicast - a group of Active switches and routers maintain a set of TCP connections for reliable data replication. Also involved in the process is a data caching element within each Active node.
- Video and Audio Transcoding [Mar99] - a digital signal of one standard is converted into another by an Active device.
- Merging of multiple remote sensor data – used in telemetry applications where a single Active Node manages multiple data sources [Ten97].
- Storage of status information – for applications such as distributed network games.
- Dynamic generation of web pages [Ten97] – an Active node capable of storing and executing programmes that generate web pages dynamically and on demand.
- Dynamic distributed caching [Ten97] – web cache servers in an Active Network that can be dynamically repositioned.
- Distributed network control [Mar99] [Raz00] [Pso99] – e.g. supporting optimised routing algorithms.
- Quality of Service (QoS) filtering for multimedia streams – e.g. consider a source feeding a single stream of multimedia to multiple heterogeneous receivers. The streams run through several routers and switches in a hop-by-hop manner. The QoS feedback commands generated at each node, at each hop, can burden the source with considerable processing. Using Active Nodes along the return feedback path to

merge multiple QoS feedback can reduce bandwidth usage. It may also be possible for Active Nodes to process the feedback and take decisions autonomously.

- Support for application aware Anycast – a source needs only to contact the nearest Anycast group [Kat00].
- Application aware local link FEC (Forward Error Correcting Code) implementations [Sto00] – Active Nodes sensing communication links with poor performance can provide additional error correction bits for packets traversing those links. This is more efficient than end-to-end error correction since the packets have additional overhead only on the required links.

4.4.2 Primitive Functional Components

The representative sample of Active Applications was systematically decomposed into a set of core functions. As it happens, it was possible to identify these core functions independent of any user data, source/destination values and the type of service provided. These Primitive Functional Components (PFCs) extracted from the above-mentioned applications are:

- Data Replication
- Data Fusion
- Data Generation
- Data Transformation
- Global State Maintenance
- Network Control Processing

4.5 *Mathematical Solutions verses Simulation*

Some research suggests that it may be possible to characterise the Internet or any other large-scale communication network using mathematical techniques. In fact, there exists a significant body of research where mathematical techniques have been used to model network dynamics. For example, an approach is adapted from fluid flow models and use Stochastic Differential Equations to describe the behaviour of packet flows and node input/output queues. Ordinary Differential Equations are obtained from the Stochastic set, which then can be solved numerically [Yua02].

As an alternative, there exists research effort to model large-scale networks with discrete-event simulations. The advantages of simulation over mathematical approaches are:

- They are capable of capturing significant detail and behavioural effects of a network (in finer granularity when compared with mathematical models) [Yua02].
- It is likely that a simulation model will exist for a particular network (with specific protocols and mechanisms), rather than a mathematical one [Yua02].
- Ability to capture complex behaviours and to view global behaviours with relative ease.
- A large quantity of parameters can be accepted.

The disadvantages are:

- The execution of discrete-event simulations is CPU intensive and may be limited by the hardware and software requirements of the simulator [Yua02].
- In computer-based simulations parallel and distributed flows of information are handled concurrently, which is a pseudo-parallel technique. Although this works well, it is not parallelism in the truest sense.
- The simulator is capable of processing a number of concurrent tasks. This is achieved by means of discrete time steps. True continuous simulation is not possible. All tasks, which are intended to be run on the simulated network, are broken into events. The progression of the simulation is broken into discrete time

steps, and all events pending at each time step are executed together. The order of execution, within a single step, is irrelevant since it is considered as an instance in real time. Time progression (from which the progression of the simulation is achieved) is equivalent to the progression of time steps. The event and time step method is commonly used in many pseudo real-time simulators to achieve parallelism and multiple task execution.

- In order to get the truest possible picture, the modeller would incorporate as many parameters as possible. However, there is a risk that unnecessary detail would have to be included even when the modeller is only interested in a few (an over complex model would demand more hardware resources and more time to complete). Mathematical models, if found, would only contain the key parameters for an accurate analysis.
- There is a significant problem in obtaining, analysing and comprehending the results of discrete-event simulations. In order to gather results, probes may be inserted into the simulation, and the insertion points are selected based on the modeller's intuitive grasp of the system under simulation. A large collection of probes could have an impact on an accurate simulation (e.g. hardware and software limitations). Discrete-event simulations also tend to generate large data sets.
- In order to understand the cause behind an 'observed' Emergent Phenomenon, one might need to speculate and re-animate the events in the exact manner in which they occurred. Since it is possible to arrive at the same defined Emergent Phenomena through several different processes and interactions, the re-animation process must be carefully executed, if at all possible. In contrast, mathematical models themselves offer explanations to various observed phenomena.

There are considerable advantages in employing mathematical models, but the initial steps are often difficult and the result produced maybe intractable. Discrete-event simulations are easier to handle, but obtaining comprehensible results often proves to be difficult. As can be seen, the direction taken by this research was to apply a discrete-event simulation technique that follows on from the development of the specifications of

the generic model. This was applicable in this situation as it provided the best possible means to detect an unforeseen anomaly.

4.6 Summary

This chapter has outlined the modelling strategy implemented in this work along with the considerations and assumptions made in the development process to facilitate the manifestation and detection of Emergence. The model is a high level abstract depiction of the Internet, which exclusively displays Active Nodes and their functionality. It is proposed that for the purposes of Emergence detection this model is adequate and would cater for most Active Network schemes currently being researched. The generic low-detail visualisation of an Active Network within the Internet is a concept believed to be unique to this work and forms the primary analysis of the system.

The chapter goes on to describe Active functionality/applications being reduced to primitives in order to simplify the modelling process. It is proposed that these primitives form the core building blocks of any Active Application to a satisfactory level thereby preserving universal compatibility.

The abstract view of the Active Network model based on the lattice structure devised by Ohira-Sawatari [Ohi98] is conceptually similar to the visualisation of the Internet (i.e. servers/clients at the edge of the system with routers/switches localised in the core).

The chapter also describes the approach to the detection of Emergence as the detection of anomalous patterns within the Active Network core. To this end a single set of Active Applications are fed into the core by the end-stations. The set is allowed to propagate, interact with each other and affect the Active Nodes. Through these interactions valuable information is gained with the aim of detecting Emergence within the system.

Creation of a simulation environment (i.e. a simulator) or modification of an existing network simulation environment is a time consuming task. It involves the design of tailor-made Active Nodes (programmed with Primitive Functional Components), custom Active Packets, distribution mechanisms, simulation dynamics, displays, results loggers and other components.

The task is made easier if the simulation dynamics followed a universal methodology (not necessarily network related), and simulators existed that would provide the functionality to cater for these universal properties. 'Petri-Nets' are a universal theory that is capable of modelling distributed systems and parallel event-driven networks. It is

an established theory with defined mathematical formulae to aid the analysis of a network.

C H A P T E R 5
P E T R I - N E T
S I M U L A T I O N

5 Petri-Net Simulation

This chapter will present:

- A detailed description of Petri-Net segments specifically used for the simulation of the Active Network Model.
- The suitability of Petri-Nets in simulating Active Network functionality.
- Petri-Net flow diagrams of the Active Network model including descriptions of packet flows, data processes, resource usage, peripheral control mechanisms and data logging.
- A description of what constitutes an Active Packet/Application in this model.

The key concepts in this chapter are: Petri-Nets, Petri-Net extensions, simulation of an Active Network, Design/CPN Petri-Net simulator, Resource Usage data, Ohira-Sawatari lattice structure, Active Network model hierarchy and components – Merge Packet component, Replicate Packet component, Direction Solver component, Local Storage component

A Petri-Net is an abstract, formal model of information flow [Cha]. The concept was introduced by Carl Adam Petri in 1962 for the computational analysis of concurrent systems. The properties, concepts and techniques of Petri-Nets are designed for the simple yet powerful modelling of systems with parallel asynchronous information flows and activities.

Petri-Net graphs contain four types of elements: circles (called Places), rectangles (called Transitions), markings (called Tokens) and Directed arcs. Directed arcs make connections from Places to Transitions and from Transitions to Places. A Petri-Net is a multi-graph since it can allow multiple arcs from one Place to several Transitions, and vice versa [Pat81].

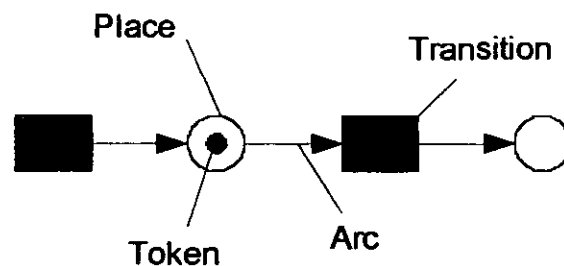


Figure 5.a: Petri-Net terminology

The net execution process consists of moving/placing Tokens from input Places, through Transitions, to output Places. The Places represent passive system components. They act as storage for the Tokens, take particular states and generally make things observable. Transitions represent active system components and function as process blocks to produce, transport and change Tokens [Rei85]. The distribution of Tokens in a 'marked' Petri-Net, at any given moment, defines the state of the net.

In order for a specific Token to move from one Place to another it requires the intermediary Transition to be 'enabled'; and then the Token only moves when the said Transition 'fires' (i.e. the execution of the Transition).

There are two rules, which govern this particular movement of Tokens within the net:

- **Enabling Rule** - a Transition is enabled if every input Place, attached to the transition, contains at least one Token.

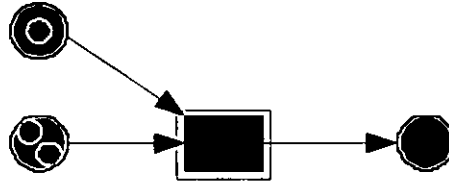


Figure 5.b: The Transition with the red border is 'enabled'

- **Firing Rule** - firing an enabled Transition removes one Token from each input Place of the Transition, and adds one Token to each output Place of the Transition (i.e. generates new Tokens) [Her97].

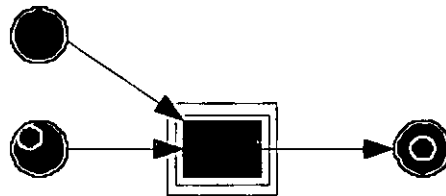


Figure 5.c: The Transition with the double red border has 'fired'

This simple mechanism of 'enabling' and 'firing' in Petri-Nets can be used to model complex interactions based around a set of fundamental structures (detailed in the proceeding section), which form the building blocks of any system.

5.1 Modelling set scenarios with Petri-Nets

5.1.1 Sequential Actions



Figure 5.1.1.a: Sequential action

Each sequential flow is described through the transportation of a single token from start to finish; travelling through a single set of alternating transitions and places.

5.1.2 Cycles

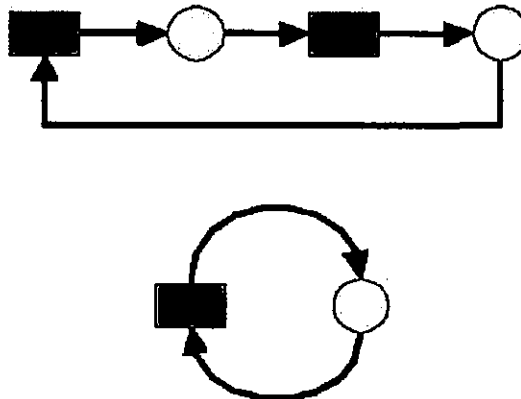


Figure 5.1.2.a: Cycles

Cycles are formed when the arcs direct back to the beginning of the sequence, thereby initiating a continuous loop of the token.

5.1.3 Dependency

This can occur when a Transition has several input Places – it can only be enabled when all of the input Place receives, at least, one Token [Her97].

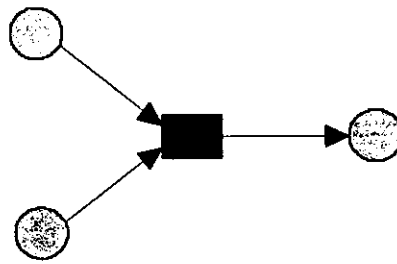


Figure 5.1.3.a: Dependency

5.1.4 Concurrent processes

This can occur when a Transition has several output Places – each Place will obtain a Token when the said Transition fires. This forms the start of a concurrent process [Her97].

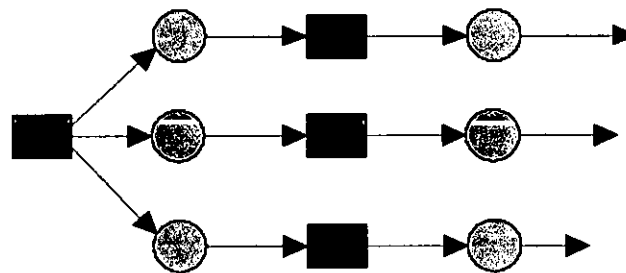


Figure 5.1.4.a: Concurrent processes

5.1.5 Synchronisation

This can occur when a Transition has several input Places - it can only be enabled when each Place receives a Token.

The Transition acts a ‘stop and wait’ element to synchronise the concurrent flows of Tokens [Her97].

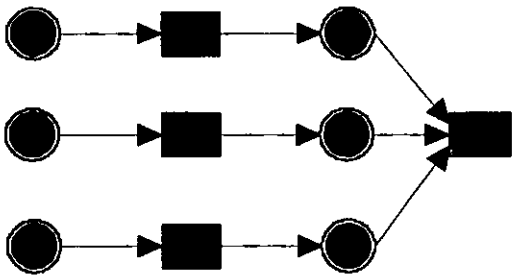


Figure 5.1.5.a: Synchronisation of concurrent flows

Synchronisation can also be used to pace-out the flow of Tokens through a process (e.g. buffers) [Her97].

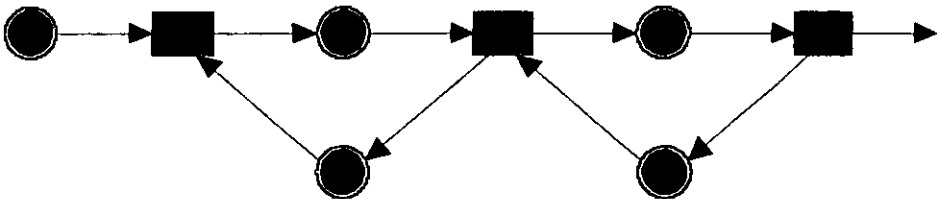


Figure 5.1.5.b: Two-level deep buffer

Synchronisation can also be used to design common resource stores [Her97].

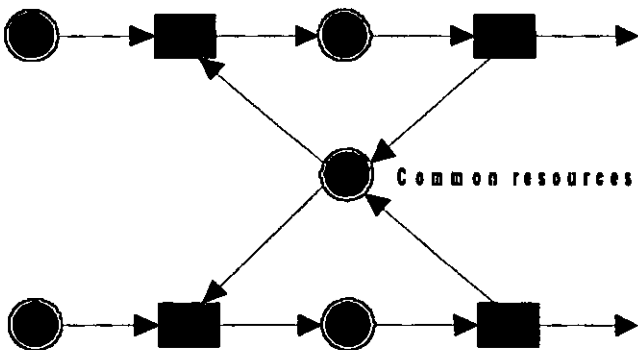


Figure 5.1.5.c: Common resource store

5.1.6 Decision-making /conflict

This can occur when a Place has several output Transitions – the Transitions will contend for the limited Tokens at that Place (i.e. Transitions are said to be in conflict).

This can be used to model asynchronous decision-making processes, which dictate the separate flows of Tokens [Cha97].

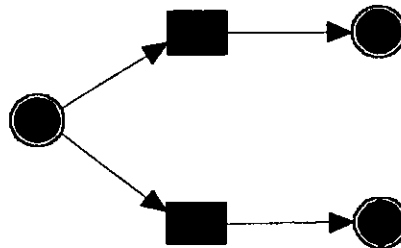


Figure 5.1.6.a: Decision-making/conflict

Note that the above set scenarios can be present within a system model as a static placement of Transitions and Places. They can also be present as dynamic structures. For example a feedback cycle, not necessarily evident through the examination of the physical layout of modelling components, can manifest itself at run time in an undetermined fashion. The ambiguity of feedback cycles would be caused by the inclusion of decision-making/conflict scenarios within the system that would redirect token flow based on the run-time input conditions.

5.2 *Extensions of Petri-Nets*

Extended Petri-Nets have important additional features, which make the modelling of Complex Systems (specifically communications networks) possible. These features add programming concepts to the model, thereby allowing the modeller to adjust the flow of Tokens within the net and to vary the Tokens themselves. These features are:

- The addition of timed delays to Transitions - to model points of delay within the network.
- Input Arc Inscriptions – specifies the type of data that must exist in order for an activity to occur (i.e. a Transition to fire).
- Guards - Boolean expressions, which define additional conditions for the enabling and firing of Transitions.
- Output Arc Inscriptions – specifications of the data that will be produced when an activity occurs (i.e. a Transition firing).
- Data-types, data-objects and variables – Tokens can be constructed as data-objects that carry complex/custom data structures. Data-types (also known as Colour-Sets) define the type attributes assigned to these data-objects (i.e. every Token in a Petri-Net is typed on some Colour-Set, just as every piece of data in an ordinary computer programme is of some data-type). Variables hold Tokens of a defined type and can be used in evaluating Boolean expressions at Transitions, at run time. Variables form part of the Arc Inscription and act as filters on the types of Tokens being transferred across [Her97].

Petri-Nets and Petri-Net extensions fall into several categories (e.g. Time Petri-Nets, Stochastic Petri-Nets, Object-oriented Petri-Nets). This work has opted to use ‘Coloured Petri-Nets’ and in particular ‘Hierarchical Coloured Petri-Net’ to form the foundations for the modelling process. Coloured Petri-Nets, as mentioned before, introduce a typed token in order to differentiate between and control process flows. Hierarchical Coloured Petri-Nets provide the facility to build complex models as a multi-tiered structure of interconnected hierarchical subnets and reusable components.

5.3 *Petri-Net Simulation of Active Networks*

It is believed that the application of Petri-Net theory to the modelling of Active Networks remains a concept unique to this work. Petri-Nets are suitable for the modelling of Active Networks because the Token/Place elements can be matched with Active Packets, Active process execution blocks and Active Nodes.

For this work an Active Packet/Application corresponds to a single Token. An Active process will correspond to a single Place or a group of Places intermixed with Transitions that provide the required Active functionality. The Colour Petri-Net feature can differentiate Active Applications types - each Token can have varying properties, a unique identification and the ability to invoke different processes within a cluster of Places/Transitions. This would be equivalent to an Active Packet invoking a specific process within an Active Node.

The third party simulator used in this project attempts to model a significant number of Active Network features deemed relevant to the Emergence of anomalous behaviour. Within the Active Network simulation, Petri-Nets are capable of displaying the following (as identified by the author):

- Specific actions taken by Active Packets – as a Token (an Active packet) enters a cluster of Places and Transitions (an Active Node) it is identified and coupled to a specific process flow. The specific process is then executed.
- Active Packets travelling through various stages of a process.
- Parallel process execution.
- Resource allocation to processes – resources (as denoted by special Tokens) are moved out of the common resource store when a process is initiated. They are returned when the process has finished execution. In some cases, a process will retain resource Tokens in an attempt to maintain a state (imprint) in memory.
- Packet interaction through the dynamic competition for resources.
- Packet transformation.
- Packet loss and delay.
- Packet replication.

- **Packet types** – the colour Petri-Net scheme has the capability to assign attributes to Tokens, thereby differentiating them according to the Active Packet types. The attributes are used to control and channel the flow of packets to various processing elements.

5.4 *Design/CPN*

The author has investigated a number of Petri Net simulation tools (reviewed in Appendix iii). Of these Design/CPN was found to be a viable package offering an elaborate and comprehensive set of features. It supports Colour Petri-Net models with complex data-types (Colour-Sets) and complex data manipulations (Arc Expressions and Guards); both specified by the “Standard ML” programming language. In addition the package allows the use of Standard ML code to customise the behaviour of simulations (i.e. additional code segments in transitions and the use of global reference variables).

The package also supports hierarchical and modular nets (i.e. complex models can be decomposed into manageable modules. Separate modules are reusable and can be constructed with well-defined interfaces). Design/CPN has the capability to model a typical scenario of 50 to 200 modules; each with 10 to 50 different Places and Transitions [Des]. Design/CPN’s user-extensibility allows for the accurate modelling of Active Networks in accordance to the specifications set by the high-level modelling scheme. It is well suited for progressive and intensive simulation of an Active Network. The package provides comprehensive documentation, samples modelling scenarios and performance analysis capabilities.

The software was originally devised by the Meta Software Corporation, Cambridge MA, USA with the help of the CPN group at the University of Aarhus, Denmark. Subsequent decisions by Meta to transfer development and support of the tool to the CPN Group at Aarhus resulted in it becoming freely available to model developers. However, as of January 2004 it has been superseded by ‘CPN Tools’, which provides an enhanced GUI (working under Windows 2000/XP with OpenGL support) with a faster simulation engine. Nevertheless, much of the modelling/simulation technology used in the new tool is based around Design/CPN. From a historical perspective, Design/CPN provided a complete modelling/simulation package from the initial stages of this research right through to completion. The version of Design/CPN used for this work runs under Linux (RedHat 8.0).

The overall suitability of a particular simulator was assessed on the following criteria:

- Ability to simulate large networks
- Capability of building hierarchical nets and high-level simulations
- Inclusion of Token differentiation (e.g. Colour Petri-Net schemes)
- Inclusion of performance analysis measures
- Adequate documentation and support
- Inclusion of a graphical display and animation of networks
- Availability and cost

5.5 Model flow diagrams/layouts

5.5.1 Model hierarchy

The Active Network model was constructed using a hierarchical and modular approach. This approach groups common functionality into ‘components’. The components are sub-sections of the Active Node ‘object’ and collectively provide the primary functions. The design/CPN object hierarchy for the Active Network model is structured as follows:

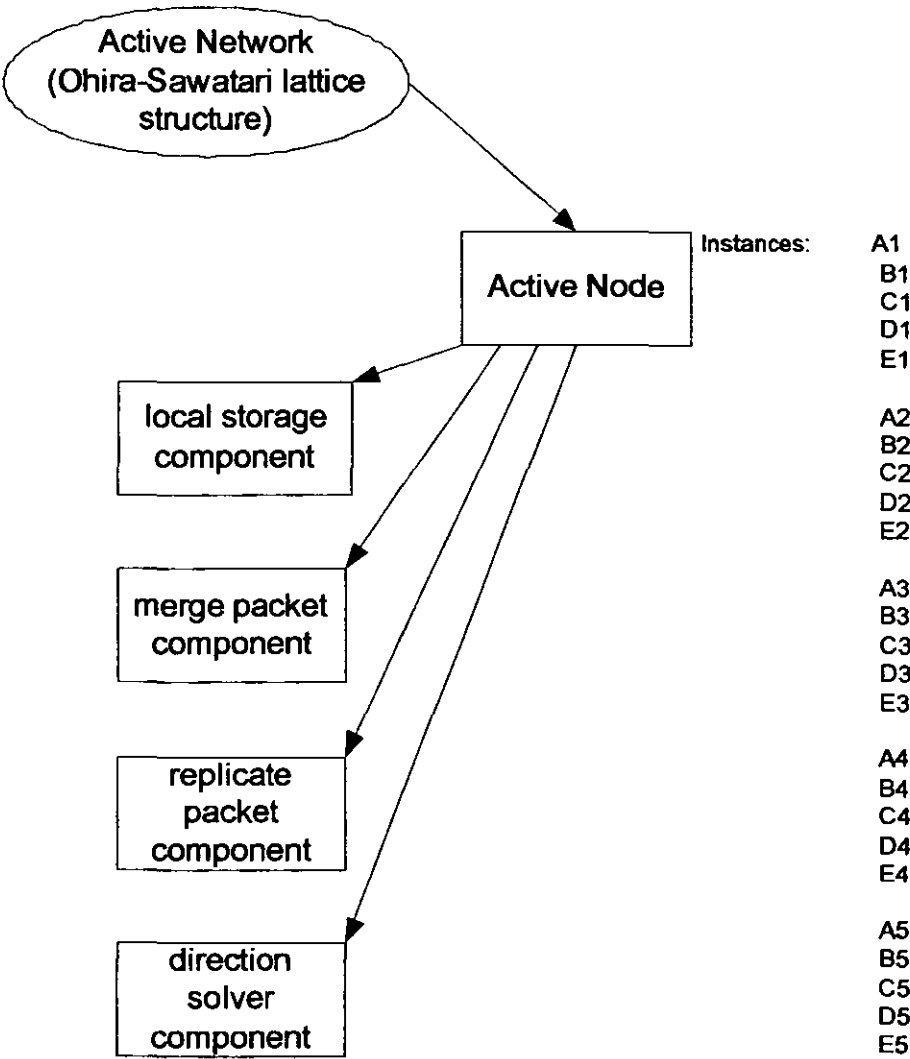


Figure 5.5.1.a: Model Layout: Active Network Model Hierarchy

The top-level object in the hierarchy is the Active network lattice structure (Figure 4.2.a and Figure 5.5.2.a). A sub-component of this network arrangement is the Active Node object (Figure 5.5.3.a), which forms the second level in the hierarchy. There are 25 instances of the Active Node object within the model representing the 25 Active Nodes (A1 to E5). Sub-components of the Active Node object include the following: “Local Storage Component” (Figure 5.5.4.a), “Merge Packet Component” (Figure), “Replicate Packet Component” (Figure 5.5.6.a) and “Direction Solver Component” (Figure 5.5.7.a). The sectioning of components was primarily based on the logical apportion of functionality (e.g. Active Merge functionality and Active Replication functionality) and/or level of diagrammatic detail required to implement specific features (e.g. Direction Solving for the next hop).

It must be noted that the following flow diagrams are not the actual Petri Net diagrams used in modelling process, but are an approximation. These approximations are used as a process by which the modelling details can be easily explained. It was believed that the use of the actual diagrams, at this stage of the report, would only complicate the documentation and thus are included in Appendix ii. The author proposes that the use of flow diagrams, along with the appended notes, will provide sufficient detail for the reader to comprehend and analyse the Active Network model.

Also note that the control flows of tokens (represented by the green coloured arcs that transfer tokens from Place to Place via Transitions), in the following diagrams, are separate from the Active Packet Token flows. The controls flows are a mechanism (in addition to the Active Packet Token flows) necessary for progression of the Petri-Net model within a simulation run. Furthermore, they do not influence the resource utilisation data; gathered as output from the simulations.

5.5.2 Active Network (Ohira-Sawatari lattice structure)

The following diagram provides a detailed and concise view of the Active Network model as abstracted in Figure 4.2.a.

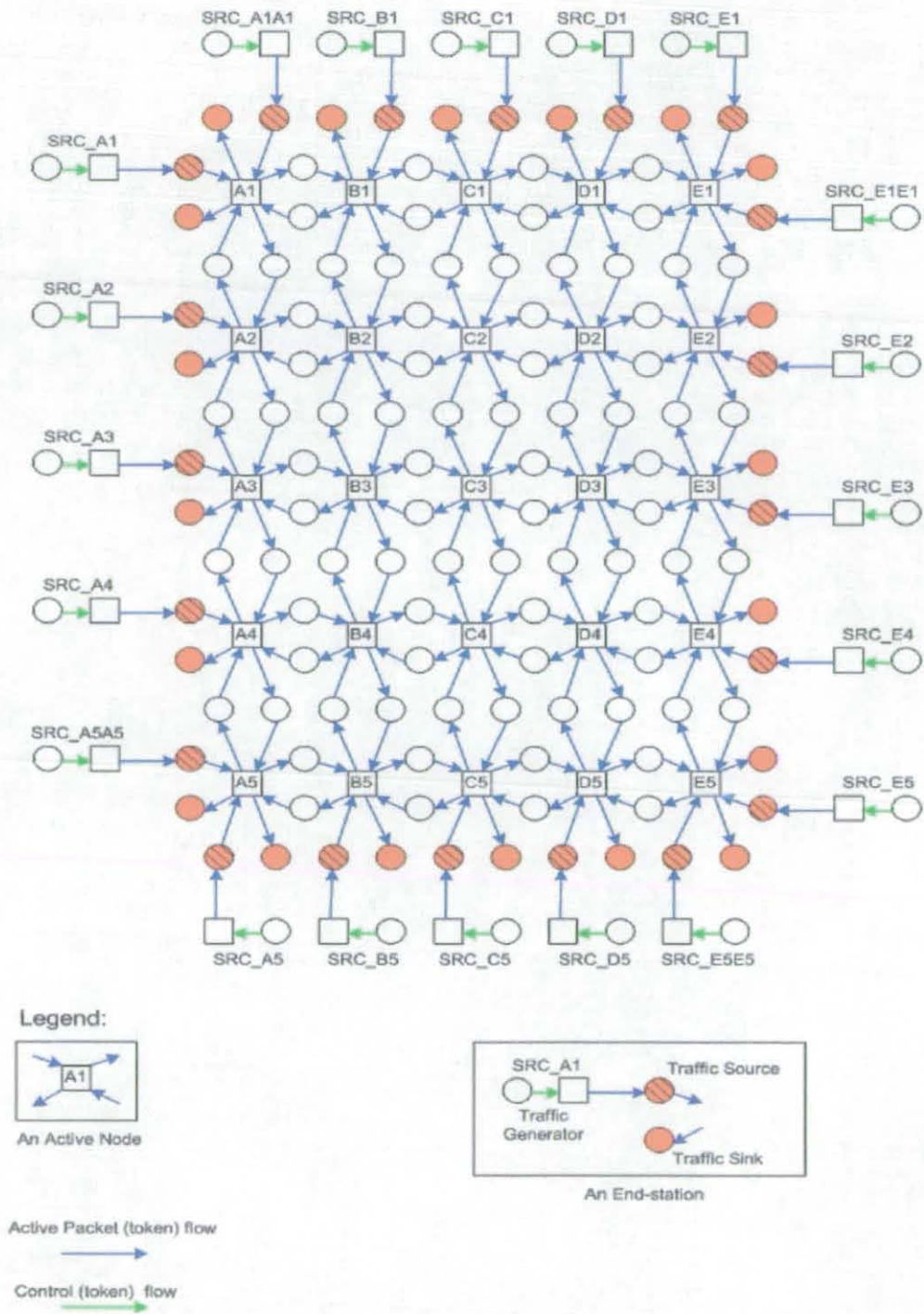


Figure 5.5.2.a: Model Layout: Active Network

Each Petri Net Transition labelled from A1 to E5 represents an ‘instance’ of an Active Node. Surrounding the lattice structured Active Node ‘core’ are 20 end-stations that form the ‘end-user edge’ of the model (Appendix ii; section ii.3).

Active Packets/Applications are generated and collected via end-stations. Within an end-station there exists a 'traffic generator' that reads an input file ("src.txt") in order to obtain the specific Active Packet/Application information corresponding to the particular end-station, which is then forwarded to the 'traffic source'.

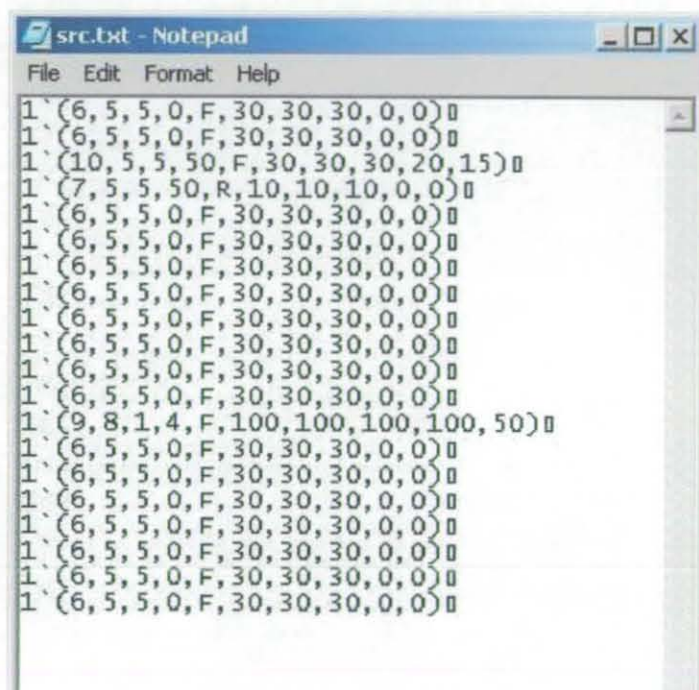


Figure 5.5.2.b: Sample Source File ("src.txt")

Each line of the input file ("src.txt") contains a data section (within the brackets) that describes an entire Active Packet/application with the value sequence corresponding to the pseudo header/data structure described below:

Application Number	Direction Indicator 1	Direction Indicator 2	Time-to-live	Route Mechanism
1 – 15 Integer value	1 – 8 Integer value	1 – 8 Integer value	Integer value	<ul style="list-style-type: none">• forward• replicate• consume• merge	

.....	Memory Requirement	Processing Requirement	Buffer Requirement	Global State Maintenance (GSM)	GSM Timer Count
	0 – 100% Integer value	0 – 100% Integer value	0 – 100% Integer value	0 – 100% Integer value	0 – 50 Integer value

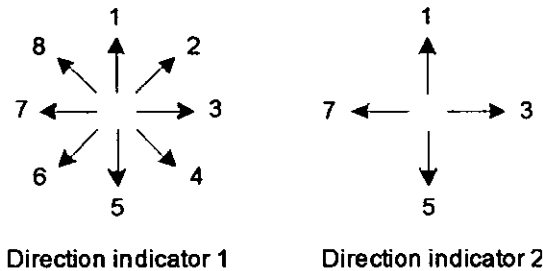


Figure 5.5.2.c: Active Packet/Application Structure and Direction Indicator values

Note that the Active packet/application does not, in reality, contain a data portion. The author proposes that the sole use of the Active header is sufficient to provide an accurate analysis of resource usage in Active Networks, since the requirement is explicitly specified within it. Furthermore, it was not the intention of this research project to assess the quality of service (QoS) aspect of data transfer in Active Networks.

5.5.3 Active Node Model

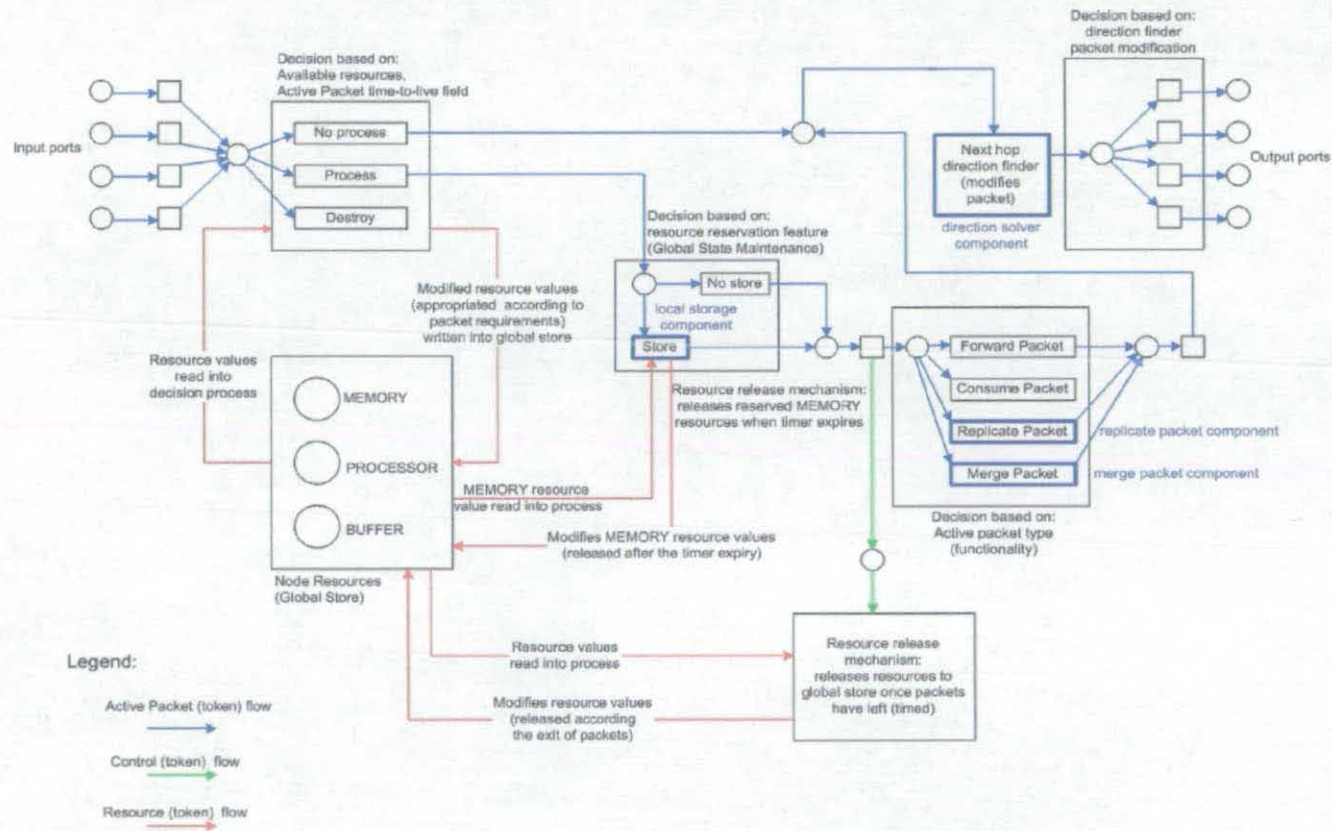


Figure 5.5.3.a: Model Layout: Active Node

The Active Node functionality (Appendix ii; section ii.4, section ii.5, section ii.6, section ii.7 & section ii.8) can best be explained through the description of processes it invokes, when an Active Packet/Application arrives and is passed through, in addition to description of interactions made with the packet:

- An Active Packet arrives at the Active Node through one of 4 possible input ports and is immediately evaluated for processing by the node.
- The decision to process the particular Active packet will depend on the “time-to-live” field and whether the node will have, at that particular time, adequate “MEMORY”, “PROCESSING” and “BUFFER” resources (Figure 5.5.2.c).
- Unprocessed packets will be forwarded to the ‘next hop’ Active Node through one of 4 possible output ports.
- The choice of output port will depend on the evaluation of the Active Packet’s 2 “Direction Indicator” field values by the “Direction Solver Component” (Figure 5.5.2.c and Figure 5.5.7.a).
- Active Packets that can be processed will initially consume the required “MEMORY”, “PROCESSING” and “BUFFER” resources from the Global Store.
- There exists a mechanism to release the consumed resources after a specified time period, which coincides with the point of exit for the corresponding packet (“MEMORY” resources have additional criteria). The specific time period for release is based on the “Route Mechanism” field of the Active packet (Figure 5.5.2.c).
- The initial consumption “MEMORY” resources may include a quantity used to provide the ‘Resource Reservation Feature’ in Active Networks. The amount and time limit for these “MEMORY” resources will depend upon the “Global State Maintenance” value and the “GSM Timer Count” value of Active Packets, respectively (Figure 5.5.2.c). There exists a mechanism to release these resources once the Timer Count expires, which is held within the “Local Storage Component” (Figure 5.5.4.a).
- Active packets are then differentiated (and processed) based on the “Route Mechanism” field.

- The packets are subsequently passed into the “Direction Solver Component” (Figure 5.5.7.a) for evaluation and passed out through the output ports.

5.5.4 Local Storage Component

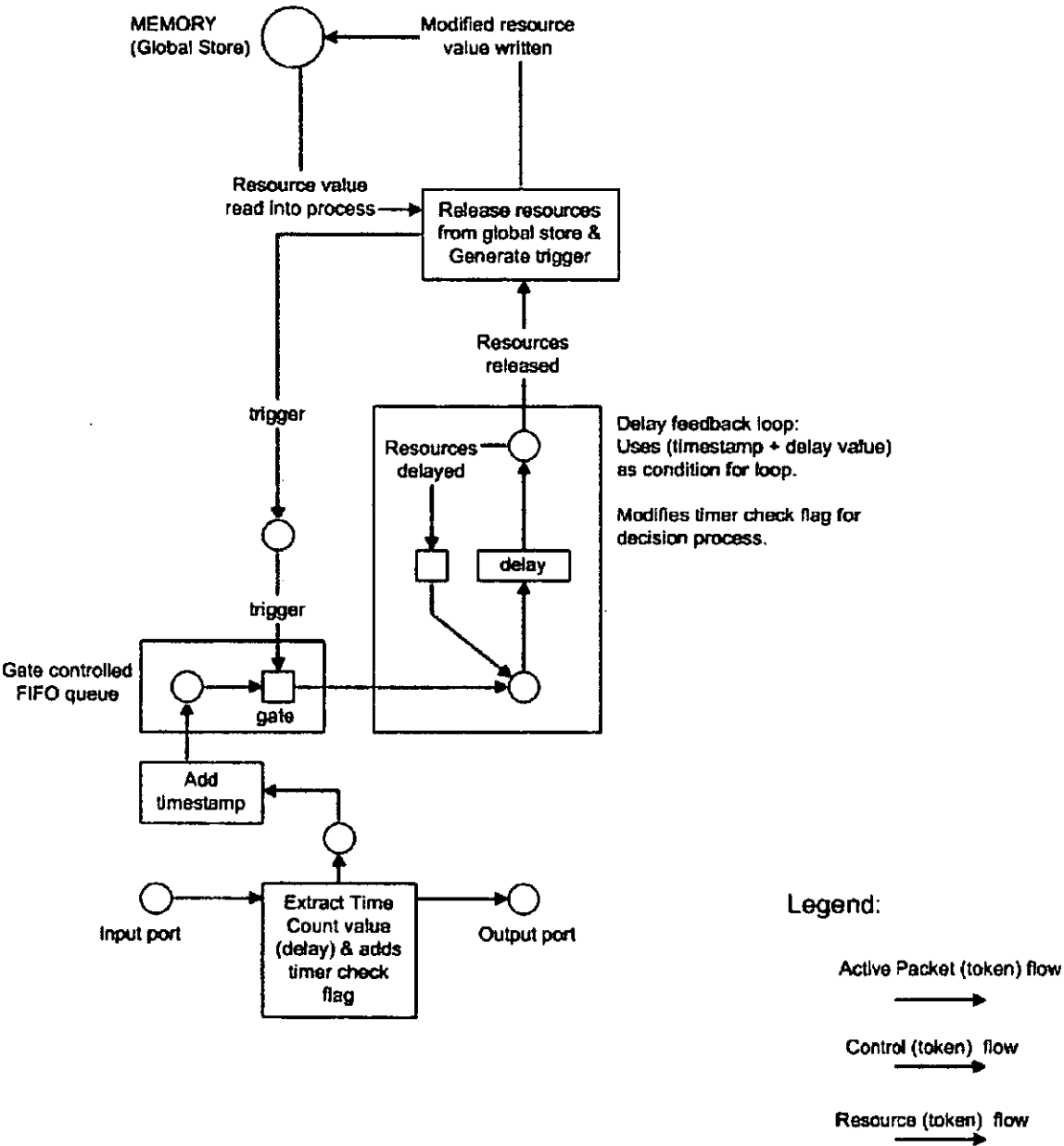


Figure 5.5.4.a: Model Layout: Local Storage Component

The objective of the “Local Storage Component” (Appendix ii; section ii.9 & section ii.10) is to provide a timed release of “MEMORY” resources that were previously consumed when the packet was initially processed.

The timed release of “MEMORY” resources is distinctly separate from normal ‘consume and release’, which is applicable to all Active Packets processed within the node. The ‘consume and release’ of additional “MEMORY” resources is an integral part of the resource reservation feature of Active Networks.

The “Local Storage Component” is invoked only when the higher layer “Active Node” object requests it. The request is generated based on the “Global State Maintenance” value and the “GSM Timer Count” value of Active Packets that pass through the Active Node (Figure 5.5.2.c).

An Active packet enters the “Local Storage Component” through the input port and immediately exits through the output port via a process designed to extract the “Global State Maintenance” value (reserved resource amount) and the “GSM Timer Count” value (timer for reservation). The extracted data can be thought of as a ‘reduced information Active Packet’.

Much of the information flow within this component is control flow that is used to add a timestamp and check flag to the extracted data.

The reservation of resources is formed by a delay loop that is controlled by the “GSM Timer Count” value and the timestamp.

The check flag is used to trigger the release of “MEMORY” resources once the delay loop criteria has been satisfied.

5.5.5 Merge Packet Component

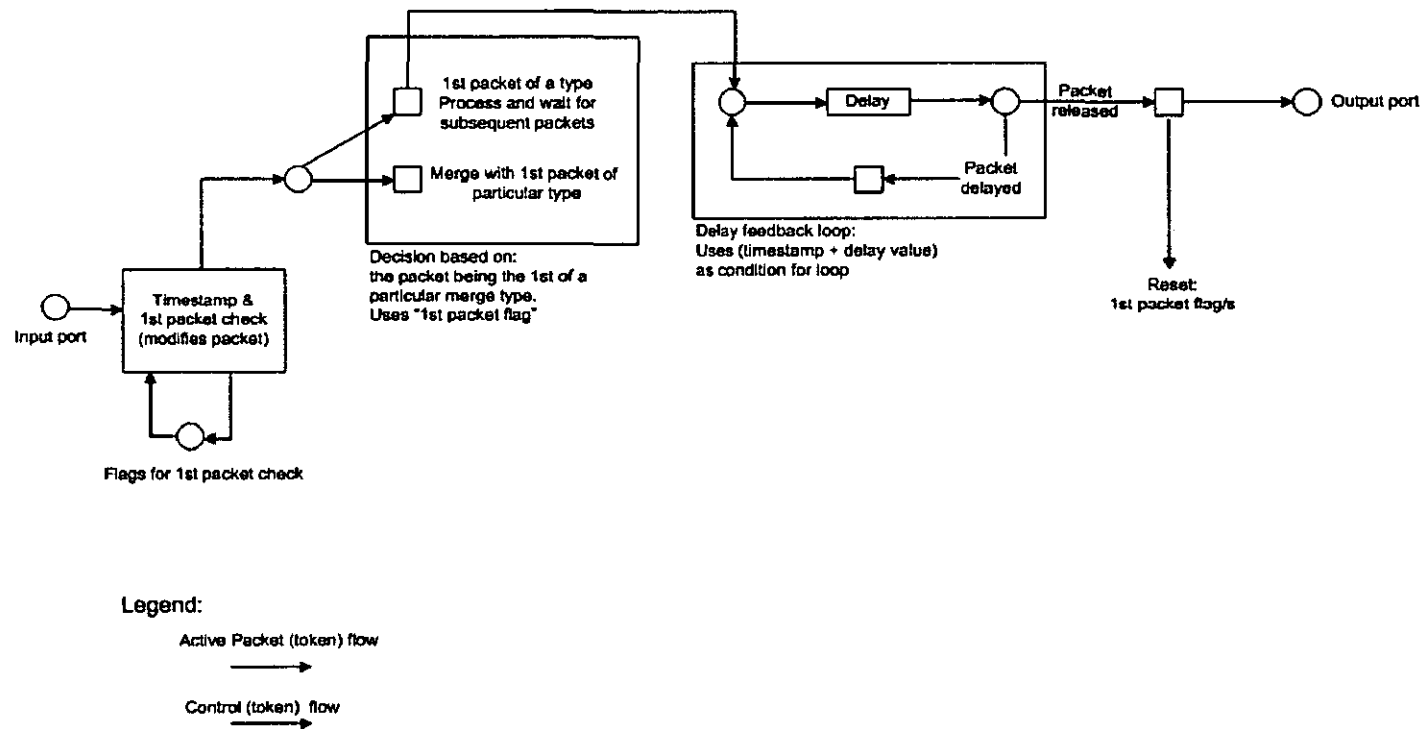


Figure 5.5.5.a: Model Layout: Merge Packet Component

The objective of this component (Appendix ii; section ii.11 & section ii.12) is to merge identical Active Packets that have been designed for the purpose into one Active Packet. The process of merging in terms of this project involves the delay of the 1st packet that entered the Active Node processing stream, which has a 'merge' value in the "Route Mechanism" field, for a limited period of time (i.e. a fixed period of 5 simulation time steps).

Subsequent 'merge' packets that match the 1st packet's "Application Number" and arrive at this component within the fixed time period will be consumed/merged.

The component identifies and tags the 1st packet of the merge process with a check flag. This flag is reset when the 1st merge packet is released after the expiry of the fixed delay period.

The delay of the 1st packet is achieved through a loop that is controlled by a timestamp, which was attached to the packet as it entered this component.

5.5.6 Replicate Packet Component

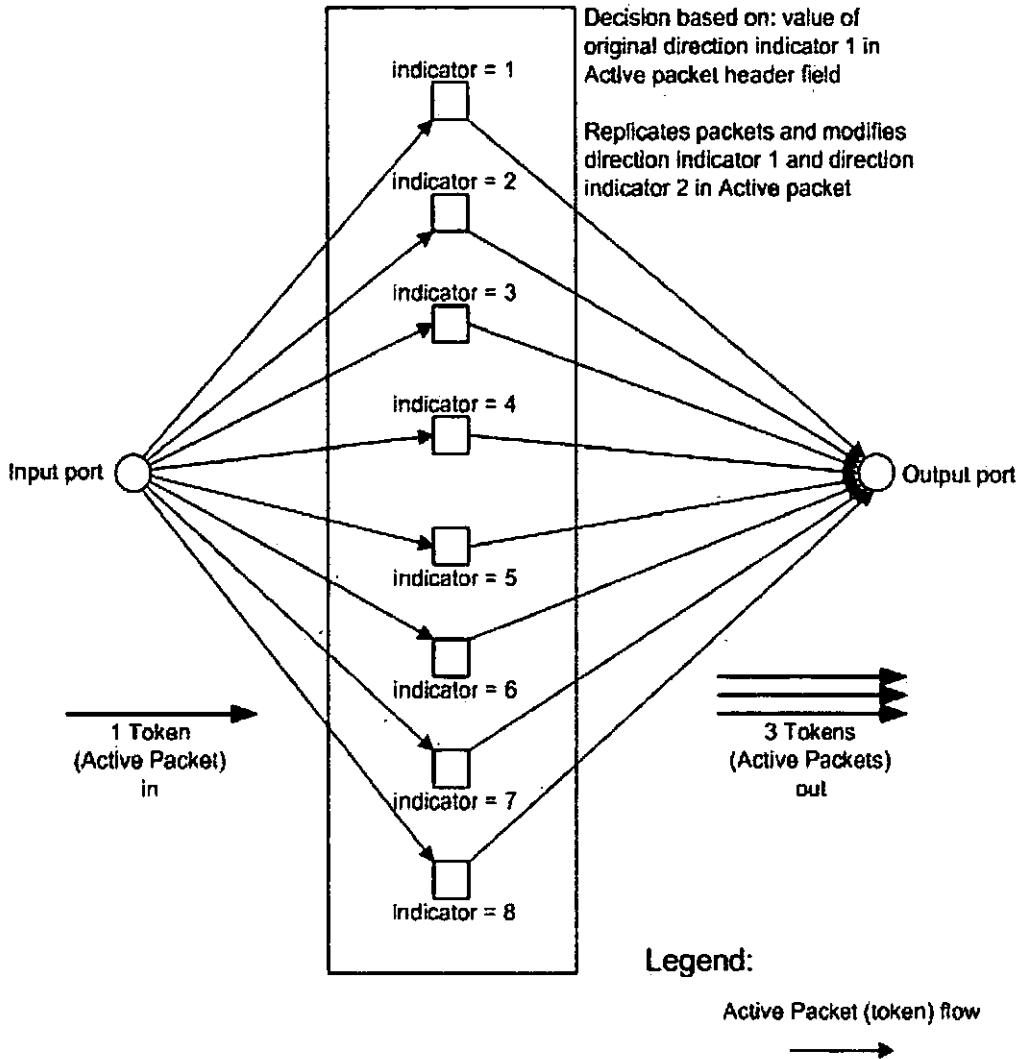


Figure 5.5.6.a: Model Layout: Replicate Packet Component

The “Replicate Packet Component” (Appendix ii; section ii.13) implements a simple mechanism to replicate all packets taken from the input port (according to a specified scheme) and output them into the main/parent “Active Node” object.

The replication scheme (Figure 5.5.6.b) uses a 1 to 3 replication of Active Packets with modified “Direction Indicator” values (i.e. the original direction and 45 degrees either side of the original).

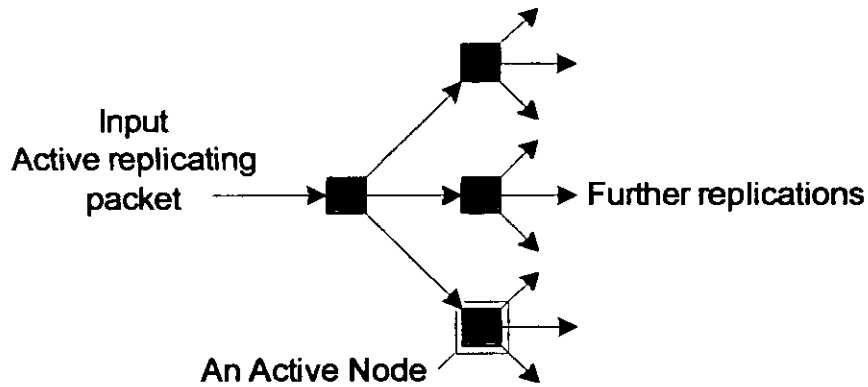


Figure 5.5.6.b: Active Packet Replication Scheme

The scheme is designed to emulate a spanning replication of data that has an unlimited replication objective and a primary direction (i.e. the direction of initial data packet). This is one of many possible spanning schemes that could have been incorporated into the model. It was primarily designed to achieve a significant ‘presence’ of a replication, which affected other application streams, whilst not allowing to have the capability to completely saturate the network.

The main section of the “Replicate Packet Component” consists of a decision process (based on the original direction of the input packet, the value of which is held in the “Direction Indicator 1” field of the Active Packet header) that replicates the input packet and modifies both Direction Indicator values of each output packet according to the scheme in Figure 5.5.6.b.

5.5.7 Direction Solver Component

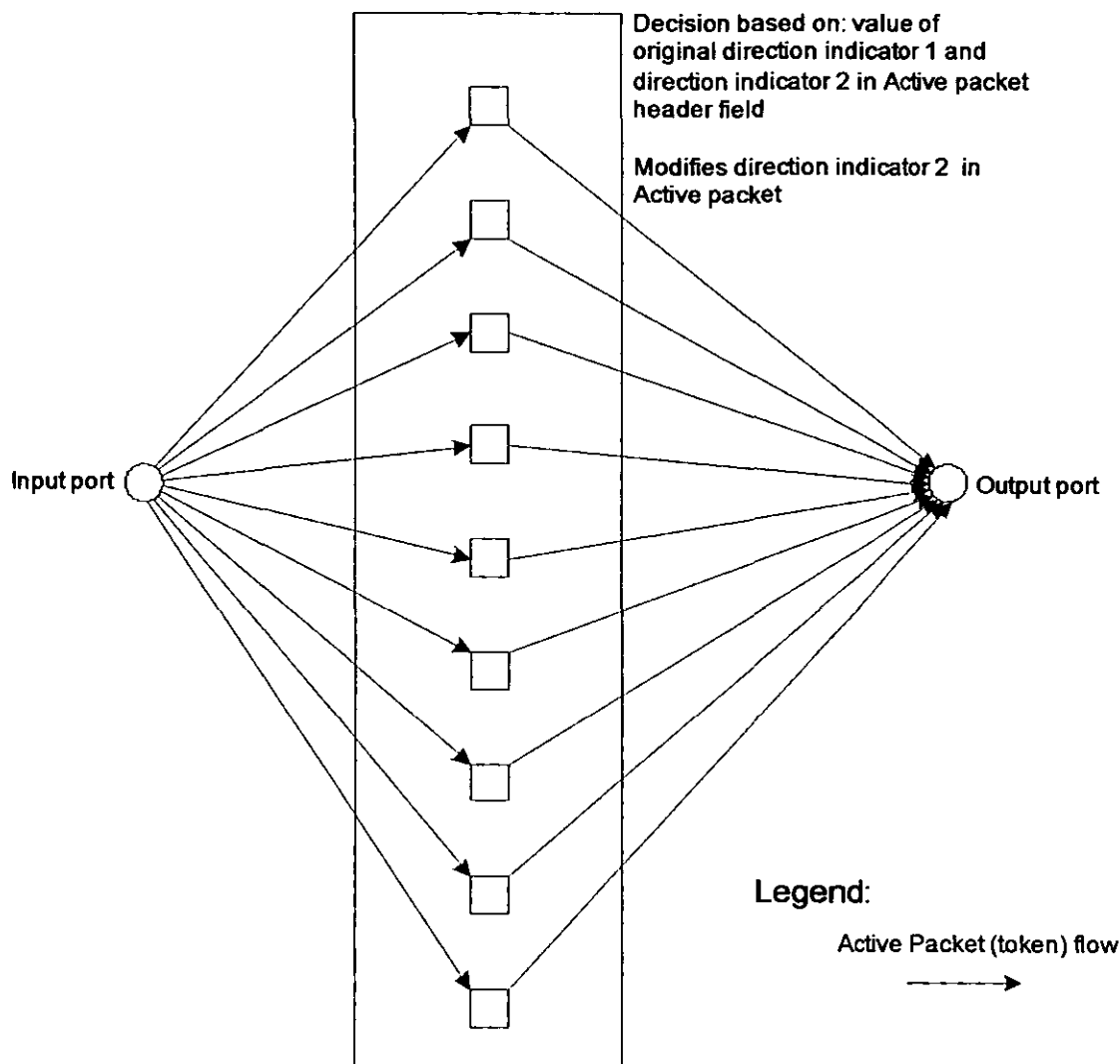


Figure 5.5.7.a: Model Layout: Direction Solver Component

The objective of this component (Appendix ii; section ii.14) is to take all Active Packets destined for the node exit and modify their “Direction Indicator 2” values. The component only modifies the secondary direction indicator value of diagonally traversing packets in order that they map correctly to one of the 4 output ports. Any horizontally traversing and vertically traversing packets are left unmodified. The diagonally traversing Active packets, in reality, travel through the network in a step-wise pattern. The alternating horizontal and vertical direction values required for this

type of travel are contained in the “Direction Indicator 2” field of the Active Packet and are adjusted accordingly by the “Direction Solver Component”. (Figure 5.5.2.c and Figure 5.5.7.a)

The combination of changes made to the “Direction Indicators” is described in the following table:

Combination	Input	Output
	“Direction Indicator 1” and “Direction Indicator 2” values	“Direction Indicator 1” and “Direction Indicator 2” values
1	2 and 1	2 and 3
2	4 and 3	4 and 5
3	6 and 5	6 and 7
4	8 and 7	8 and 1
5	2 and 3	2 and 1
6	4 and 5	4 and 3
7	6 and 7	6 and 5
8	8 and 1	8 and 5
All other combinations of “Direction Indicator” values remain unaffected		

5.5.8 Other components: Data Logging

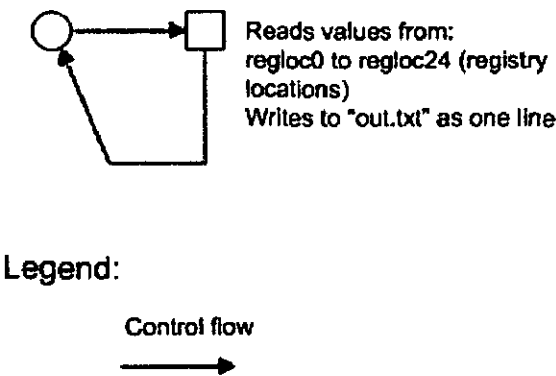


Figure 5.5.8.a: Model Layout: Data Logging Component

The input for the Emergence detection process, described in the proceeding chapter, consists of a data log of “MEMORY” resource activity, which was recorded throughout a simulation run of the Active Network.

The data logging process (Appendix ii; section ii.15) consists of a simple Petri Net flow loop that polls and records (i.e. writes to an output log file “out.txt”) 25 Register Locations, at every simulation time step.

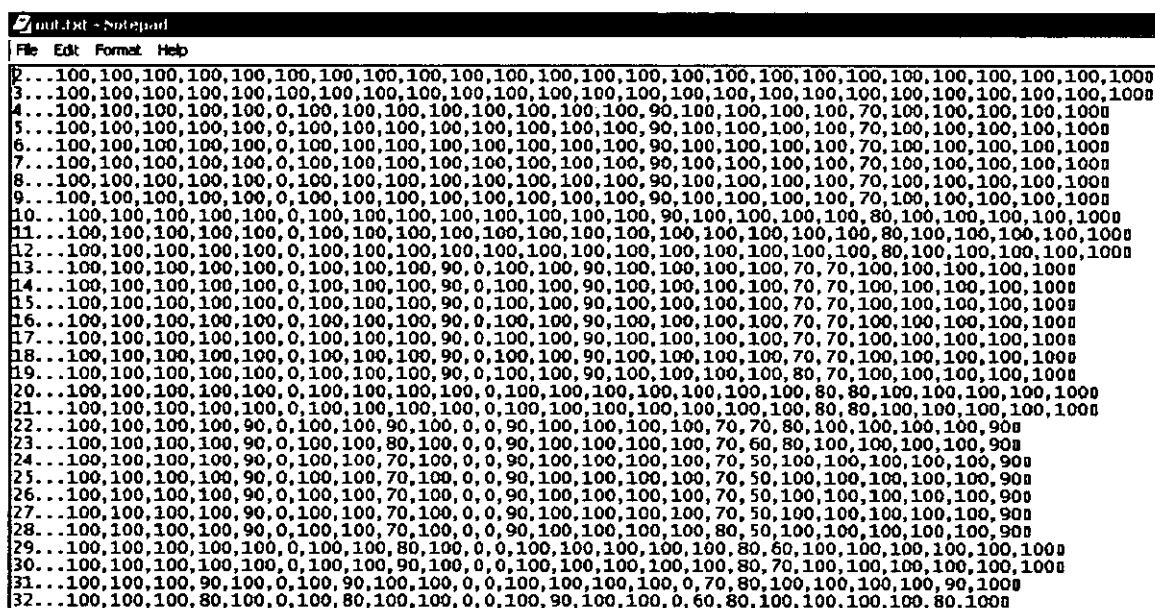


Figure 5.5.8.b: Sample Output log file “out.txt”

A specific Register Location contains, at any given simulation time step, the current value of “MEMORY” resources contained within a specific instances of the Active Node object.

A Register Location is updated whenever the “MEMORY” Place (Global Store) value, of the corresponding Active Node instance, changes during a simulation run (Figure 5.5.3.a). This update is achieved through the inclusion of Standard ML code in Transitions in the Active Node object and its components.

Each line within the output log file (“out.txt”) contains the simulation time step number followed by 25 Register Location values (“regloc0” to “regloc24”). The following table

maps the Register Location names with the Active Node names who's "MEMORY" resource values are held within the registers:

Register location number/name	Active Node Names
"regloc0"	E5
"regloc1"	E4
"regloc2"	E3
"regloc3"	E2
"regloc4"	E1
"regloc5"	D5
"regloc6"	D4
"regloc7"	D3
"regloc8"	D2
"regloc9"	D1
"regloc10"	C5
"regloc11"	C4
"regloc12"	C3
"regloc13"	C2
"regloc14"	C1
"regloc15"	B5
"regloc16"	B4
"regloc17"	B3
"regloc18"	B2
"regloc19"	B1
"regloc20"	A5
"regloc21"	A4
"regloc22"	A3
"regloc23"	A2
"regloc24"	A1

5.6 Summary

This chapter has discussed, in considerable detail, how the Active Network model was constructed, simulated and prepared for analysis.

The first section of this chapter describes the general diagrammatic concepts of Petri Nets and their use with communication network type structures.

The sole use of the diagrammatic semantics of Petri Nets (in order to provide solutions for system design) is not uncommon. Most Petri Net modelling packages also provide powerful mathematical analysis tools to develop “Occurrence Graphs” that can be used to gain a detailed understanding of systems. For the purposes of this project these mathematical tools provided no useful contribution. This is because the basic premise of the research was to establish a global view of system functionality in Active Networks and then proceed to detect Emergence as patterns within this view.

The project does, however, consider the use of Petri Net extensions such as “Color Petri Nets”, Boolean Transition Guards, Arc Inscriptions and Hierarchical decomposition invaluable in the accurate modelling of Active Networks.

This chapter follows on to explain the process by which the Petri Net concepts are paired with the High-level Active Network model features (previously identified in chapter 4).

The chapter also provides, as reference, a significant list of Petri Net modelling tools and their specific attributes. The list formed the basis of an evaluation process to find the most suitable Petri Net modelling/simulation tool. As a result, Design/CPN was highlighted as being the best suited for this particular modelling and simulation task. Some of the key ‘selling points’ were its ability to create hierarchical models using well-defined components and its ability to construct and execute “Standard ML” code segments within “Transitions”. This provided a high degree of flexibility in the implementation of features (e.g. data-logging) that would not have been possible through a solely graphical simulation environment.

The last section of this chapter contains detailed flow diagrams of the Active Network model. Included in the diagrams are the breakdown of components (object hierarchy), the specific detail pertaining to resource usage and the flow of Active Packets/Applications.

C H A P T E R 6
D E T E C T I O N O F
E M E R G E N C E

6 Detection of Emergence

This chapter will present:

- An analysis of possible detection techniques for Emergence in systems.
- A description of the strategy used to test the suitability of a likely technique.
- A description of Self-Similarity and its link to an Emergent Behaviour in the Active Network model; termed a “Cascading Effect”.
- A detailed mathematical derivation of the R/S statistic used to measure and quantify Self-Similarity.
- Approximations and limitations of the R/S statistic.
- A detailed mathematical derivation of a confidence value employed to support the R/S statistic.
- The application of the R/S statistic, to the Active Network simulation results, in order to produce Hurst values (i.e. Self-Similarity measure) for Active Node resource usage fluctuations.
- The link between the Emergence of a “Cascading Effect” and Self-Similarity.

The key concepts in this chapter are: detection of Emergence, Self-Similarity of Active Node resource usage fluctuations, the R/S statistic, approximations, the Hurst value, the r^2 “goodness-of-fit” confidence value, MATLAB algorithm to produce a Hurst value via regression analysis, mathematical derivations

6.1 Overview

The detection of Emergent Behaviour in any complex system requires a significant amount of empirical study as discovered during the course of the research work. Any potential technique must be customisable to suit the results generated by system modelling and simulation. As a precursor to the evaluation of a detection technique, it is proposed that Emergence is formalised as a dramatic change in the functioning of a complex system, which is recognised as the manifestation of characteristics mentioned in chapter 3; section 3.1. This would in turn transform a system from a normal to an anomalous state or from a stable to an unstable state.

During the course of this investigation into potential detection techniques it was discovered that, whilst researchers agree upon the importance of discovering Emergence in systems, there exists little in the way of development of detection tools to suit all systems. This chapter describes 3 examples of probable methodologies that are sourced from research work, which proposes radical steps forward in the detection of Emergence in general systems. These are:

- Emergence as the loss of complexity in a system
- Emergence through the measure of self organisation
- Emergence as patterns in a system

The concepts fundamental to each of the 3 broad methodologies are described, in brief, in the following sections along with the reasons for choosing one particular detection method for this research.

6.1.1 Emergence as the loss of complexity in a system

This detection method was proposed by a single piece of research work established to measure Active Network health and to provide some form of management for such systems. The proposed method includes the use of Kolmogorov Complexity and general Complexity Theory to:

- *“Build self-managed networks”*
- *Provide “vulnerability analysis techniques that draw on fundamental properties of information to identify, analyse and correct faults as well as security vulnerabilities in an information system”.*

This research visualises the Active Network not as a topology, but through ‘Complexity Measurements’. In other words the paradigm investigates the variables of the system (e.g. resource usage) and their relationships in order to compute complexities of the network in different dimensions (e.g. dimensions could be availability of services, health of devices, application performance, etc.). A high Complexity Measure is a result of high randomness in the system dynamics and translates to a healthy system. Dimensions of low complexity indicate potential problems in the system and possible Emergent Behaviour [Kul01].

Whilst this research provides a good technique in the detection of Emergence in Active Networks, it proved difficult to implement on top of an already developed high-level model. The complexity measurement process requires specific Management Information Base (MIB) components for the system, which relate to system variables. The visualisation of the Kolmogorov model is as a *“space filled with entities that represent the values of various monitored objects from the managed system”*. This is significantly different from the topology-based approach taken by the authors in developing high-level abstract views of Active Networks.

6.1.2 Emergence through the measure of self organisation

The detection of Emergence by this method relies on analogies taken from natural systems, in particular the structures of swarming and fully coordinated (“crystal”)

behaviours in simple flocking systems. The method is based on a single piece of research that makes use of natural system analogies to provide a measure for self-organisation in general systems. This in turn is used to indicate the presence or absence of Emergence. The research work proposes that the dynamics of a general complex system be modelled as being produced by an attractor/generator. The behavioural type exhibited by the system, whether it is a swarming or crystalline structure, is directly related to the dimensions of the attractor/generator. The research goes on to extract the measure of dimensionality (Ω) for the generator of a sample system. Furthermore, it proposes that the system has the ability to exhibit Emergent Behaviour based on the sudden transition of the Ω measure given smooth changes in system parameters [Wri00]. Even though the research provides a clear definition and methodology of Emergence, the system used in the analysis relies on non-discrete models of systems components that have defined mathematical functions. Thus the measure of dimensionality (Ω) is derived through the solution of a set of custom mathematical equations. This detection method is difficult to implement in models developed as discrete event-driven systems such as Active Networks.

6.1.3 Emergence as patterns in a system

This method relies on the belief that a dramatic change in system behaviour can be 'observed' and 'comprehended' through some visual technique designed for the recognition of Emergence. This concept is fundamental to many Emergence research projects [Bon97] [See]. Certainly in the above two sections (6.1.1 and 6.1.2) the structures, which are subject to the measurements described, are clearly patterns in a system. However the methods of pattern identification and the establishment of measures for Emergence vary. Similarly this section describes a range of practical pattern detection and measurement techniques that are used in research fields, other than Emergence detection, to a high degree of success.

For this work, the detection of Emergence as patterns in the system proved to be a viable option with respect to the following factors:

- **Applicability** – the results generated from the system modelling and simulation process consists of a 2-dimensional matrix of integers representing the resource

usage of all Active Nodes (as a percentage) at each simulation time step. All of the pattern detection techniques investigated within this project provided mathematical sequences that readily dealt with 2-dimensional number arrays. These sequences were easily programmed into scripts (executed in MATLAB) that provided algorithms for enhancement and detection.

- Adaptability - the techniques could be modified to suit the data type and sample size.
- Ease of use – from an engineering viewpoint, the development of pattern enhancing/recognition programmes, under MATLAB, was made relatively uncomplicated through the existence of sample code sequences and comprehensive reference material.

Described below are several types of pattern recognition techniques that were experimented with as the project progressed:

6.1.3.1 Image Enhancement algorithms – Edge-detection & Histogram Equalisation

The 2-dimensional matrix of logged simulation data is similar to the mathematical representation of a pixelated image prepared for image manipulation/enhancement (e.g. a grey-scale image can be represented in MATLAB as a matrix with the same dimensions as the image. Each pixel of the image is allocated a 'coordinate', which corresponds to the row and column number within the matrix. The grey scale intensity of each pixel is held as a value in the corresponding matrix element). Given this similarity it is possible to represent the raw data from the Active Network simulation as an image and prepare it for feature detection. Of the possible image detection/enhancement techniques available two were selected based on their strong analogous relationship with resource usage in an Active Network.

- With an Edge-Detection macro it is possible to identify a continuous high-contrast 'edge' within the 'image'. This translates to a continuous high or low resource usage within the system at specific Active Nodes, at specific time steps or a combination of both.

- The Histogram Equalisation/Threshold method is used to increase image contrast thereby clearly differentiating clusters of high and low resource usages within the data.

6.1.3.2 *Fourier analysis*

Fourier Analysis (1-dimensional and 2-dimensional) provides useful visualisation of matrix data in the frequency domain. Whilst it is a well-established technique in the enhancement and feature detection of images, the onus here was to use the method to identify unique frequency components with high amplitudes or clusters of components with particularly high frequency ranges. The identification of these frequency components would thus indicate the presence of a dominant fluctuation or a general high fluctuation of resource usage; both indicative of Emergence within the system [Gon02].

6.1.3.3 *Wavelet analysis*

Wavelet analysis is similar to Fourier analysis except that the frequency decomposition of data is conducted through a pre-defined non-periodic waveform ("wavelet") as opposed to a sine wave [Gra95]. In addition to the identification of strong frequency components, Wavelets have been used to identify Self-Similarity in data [MATa] [MATb]. As proposed by this research the idea that Self-Similarity is key to the detection of Emergence gives significant value to Wavelet analysis.

6.1.3.4 *Cross-correlation analysis*

Cross-correlation is used in discrete signal analysis to obtain levels of similarity between signal data sets. An investigation was undertaken to determine if this method could be used to correlate resource usage (over time) of individual Active Nodes with each other to find patterns that are similar (i.e. correlation of resource usage across the nodes at various data segment sizes) [Gon02]. A high degree of correlation would translate to a discrete pattern in resource usage traversing the network.

6.1.3.5 Self-Similarity analysis

The general idea of Self-Similarity has a strong relationship with Emergence in that it can be thought of as a pattern in the data that finds itself replicated at various resolutions. Self-Similarity provided credible results in the resolution of Emergence in Active Networks and is described in detail in section 6.3.

6.2 *Detection technique suitability testing*

In order to evaluate the effectiveness of any one technique, a suitability testing procedure was developed by using a group of test cases simulated through the Active Network model. The test cases were designed to contain an 'Emergence Inducing Factor' (EIF) in various hypothetical configurations. Emergence Inducing Factors are elements built into the inputs of a system or components of the system itself that are 'believed' to push it into Emergent Behaviour. The factors are chosen on various criteria and relate to the system model under scrutiny. In the case of this work the factor was perceived to be of value based on the effect it has on system instability.

After much consideration, the inducing factor was chosen to be a self-replicating Active Application/Packet that would replicate Active streams and span the network. Initial assessments of the replicating scheme indicated that it would push the system into an unstable state. However, subsequent experiments have shown that there are several factors present within the system that would control the replication and thereby indicate the presence of Emergence. Details of this are described in chapter 7, section 7.3.

In a simulation run the replication scenario was left to dominate resource usage at each node and analysed for potential effects on itself and other network traffic. The replication test cases were each modified to contain variations in the number of replication packets injected into the network, the direction taken across the lattice node structure and the amount of resources utilised at each encountered Active Node. These results were compared with simulations that did not possess the Emergence Inducing Factor. The entire process of suitability testing was iterative and was repeated for each of the probable detection techniques mentioned in section 6.1.3 in order to visually detect any interesting features.

The data analysed, in order to determine the presence of any anomalous characteristic, is the resource utilisation of all the Active Nodes under simulation. The resource utilisation statistic is a key component of this research for two reasons:

- Node resources invariably function as the enabler for all Active processes

- Resource utilisation statistics provide an abstract view of network performance and Active Network functionality.

6.3 Self-Similarity

Of the pattern recognition techniques reviewed/tested only the discovery of Self-Similarity within the data proved to be of significance in isolating Emergent Behavioural patterns. The Self-Similarity macro, developed through MATLAB (Appendix i), was able to isolate an Emergent characteristic within the replicating packet scheme that would otherwise have been hidden. The author has termed this Emergent Behaviour as a **“Cascading Effect” in resource utilisation** - a replicating phenomenon that, once formed within the Active nodes, continues to sustain itself through feedback. The phenomenon is detected empirically and is used as a starting point for the comprehension of the causes of this particular Emergent Property.

The Self-Similarity measurements of the data were established by calculating the Hurst parameter (a well-known estimator of Self-Similarity). There are several techniques present in mathematical literature designed to estimate the Hurst value of a data set. This research project used the classical R/S statistic [Lel94] to calculate the Hurst values of the resource usage fluctuations for each of the Active Nodes - each Petri Net simulation of the Active Network (Figure 4.2.a) produces an output matrix with 25 columns (one for each Active node) each containing resource usage values for 500 time steps. The columns were individually analysed for Self-Similarity using the R/S statistic (i.e. Hurst values calculated).

6.3.1 Calculation of the Hurst parameter – Rescaled Range statistic (R/S statistic)

“The R/S statistic is the range of partial sums of deviations of a time series from its mean, rescaled by its Standard Deviation” [Naw95]. This statement is best explained through the following derivation and through the MATLAB algorithm in Appendix i.

For a given set of observations taken from the original trace; $X_1, X_2, X_3, \dots, X_n$ for n periods with a sample Mean of \bar{X}_n and a sample Standard Deviation of $S(n)$, the classic Rescaled Range Statistic will be:

$$\frac{R(n)}{S(n)} = \frac{1}{S(n)} \left[\max_{1 \leq k \leq n} \sum_{j=1}^k (X_j - \bar{X}_n) - \min_{1 \leq k \leq n} \sum_{j=1}^k (X_j - \bar{X}_n) \right] \quad \text{---[1]}$$

Hurst found the following simple relationship represented the ‘expectation’ of the R/S statistic value well:

$$E\left[\frac{R(n)}{S(n)}\right] = an^H \quad \text{as } n \rightarrow \infty \quad \text{---[2]}$$

Where a is a constant and H is the Hurst parameter/value [Le194].

By sequentially varying the n sample number (also referred to as the sample size), one can calculate a corresponding R/S statistic. The following transformation is performed, on the above equation, prior to the estimation of the Hurst parameter:

$$\log\left(E\left[\frac{R(n)}{S(n)}\right]\right) = \log(a) + H \log(n) \quad \text{---[3]}$$

By plotting the log of the R/S value against the log of n , and estimating the slope of the relationship via regression analysis, one can obtain the H value for a particular trace. Self-similar or persistent behaviour is generally characterised by a Hurst value in the range of $0.5 < H \leq 1$. Non-persistent behaviour is characterised by a Hurst value in the range of $0 < H \leq 0.5$ [Naw95].

A confidence level for the regression estimate was also generated through a measure of “goodness-of-fit of linear regression” (which is denoted by r^2). The value of r^2 ranges between 0.0 and 1.0, and is a fraction-measure of the goodness-of-fit. It has no units. An r^2 value of 0.0 equates to a random variation of the y values in relation to the x values (i.e. the scatter plot values do not fit the regression line at all and there is no confidence in the prediction of y values based on x values and the regression line). In this case the ‘best-fit’ regression line is a horizontal line drawn through the mean of the y values. An r^2 value of 1.0 equates to a scatter plot where all the point lie precisely on a straight line with zero residual error (i.e. there is 100% confidence in the prediction of y values based on x values and the regression line) (Figure 6.3.1.a) [Mot03].

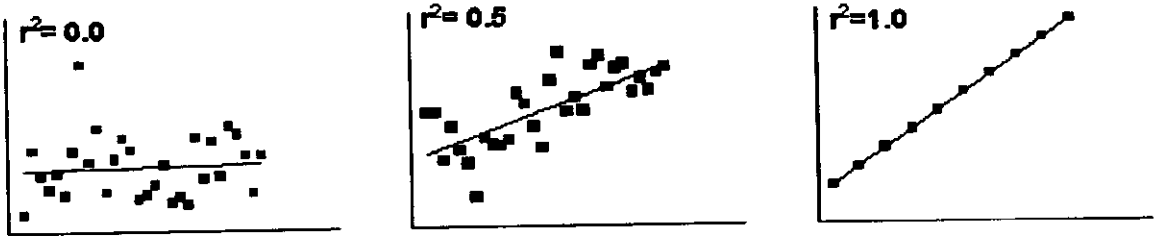


Figure 6.3.1.a: Examples of varying goodness-of-fit values (r^2) for regression lines

$$r^2 = 1 - \frac{SS_{\text{reg}}}{SS_{\text{tot}}} \quad \text{---[4]}$$

where

$$SS_{\text{reg}} = \sum_{i=1}^N (y_i - y'_i)^2 \quad \text{---[5]}$$

and

$$SS_{\text{tot}} = \sum_{i=1}^N (y_i - \bar{y})^2 \quad \text{---[6]}$$

$i = 1, 2, 3, \dots, N$ represents the x-axis points (i.e. log of sample size n), of the range of coordinates taken to calculate the regression line.

y_i is the actual log value R/S statistic calculated value for the x-axis point i .

y'_i is the regression line estimate of the x-axis point i .

\bar{y} is the mean of actual log values of the R/S statistic (y_i) of the range of coordinates taken to calculate the regression line. The MATLAB algorithm for the r^2 analysis is presented in Appendix i.

The RS statistic is benchmarked by calculating Hurst parameters for traces where the Self-Similarity is evident. The Vonkoch curve [MATb] is a prime candidate for a high Hurst value. In contrast, a randomly generated trace will generate a low Hurst value (Figure 6.3.1.b, Figure 6.3.1.c, Figure 6.3.1.d, Figure 6.3.1.e).

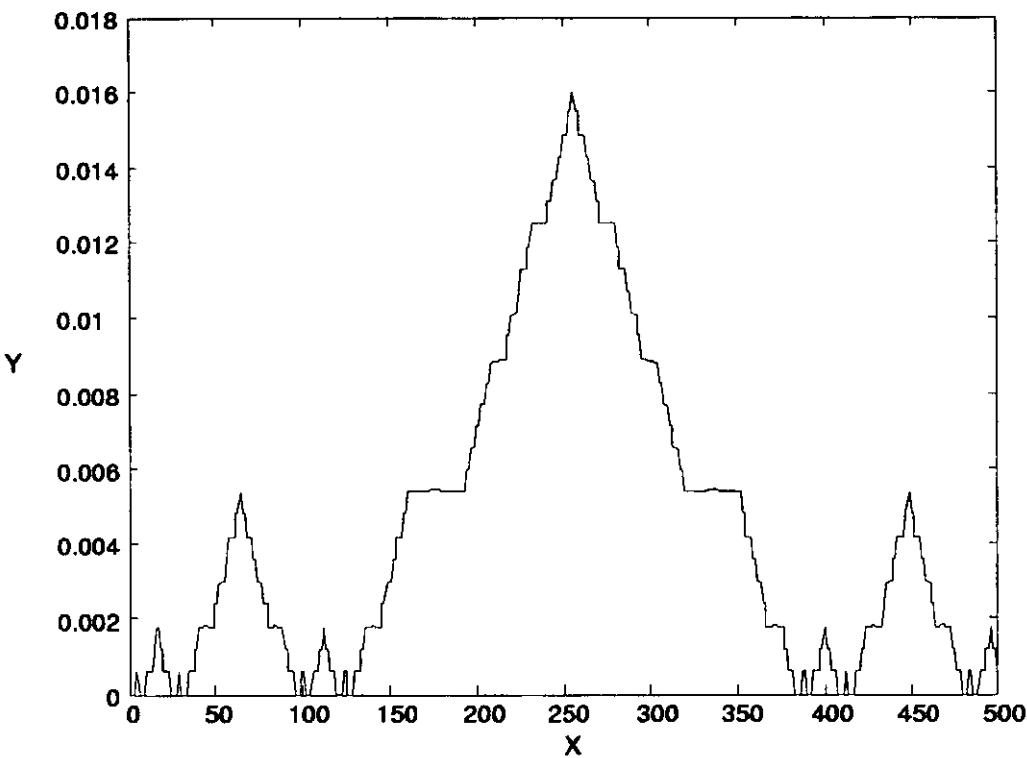


Figure 6.3.1.b: The Vonkoch self-similar curve

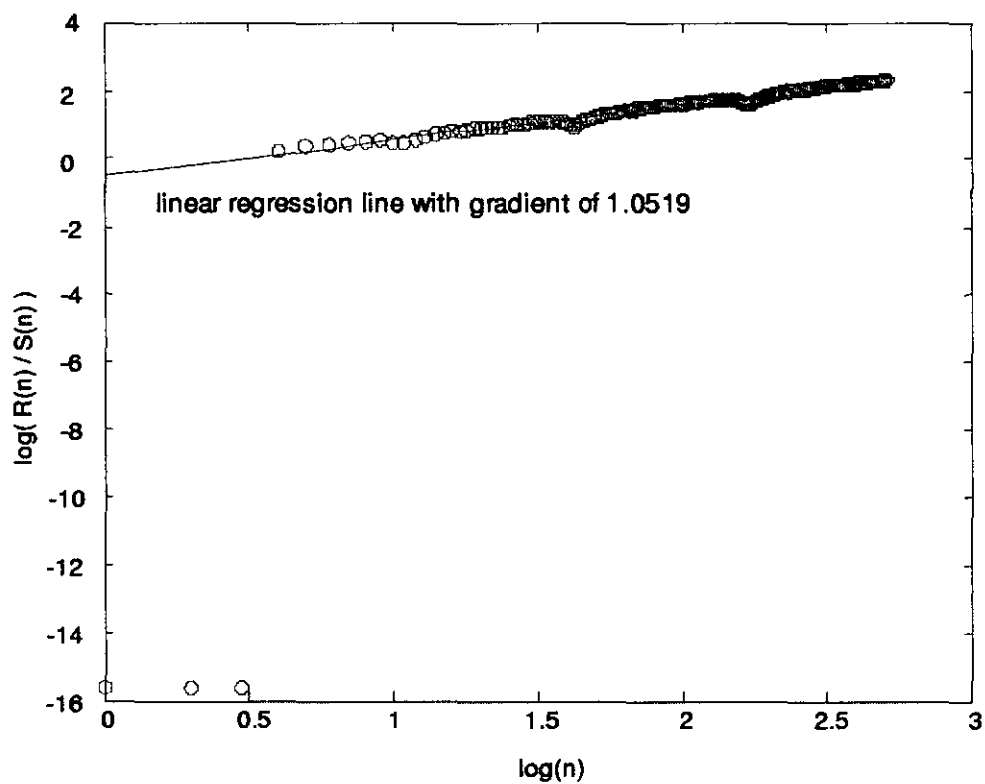


Figure 6.3.1.c: R/S statistic plot for the Vonkoch curve. Hurst value = 1.0519

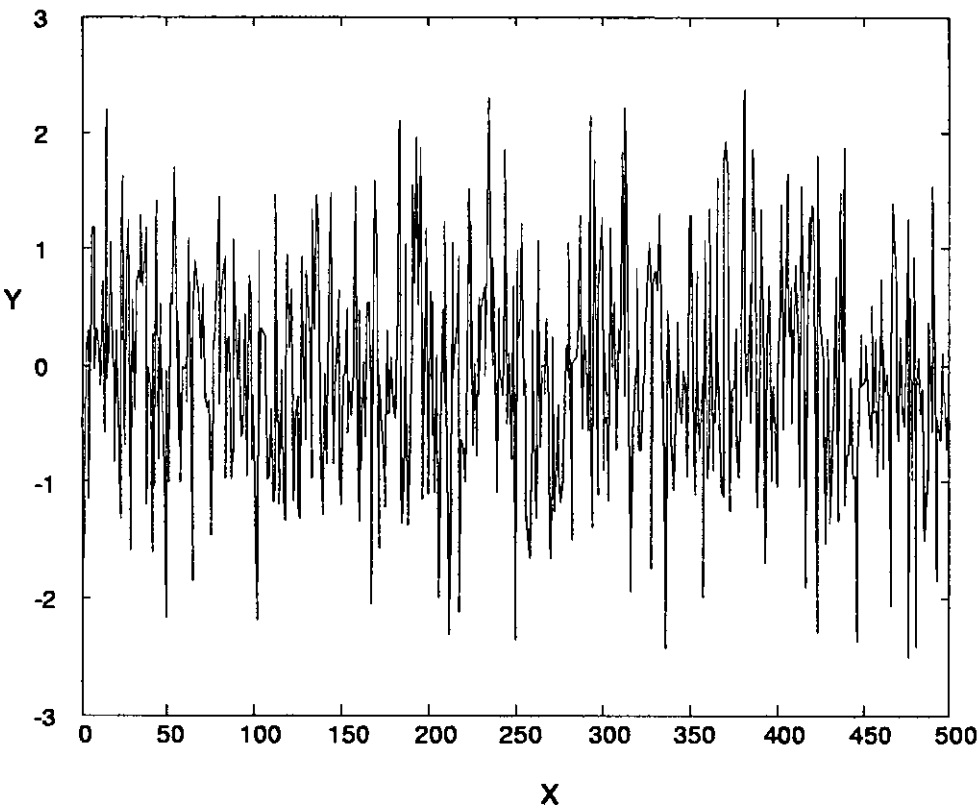


Figure 6.3.1.d: Random trace

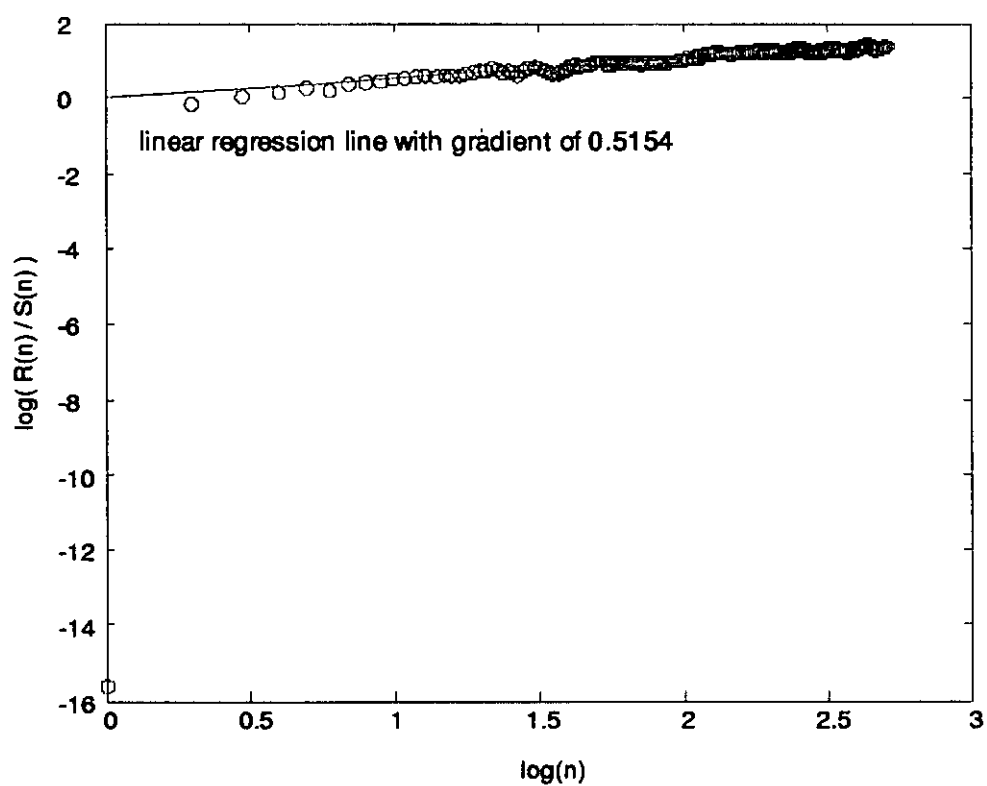


Figure 6.3.1.e: R/S statistic plot for the random trace. Hurst value = 0.5154

6.3.2 Approximation and Stability of the R/S statistic

The R/S calculations do not produce traces that are linear in an ideal sense and can be segmented into two distinct sections (Figure 6.3.1.c and Figure 6.3.1.e). The left most section (containing low and negative R/S values) reflects the peculiarity of the R/S calculation whereby the initial samples of the original trace are too few (i.e. a small n value) to make an accurate estimation (equation [1]) [Lel94]. Coupled with this the fact that there is an initial ‘transient’/startup phase within the Active network where the node resources are under-utilised; it leads to a situation where the R/S statistic values contribute little to the overall Hurst value calculation. The length of this ‘low-R/S value’ section (i.e. number of data points) will vary from node to node. In particular, the ‘transient’ phase is evident in scenarios that have been scrutinised and classified as Emergent by this work (i.e. during a ‘transient’ phase Active Replication Packets are **in the process of** stabilising resource usage to patterns that are self-similar – see proceeding section. It is possible that the initial samples of the R/S calculation, per Active Node, will reflect this factor by taking low values. It is also possible that the length of the ‘low-R/S value’ section, per Active Node, will also reflect the number of simulation time steps taken to stabilise and dominate resource utilisation). However whilst notable, the author has deliberately discounted the ‘low-R/S value’ section from further analysis.

The Hurst parameter may be subject to errors when calculated through regression analysis, as there is a possibility of the coefficients being biased by autocorrelation [Naw95]. Furthermore, graphical R/S analysis (regression analysis) is not accurate enough to calculate the Hurst value to 4 decimal places. Equation [2] notes that the relationship holds when $n \rightarrow \infty$. Therefore any sample size (n) with a finite upper bound would result in an approximation of the Hurst value.

As a result of autocorrelation, errors in graphical regression analysis and a finite sample number, the Hurst value is sometimes seen to rise above 1 (its theoretical limit). However, the author believes that these factors do not affect the validity of the results. The R/S calculation with regression analysis is a robust method used to determine whether Self-Similarity is supported by the data; subsequently used to calculate an empirical ‘estimate’ of the Hurst value. The analysis is formed on this premise and

focuses on nodes displaying a Hurst value above 0.9, which in turn, has proved to be an 'indicator' for the presence of Cascading Effects.

In Self-Similarity and Long-Range Dependence calculations the stability of the Hurst parameter can be subject to 'transient effects'. The calculation of the Hurst value is in effect the regression analysis estimation of the 'rate of change' of the R/S statistic. This rate of change estimation will inherently average out any transient changes (i.e. breaks and discontinuities in the R/S plot), the 'significant' of which will give rise to alternative Hurst values for the duration of the discontinuity. This may represent important phenomena within the system processes. To neglect these phenomena is to exclude a section of information that might point to further Emergence within the system.

The discontinuities in the plot can also be represented as errors above and below the linear regression line. It was noted however that the R/S calculations and regression analysis, for the results generated by this Active Network model, showed remarkably little error around the linear line of best fit (note: the analysis excludes the initial 'low-R/S value' section as mentioned above). The r^2 values for the linear regression lines calculated throughout this work (chapter 6; section 6.3.1) showed high values, of which a majority are above 0.9. This proves that the simulation scenarios developed by this project do not give rise to transitory R/S value variations and ensures the stability of Hurst values for Active Node resource usage fluctuations.

The lack of transitory effects can be linked to the particular simulation configuration used in this project. As mentioned in chapter 4; section 4.3.14, the inputs to the network simulation consisted of a 'single shot' of Active Packets/Applications that were allowed to traverse the network and interact with each other through the competition for resources. Any persistent structure that manifested within the simulation is solely caused by the 'single shot' of input packets. Similarly the lack of persistent data patterns within node resources was also unaffected due to the non-continuous nature of the input scenario. This has a positive effect on the accuracy of the linear regression analysis by eliminating any potential transitory changes in the R/S values, thereby giving an accurate and stable Hurst value for each Active Node.

As a result of this analysis the author has concluded the following:

- The contribution of the ‘single shot’ input scenario to Self-Similarity in resource usage is unique and interdependent.
- A clear ‘cause and effect’ relationship can be identified between the simulation inputs and Self-Similarity.
- The estimation of the Hurst parameter is made accurate by the input conditions.

6.3.3 Cascading Effects and Self-Similarity

The detection process for Cascading Effects is linked to the network exhibiting high levels of Self-Similarity (i.e. with Hurst values above 0.9). This result was discovered when the network was forced into a potentially ‘uncontrolled’ state whereby an Active Packet was injected (along with a representative sample of non-replicating Active packets), which replicated itself at every Active Node it encountered that had adequate resources (Figure 5.5.6.b). If it could not find adequate resources (due to other packets streams taking up resources or other replicating packets of the original taking up resources), it would progress onto the ‘next hop’ node (and so on until it encountered a node with adequate resources to process or an end-station). Because of replication the original direction may ‘span’ into multiple directions as the simulation progresses.

The result of this type of replication is the specific fluctuations, of resource usage, cascading throughout the network. The post simulation analysis of one type of resource (MEMORY usage), per Active node, indicated a high degree of Self-Similarity (i.e. a Hurst value of above 0.9) in a number of nodes.

The threshold value of 0.9 was based on the empirical evaluations of several predetermined simulation scenarios. The iterative process used in section 6.2 was reused in order to arrive at this value. This process is diagrammatically represented in Figure 6.3.3.a.

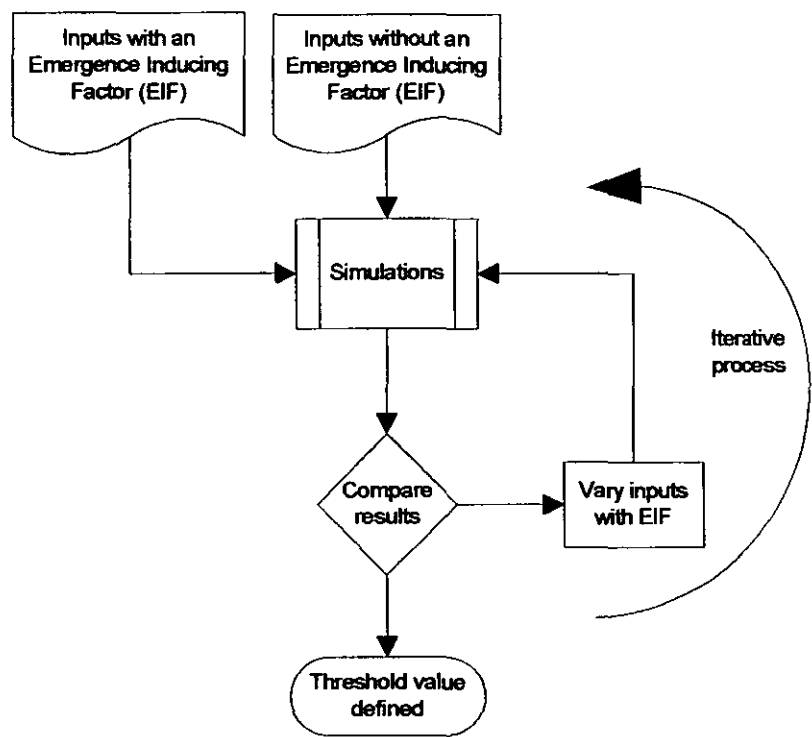


Figure 6.3.3.a: Process of experimentation and the definition of the "Cascading Effect" threshold of 0.9

As a consequence of the investigation into Self-Similarity, it was possible to link the two phenomena; levels of fluctuation of resource usage to levels of Self-Similarity within the nodes. A positive relationship was also formed from the number of nodes displaying Self-Similarity (above 0.9) and the effects of the cascade within the network.

6.4 Summary

This chapter describes the process by which the techniques for the detection of Emergence were discovered, selected and developed for the existing Petri Net model of an Active Network.

As can be seen from this chapter much of the thought process for the selection of a suitable technique was based on its ability to handle simple, discrete event-driven systems and their results. Thus some of the reviewed techniques were discounted as being useful (individually), even though they provided elegant solutions to the detection of anomalous behaviour and Emergence. The 3 broad methodologies mentioned in this chapter (Emergence as the loss of complexity in a system, Emergence through the measure of self organisation and Emergence as patterns in a system) have, however, similar cross-technological concepts such as:

- Resource usage is an indicator of anomalous behaviour
- Self-organisation is a loss of complexity
- Loss of complexity can be visualised as the formation of patterns
- Emergence is a particular subset of patterns (i.e. anomalous and unexpected patterns)

Therefore these concepts were useful in strengthening the case for the selected detection technique.

It was concluded that Emergent Behaviour could be isolated simply through the detection of patterns in the data produced by the Active Network model. The subsequent research effort was then focused on pattern detection algorithms (e.g. cross-correlation) and techniques to re-structure and visualise data (e.g. Image enhancement, Fourier and Wavelet analysis).

This chapter has described the development of a suitability test that was used in the assessment of the reviewed methods. The strategy was based on the premise that the inclusion of an Emergence Inducing Factor in Petri-Net simulations and the use of the generated results, as an input to a potential detection process, provides a means of assessing suitability. Therefore the experimentation of all pattern detection techniques was conducted concurrently using the same input data in accordance with the pre-

developed strategy. Each experiment was analysed for effectiveness based on the clarity of results and the ability to identify anomalous behaviour (Emergent or otherwise).

The development of algorithms for the experiments was achieved entirely through MATLAB, which contained many of the required mathematical functions and tools as single pre-built commands.

The Fourier and Wavelet analyses for these experiments did not provide readily distinguishable Emergence metrics for examination (i.e. detected no significant/distinctive changes in the simulation output results when presented with 2 significantly varied input data sets - one input set was perceived to have high probability of causing Emergence whilst the other was perceived to be relatively safe and 'Emergence free'). Image enhancement and edge-detection techniques provided no visual improvements to the output data from the Petri Net simulations. Cross-correlation results lacked clarity and provided no indication of having 'picked up' any patterns in the output data sets. However, it is worth noting that the model may well hold a collection of Emergence Inducing Factors, which results in a varied set of Emergent Behaviour within the system. As a result one cannot totally discount the above-mentioned techniques in detecting new phenomena.

This chapter describes in detail the development process of the Self-Similarity algorithm. It also highlights the potential of obtaining definitive results, in the detection of an Emergent Behaviour, through this method (a behaviour that could not have been foreseen or detected by any of the other techniques reviewed).

Self-Similarity is measured through the Hurst parameter and whilst there are many mathematical algorithms developed to calculate the Hurst value, including Wavelets, only the R/S technique was suitable for the data produced by the Active Network model. This chapter also gives definition to the detected Emergent Behaviour (i.e. a Cascading Effect) including its identifying characteristics.

The proceeding chapter follows on to provide a more results-oriented case analysis of the Cascading Effect. It is worth noting that the results analysis and Emergence definition are part of a collective body of experimentation performed using the Petri-Net model/simulations and the R/S statistic algorithm (according to the process flow described in Figure 6.3.3.a).

C H A P T E R 7
R E S U L T S

7 Results

This chapter will present:

- Simulation results presented in terms of Case Studies.
- Simulation scenarios developed using Controlled and Random Inputs.
- The identification of the “Cascading Effect” within the results.
- The identification of a secondary Emergent Behaviour
- The identification of root causes of the “Cascading Effect”.
- The proposal that the presence of a “Cascading Effect” is Emergence within the network.

The key concepts in this chapter are: the “Cascading Effect”, secondary Emergence, case studies, 0.9 Hurst value threshold, r^2 “goodness-of-fit” value for regression analysis

7.1 Overview

The basic modelling concepts developed throughout this work have provided a high-level abstract definition of an Active Network. The Petri Net simulation process developed these models as workable diagrams that produced time-step data of resource usage in all Active Nodes within the network (a 25-node network with lattice type interconnections). This time-step data was used as an input to the Emergence analysis process. The intermediate results produced by the simulations are themselves abstract and suit the high-level Emergence analysis.

As a part of the simple derivation of Active Networks several modelling and simulation considerations were made. These modelling considerations are mentioned in chapter 4; section 4.3. In addition to these considerations several key points are highlighted as being relevant, during the simulation process, in order to arrive at a specific output resource usage map:

- As inputs, each end-station produces one Active Packet per simulation run. This Active Packet was redefined, for the purpose of this project, to represent an entire Active Application stream, its functionality and payload. Thus, with reference to the lattice structure model of an Active Network, the Petri Net simulator would inject 20 Active Packets/Applications into the core network per simulation run (Figure 4.2.a and Figure 5.5.2.a). This resulted in a 'single shot' simulation of Active Applications and their interactions. The 'single shot' simulation scenario was deemed prudent and in keeping with the abstract nature of the Active Network model. It provided a sufficiently simplified view of Active Packet interactions and network state at any given time step. This followed on to the use of a simplified Emergence detection process using Self-Similarity.
- The input into the simulator was a text file that contains 20 entries specifying Active Packet/Applications that would be injected into the core of the network (Figure 5.5.2.b). This file can be generated randomly or manually; specifying, in particular, the directions taken across the network by the packets, amount of resources consumed and the forwarding mechanisms (linked to packet type). Manual generation of the input text file was tightly controlled to incorporate an Emergence

Inducing Factor. This was, in part, previously used for the experimentation and suitability testing of potential Emergence detection techniques as described in chapter 6; section 6.2.

- The automated random generation of the input text file was used to add levels of uncertainty and unpredictability into simulation runs and thereby produce results that were credible under abstract modelling conditions.
- The simulation output results that were analysed for Emergence were the node MEMORY resource usage statistics in percentage terms.
- Each simulation run was 500 time-steps and the resulting output log consists of a 25-column by 500-row matrix of integers (of MEMORY resource usage).
- The Self-Similarity calculation (R/S statistic) was for each Active Node per simulation run and therefore used a column wise calculation/decomposition of the above matrix.

7.2 Results: Case Study Analysis

This work presents the observation and analysis of results as case studies that primarily include the Hurst parameter values of ‘Active Node resource usage fluctuations’ within an Active Network simulation. Case study 1, case study 2, case study 3 and case study 4 were obtained from simulation runs made using manually specified input source files. Subsequent case studies were obtained using randomly generated input source files.

The first 4 case studies were designed to specifically highlight the “Cascading Effect” with tightly controlled input variables. The case studies showed the changes in the “Cascading Effect” (i.e. the presence or absence of it) when subject to changes in specific input variables in one source input of a simulation - From the analysis in chapter 6, this work had initially identified an “Emergence Inducing Factor” in the form of a self-replicating Active Packet/Application. This packet was included in the source input file (Figure 5.5.2.b), as one source input, for the simulation runs that produced the graphs shown in the first 3 case studies below. Case study 4 was a control experiment where the Active Replication scheme was not included. Thus, the self-replication Active Packet/Application was subject to input variable changes in the first 3 experiments. This allowed the simulations to adjust the effects of the replication on the network and thereby vary the effects of the “Cascading Effect”.

The first 4 case studies provided a foundation for the analysis of subsequent simulation produced with randomly generated input sources. This foundation provided a behavioural template for the Hurst parameter and the “Cascading Effect” with a reduced set of variations in input criteria. The use of randomly generated sources will increase the levels of variation and unpredictability. However, a template of expected behaviour will provide focus for the analysis in order to detect “Cascading Effects”.

7.2.1 Case Study 1

With reference to Figure 4.2.a and Figure 5.5.2.a, the input source file (table below) for this case study was constructed in order to inject a self-replicating Active Packet into Active Node A5 (SRC_A5) that took an initial direction from Node A5 to E1 (Figure 5.5.2.c). This replication was directed into the core of network and spanned based on the scheme described in chapter 5 (Figure 5.5.6.b). The replicating packets consumed, for the duration of the process, 20% of MEMORY resources, 20% of PROCESSOR resources and 20% of BUFFER resources from each Active Node it encountered and had adequate resources.

Other inputs within the source file were designed not to contain any form of replication. Therefore the Emergence of the “Cascading Effect” was singularly linked to the self-replication scenario (i.e. the Emergence Inducing Factor). The Hurst (self-similarity) values and the r^2 confidence values for this case study are shown in the following graph:

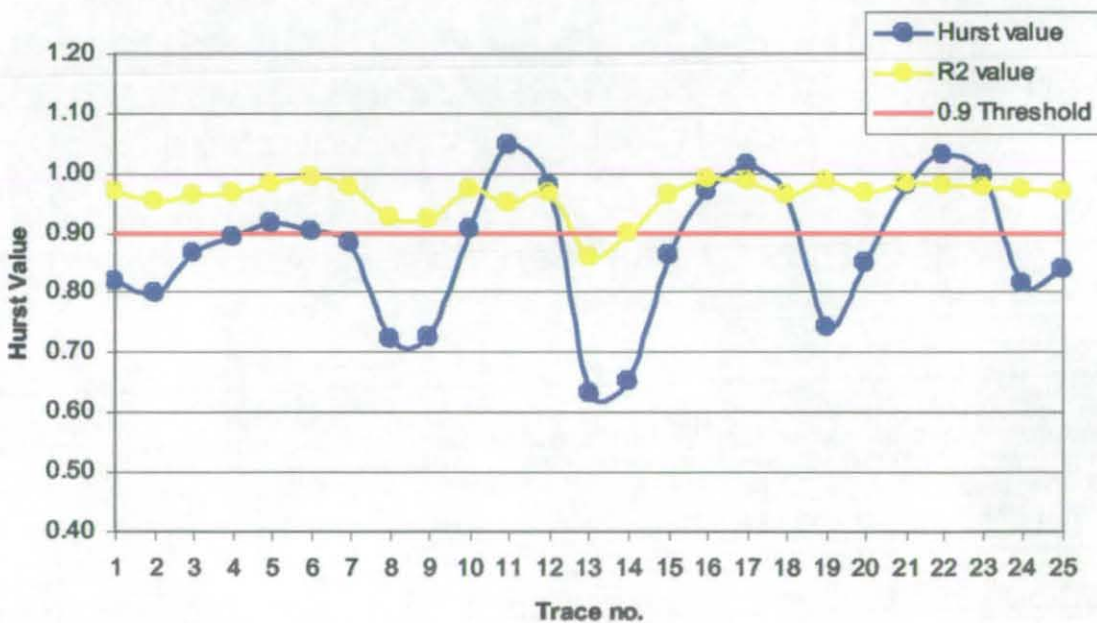


Figure 7.2.1.a: Case Study 1: Hurst analysis of Active Network

Input source file for the Case Study 1 simulation		
End-station no	End-Station name	Input packet
0	SRC A1	I'(2,4,3,50,M,20,10,10,10,15)
1	SRC A1A1	I'(2,4,3,50,M,20,10,10,10,15)
2	SRC B1	I'(2,5,5,50,M,20,10,10,10,15)
3	SRC C1	I'(6,5,5,1,F,30,30,30,0,0)
4	SRC D1	I'(6,5,5,1,F,30,30,30,0,0)
5	SRC E1	I'(2,6,7,50,M,20,10,10,10,15)
6	SRC E1E1	I'(6,5,5,1,F,30,30,30,0,0)
7	SRC E2	I'(6,7,7,1,F,30,30,30,0,0)
8	SRC E3	I'(6,7,7,1,F,30,30,30,0,0)
9	SRC E4	I'(6,7,7,1,F,30,30,30,0,0)
10	SRC E5	I'(6,8,7,50,F,60,60,60,30,5)
11	SRC E5E5	I'(6,1,1,1,F,30,30,30,0,0)
12	SRC D5	I'(6,1,1,1,F,30,30,30,0,0)
13	SRC C5	I'(6,1,1,1,F,30,30,30,0,0)
14	SRC B5	I'(10,1,1,50,F,60,60,60,0,0)
15	SRC A5	I'(8,2,1,1,R,20,20,20,0,0)
16	SRC A5A5	I'(6,3,3,1,F,30,30,30,0,0)
17	SRC A4	I'(6,3,3,1,F,30,30,30,0,0)
18	SRC A3	I'(6,3,3,1,F,30,30,30,0,0)
19	SRC A2	I'(6,3,3,1,F,30,30,30,0,0)

With reference to Figure 7.2.1.a 40% of the Active Nodes showed significantly high levels of Self-Similarity in their resource usage fluctuations (i.e. Hurst values above the 0.9 threshold). **The main observation of this work is that a “Cascading Effect” was present within this simulation instance of the network.**

The r^2 “goodness-of-fit” data for the simulation showed significantly high values for all of the Active Nodes (chapter 6; section 6.3.2). This provides a high degree of confidence in the accuracy of the Hurst value calculations for this simulation.

7.2.2 Case Study 2

With reference to Figure 4.2.a and Figure 5.5.2.a, the input source file (table below) for this case study was constructed in order to inject a self-replicating Active Packet into Active Node A5 (SRC_A5) that took an initial direction from Node A5 to E1 (Figure 5.5.2.c). This replication was directed into the core of network and spanned based on the scheme described in chapter 5 (Figure 5.5.6.b). The replicating packets consumed, for the duration of the process, 60% of MEMORY resources, 60% of PROCESSOR resources and 60% of BUFFER resources from each Active Node it encountered that had adequate resources.

Compared with Case Study 1 the only variables changed were the MEMORY, PROCESSING and BUFFER resource values for the Active Replication Packet. Other inputs within the source file were designed not to contain any form of replication. Therefore the Emergence of the “Cascading Effect” (or the lack of it) was singularly linked to the self-replication scenario (i.e. the Emergence Inducing Factor). The Hurst (self-similarity) values and the r^2 confidence values for this case study are shown in the following graph:

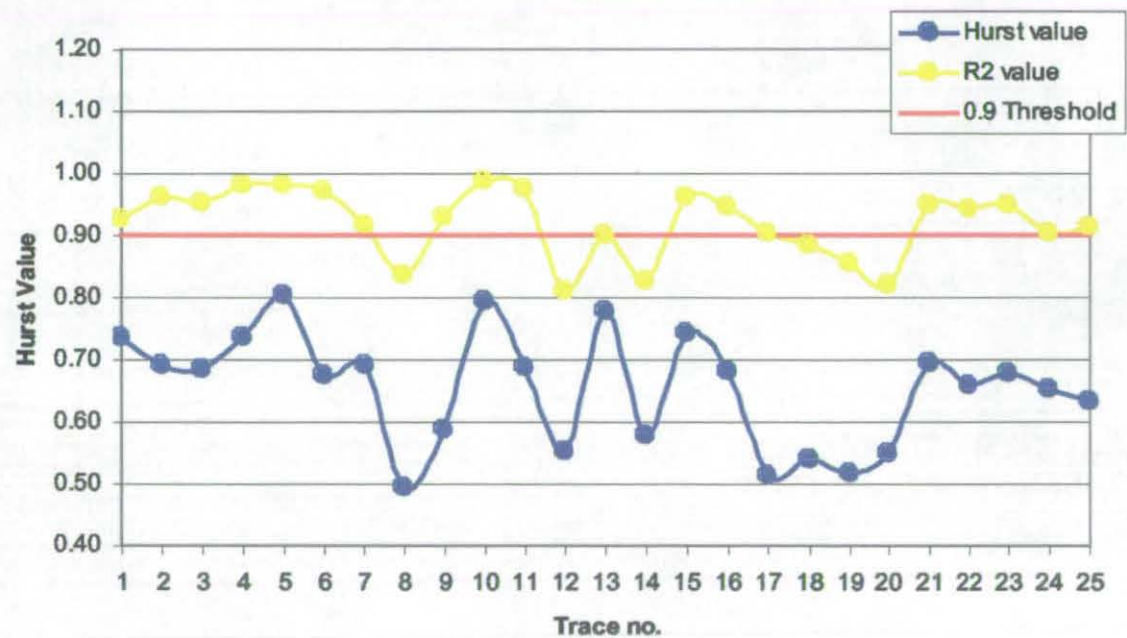


Figure 7.2.2.a: Case Study 2: Hurst analysis of Active Network

Input source file for the Case Study 2 simulation		
End-station no	End-Station name	Input packet
0	SRC A1	I'(2,4,3,50,M,20,10,10,10,15)
1	SRC A1A1	I'(2,4,3,50,M,20,10,10,10,15)
2	SRC B1	I'(2,5,5,50,M,20,10,10,10,15)
3	SRC C1	I'(6,5,5,1,F,30,30,30,0,0)
4	SRC D1	I'(6,5,5,1,F,30,30,30,0,0)
5	SRC E1	I'(2,6,7,50,M,20,10,10,10,15)
6	SRC E1E1	I'(6,5,5,1,F,30,30,30,0,0)
7	SRC E2	I'(6,7,7,1,F,30,30,30,0,0)
8	SRC E3	I'(6,7,7,1,F,30,30,30,0,0)
9	SRC E4	I'(6,7,7,1,F,30,30,30,0,0)
10	SRC E5	I'(6,8,7,50,F,60,60,60,30,5)
11	SRC E5E5	I'(6,1,1,1,F,30,30,30,0,0)
12	SRC D5	I'(6,1,1,1,F,30,30,30,0,0)
13	SRC C5	I'(6,1,1,1,F,30,30,30,0,0)
14	SRC B5	I'(10,1,1,50,F,60,60,60,0,0)
15	SRC A5	I'(8,1,1,1,R,60,60,60,0,0)
16	SRC A5A5	I'(6,3,3,1,F,30,30,30,0,0)
17	SRC A4	I'(6,3,3,1,F,30,30,30,0,0)
18	SRC A3	I'(6,3,3,1,F,30,30,30,0,0)
19	SRC A2	I'(6,3,3,1,F,30,30,30,0,0)

With reference to Figure 7.2.2.a none of the Active Nodes showed significantly high levels of Self-Similarity in their resource usage fluctuations (i.e. Hurst values above the 0.9 threshold). **The main observation of this work is that there wasn't a "Cascading Effect" present within this simulation instance of the network.**

The r^2 "goodness-of-fit" data for the simulation showed significantly high values for all of the Active Nodes (chapter 6; section 6.3.2). This provides a high degree of confidence in the accuracy of the Hurst value calculations for this simulation.

7.2.3 Case Study 3

With reference to Figure 4.2.a and Figure 5.5.2.a, the input source file (table below) for this case study was constructed in order to inject a self-replicating Active Packet into Active Node A4 (SRC_A4) that took an initial direction from Node A4 to B5 (Figure 5.5.2.c). This replication was directed away from the centre of the network and spanned based on the scheme described in chapter 5 (Figure 5.5.6.b). The replicating packets consumed, for the duration of the process, 20% of MEMORY resources, 20% of PROCESSOR resources and 20% of BUFFER resources from each Active Node it encountered that had adequate resources.

Compared with Case Study 1 the only element changed in the simulation was the initial input point (Node) and the direction of the Replication Packet.

Other inputs within the source file were designed not to contain any form of replication. Therefore the Emergence of the “Cascading Effect” was singularly linked to the self-replication scenario (i.e. the Emergence Inducing Factor). The Hurst (self-similarity) values and the r^2 confidence values for this case study are shown in the following graph:

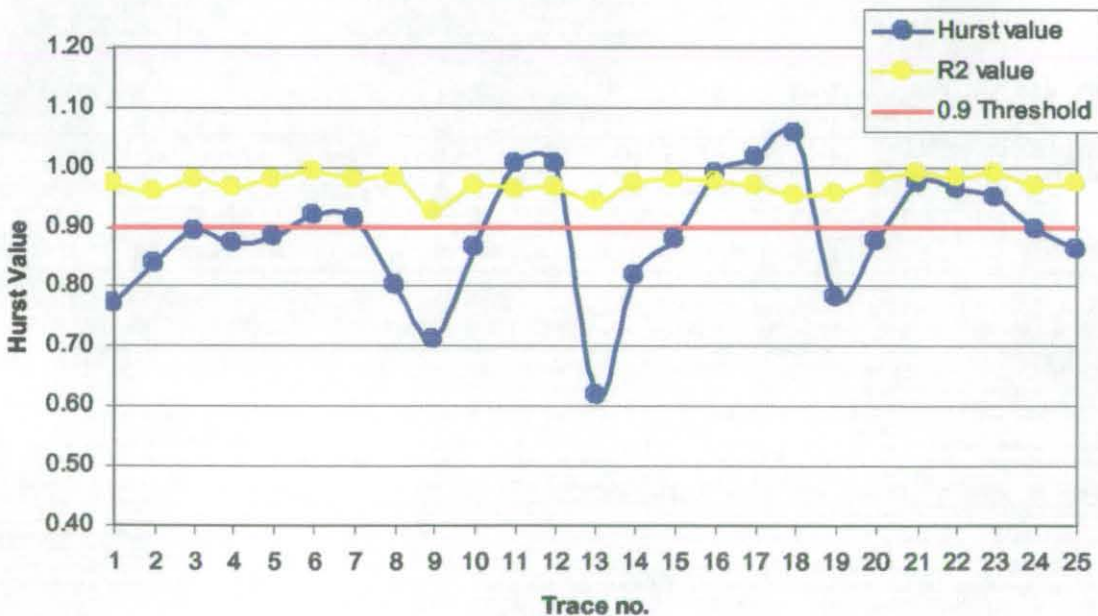


Figure 7.2.3.a: Case Study 3: Hurst analysis of Active Network

Input source file for the Case Study 3 simulation		
End-station no	End-Station name	Input packet
0	SRC A1	I'(2,4,3,50,M,20,10,10,10,15)
1	SRC A1A1	I'(2,4,3,50,M,20,10,10,10,15)
2	SRC B1	I'(2,5,5,50,M,20,10,10,10,15)
3	SRC C1	I'(6,5,5,1,F,30,30,30,0,0)
4	SRC D1	I'(6,5,5,1,F,30,30,30,0,0)
5	SRC E1	I'(2,6,7,50,M,20,10,10,10,15)
6	SRC E1E1	I'(6,5,5,1,F,30,30,30,0,0)
7	SRC E2	I'(6,7,7,1,F,30,30,30,0,0)
8	SRC E3	I'(6,7,7,1,F,30,30,30,0,0)
9	SRC E4	I'(6,7,7,1,F,30,30,30,0,0)
10	SRC E5	I'(6,8,7,50,F,60,60,60,30,5)
11	SRC E5E5	I'(6,1,1,1,F,30,30,30,0,0)
12	SRC D5	I'(6,1,1,1,F,30,30,30,0,0)
13	SRC C5	I'(6,1,1,1,F,30,30,30,0,0)
14	SRC B5	I'(10,1,1,50,F,60,60,60,0,0)
15	SRC A5	I'(6,3,3,1,F,30,30,30,0,0)
16	SRC A5A5	I'(6,3,3,1,F,30,30,30,0,0)
17	SRC A4	I'(8,4,3,1,R,20,20,20,0,0)
18	SRC A3	I'(6,3,3,1,F,30,30,30,0,0)
19	SRC A2	I'(6,3,3,1,F,30,30,30,0,0)

With reference to Figure 7.2.3.a 40% of the Active Nodes showed significantly high levels of Self-Similarity in their resource usage fluctuations (i.e. Hurst values above the 0.9 threshold). **The main observation of this work is that a “Cascading Effect” was present within this simulation instance of the network.**

The r^2 “goodness-of-fit” data for the simulation showed significantly high values for all of the Active Nodes (chapter 6; section 6.3.2). This provides a high degree of confidence in the accuracy of the Hurst value calculations for this simulation.

7.2.4 Case Study 4

With reference to Figure 4.2.a and Figure 5.5.2.a, the input source file (table below) for this case study was constructed as a control experiment that did not contained any self-replicating Active Packets.

Compared with Case Study 1 the only element changed in the simulation is the **lack** of Active Replication packet. The Hurst (self-similarity) values and the r^2 confidence values for this case study are shown in the following graph:

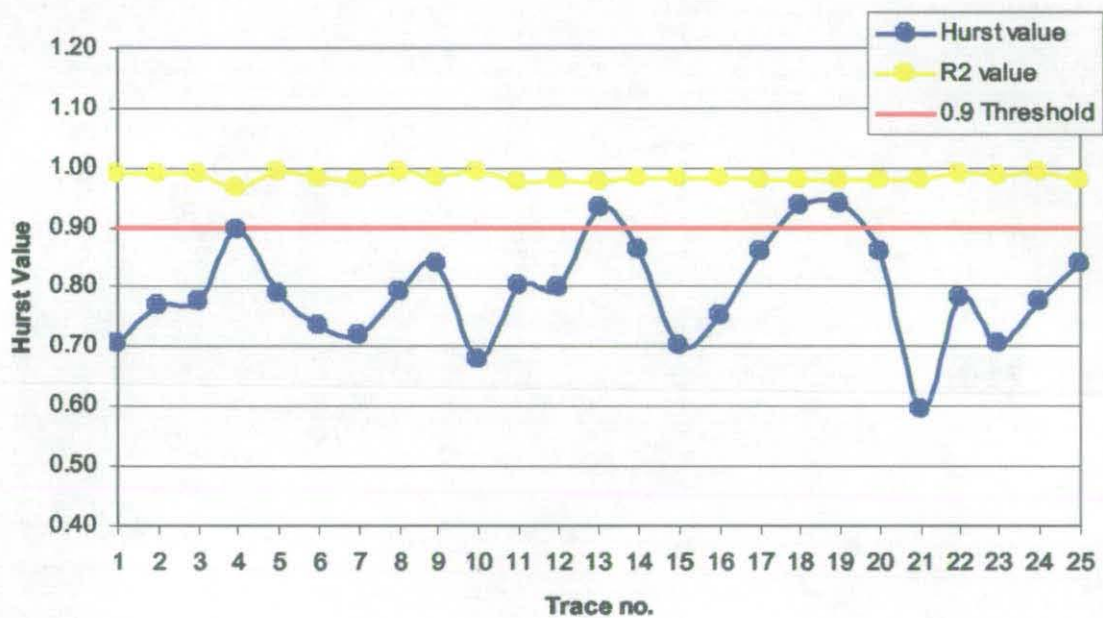


Figure 7.2.4.a: Case Study 4: Hurst analysis of Active Network

Input source file for the Case Study 4 simulation		
End-station no	End-Station name	Input packet
0	SRC A1	1'(2,4,3,50,M,20,10,10,10,15)
1	SRC A1A1	1'(2,4,3,50,M,20,10,10,10,15)
2	SRC B1	1'(2,5,5,50,M,20,10,10,10,15)
3	SRC C1	1'(6,5,5,50,F,30,30,30,0,0)
4	SRC D1	1'(6,5,5,50,F,60,60,60,0,0)
5	SRC E1	1'(2,6,7,50,M,20,10,10,10,15)
6	SRC E1E1	1'(6,5,5,50,F,30,30,30,0,0)
7	SRC E2	1'(6,7,7,50,F,30,30,30,0,0)
8	SRC E3	1'(6,7,7,50,C,30,30,30,0,0)
9	SRC E4	1'(6,7,7,50,F,30,30,30,0,0)
10	SRC E5	1'(6,8,7,50,F,60,60,60,30,5)
11	SRC E5E5	1'(6,1,1,50,F,30,30,30,0,0)
12	SRC D5	1'(6,1,1,50,F,30,30,30,0,0)
13	SRC C5	1'(6,1,1,50,F,30,30,30,0,0)
14	SRC B5	1'(10,1,1,50,F,60,60,60,0,0)
15	SRC A5	1'(6,1,1,50,F,30,30,30,0,0)
16	SRC A5A5	1'(6,3,3,50,F,30,30,30,0,0)
17	SRC A4	1'(6,3,3,50,C,30,30,30,0,0)
18	SRC A3	1'(6,3,3,50,F,30,30,30,0,0)
19	SRC A2	1'(6,3,3,50,F,30,30,30,0,0)

With reference Figure 7.2.4.a to only 12% of the Active Nodes showed significantly high levels of Self-Similarity in their resource usage fluctuations (i.e. Hurst values above the 0.9 threshold). **The main observation of this work is that there wasn't a "Cascading Effect" present within this simulation instance of the network.**

The r^2 "goodness-of-fit" data for the simulation showed significantly high values for all of the Active Nodes (chapter 6; section 6.3.2). This provides a high degree of confidence in the accuracy of the Hurst value calculations for this simulation.

7.2.5 Case Study 5

With reference to Figure 4.2.a and Figure 5.5.2.a, the input source file (second table below) for this case study was randomly generated with several self-replicating Active Packets that were injected into Active Nodes B1 (SRC_B1), E4 (SRC_E4), E5 (SRC_E5) and D5 (SRC_D5) (Figure 5.5.2.c).

The details of the initial directions, the resource usage requirement and the MEMORY reservation requirement for the Active Replication Packets/Applications are as follows:

Source	Injected Node	Initial Spanning Direction	Memory Resource requirement	Processor Resource requirement	Buffer Resource requirement	Memory Reservation and Time Limit
SRC_B1	B1	B1 to A2	85%	66%	61%	47% for 18 time steps
SRC_E4	E4	E4 to A4	79%	24%	55%	0%
SRC_E5	E5	E5 to A1	60%	55%	36%	49% for 13 time steps
SRC_D5	D5	D5 to C5	27%	58%	3%	13% for 40 time steps

Not all of the replications were directed into the core of the network. However, all replications spanned based on the scheme described in chapter 5 (Figure 5.5.6.b). The replicating packets consumed, for the duration of the process, the above-mentioned MEMORY, PROCESSOR and BUFFER resource values from each Active Node it encountered that had adequate resources.

Other inputs within the source file did not contain any form of replication. Therefore the Emergence of the “Cascading Effect” (or the lack of it) was linked to the self-replication scenarios (i.e. the Emergence Inducing Factor). The Hurst (self-similarity) values and the r^2 confidence values for this case study are shown in the following graph:

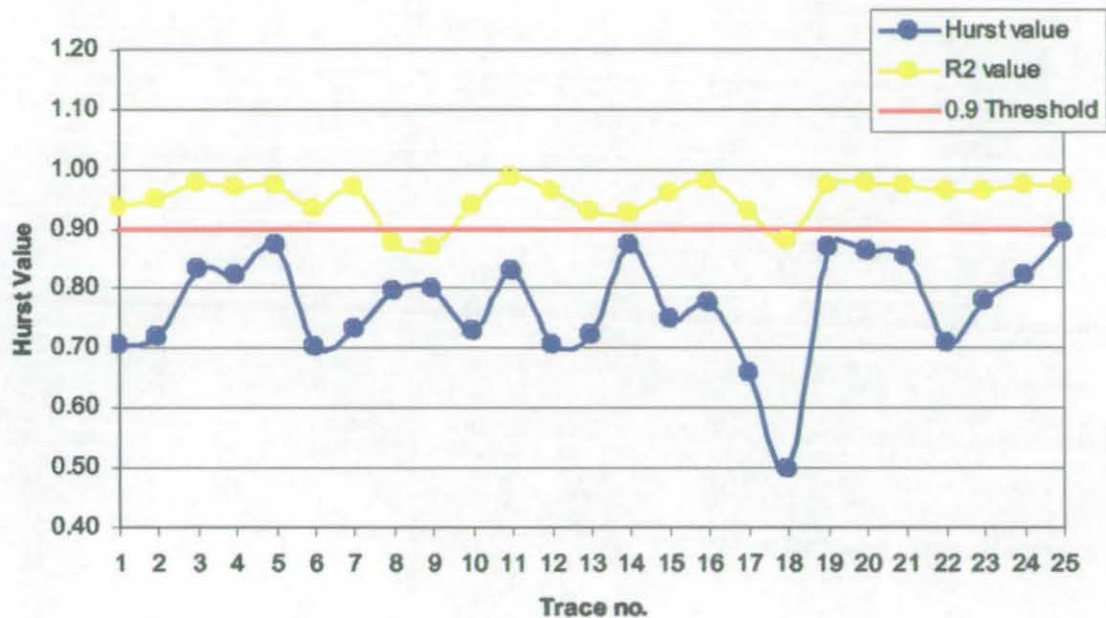


Figure 7.2.5.a: Case Study 5: Hurst analysis of Active Network

Input source file for the Case Study 5 simulation		
End-station no	End-Station name	Input packet
0	SRC A1	1^(4,8,7,1,M,30,10,10,20,15)
1	SRC A1A1	1^(2,2,3,1,M,20,10,10,10,15)
2	SRC B1	1^(13,6,5,4,R,85,66,61,47,18)
3	SRC C1	1^(10,4,5,45,F,51,97,41,37,8)
4	SRC D1	1^(2,4,3,1,M,20,10,10,10,15)
5	SRC E1	1^(14,6,5,21,C,67,12,93,25,16)
6	SRC E1E1	1^(12,6,7,48,F,72,2,6,0,23)
7	SRC E2	1^(9,7,7,26,C,79,40,85,63,5)
8	SRC E3	1^(14,2,3,8,C,94,89,19,14,30)
9	SRC E4	1^(12,7,7,1,R,79,24,55,0,23)
10	SRC E5	1^(9,8,7,17,R,60,55,36,49,13)
11	SRC E5E5	1^(15,7,7,6,C,88,33,29,8,2)
12	SRC D5	1^(12,6,5,0,R,27,58,3,13,40)
13	SRC C5	1^(7,3,3,28,C,31,77,72,16,24)
14	SRC B5	1^(4,1,1,1,M,30,10,10,20,15)
15	SRC A5	1^(7,3,3,28,C,45,25,71,23,49)
16	SRC A5A5	1^(12,7,7,26,F,28,73,4,14,48)
17	SRC A4	1^(3,5,5,1,M,20,20,20,0,0)
18	SRC A3	1^(4,6,5,1,M,30,10,10,20,15)
19	SRC A2	1^(2,8,7,1,M,20,10,10,10,15)

With reference to Figure 7.2.5.a none of the Active Nodes showed significantly high levels of Self-Similarity in their resource usage fluctuations (i.e. Hurst values above the 0.9 threshold). **The main observation of this work is that there wasn't a "Cascading Effect" present within this simulation instance of the network.**

The r^2 "goodness-of-fit" data for the simulation showed significantly high values for all of the Active Nodes (chapter 6; section 6.3.2). This provides a high degree of confidence in the accuracy of the Hurst value calculations for this simulation.

7.2.6 Case Study 6

With reference to Figure 4.2.a and Figure 5.5.2.a, the input source file (second table below) for this case study was randomly generated with several self-replicating Active Packets that were injected into Active Nodes E4 (SRC_E4), A5 (SRC_A5) and A3 (SRC_A3) (Figure 5.5.2.c).

The details of the initial directions, the resource usage requirement and the Memory reservation requirement for the Active Replication Packets/Applications are as follows:

Source	Injected Node	Initial Spanning Direction	Memory Resource requirement	Processor Resource requirement	Buffer Resource requirement	Memory Reservation and Time Limit
SRC_E4	E4	E4 to D5	26%	71%	2%	26% for 21 time steps
SRC_A5	A5	A5 to A5	43%	37%	69%	27% for 29 time steps
SRC_A3	A3	A3 to A5	7%	52%	98%	6% for 12 time steps

None of the replications were directed into the core of the network. However, all replications spanned based on the scheme described in chapter 5 (Figure 5.5.6.b). The replicating packet consumed, for the duration of the process, the above-mentioned MEMORY, PROCESSOR and BUFFER resource values from each Active Node it encountered that had adequate resources.

Other inputs within the source file did not contain any form of replication. Therefore the Emergence of the “Cascading Effect” (or the lack of it) was linked to the self-replication scenarios (i.e. the Emergence Inducing Factor). The Hurst (self-similarity) values and the r^2 confidence values for this case study are shown in the following graph:

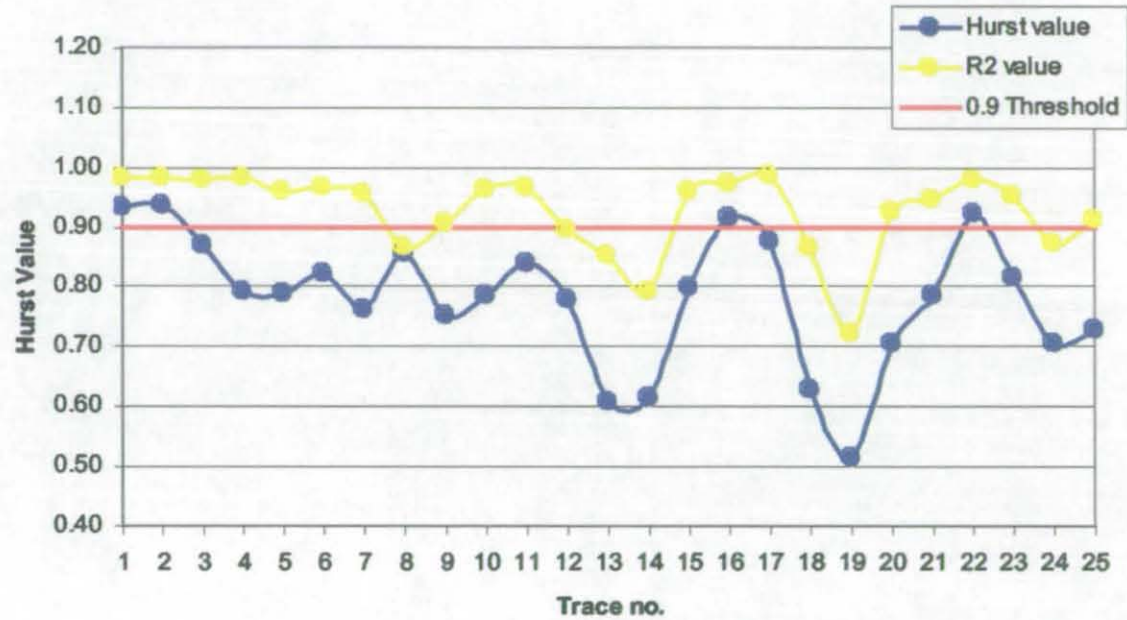


Figure 7.2.6.a: Case Study 6: Hurst analysis of Active Network

Input source file for the Case Study 6 simulation		
End-station no	End-Station name	Input packet
0	SRC A1	1'(7,1,1,8,C,93,38,19,60,4)
1	SRC A1A1	1'(1,1,1,1,M,10,10,10,0,0)
2	SRC B1	1'(15,3,3,5,C,86,31,25,5,0)
3	SRC C1	1'(4,2,1,1,M,30,10,10,20,15)
4	SRC D1	1'(7,3,3,11,C,48,94,24,9,32)
5	SRC E1	1'(5,2,3,1,M,40,40,40,25,10)
6	SRC E1E1	1'(11,1,1,17,F,24,5,0,11,13)
7	SRC E2	1'(7,4,5,11,F,49,44,25,10,7)
8	SRC E3	1'(6,2,3,5,C,88,83,63,7,2)
9	SRC E4	1'(11,6,7,25,R,26,71,2,26,21)
10	SRC E5	1'(2,4,3,1,M,20,10,10,10,15)
11	SRC E5E5	1'(6,2,1,6,C,38,84,14,23,2)
12	SRC D5	1'(14,7,7,0,F,27,22,17,27,22)
13	SRC C5	1'(10,8,7,12,F,66,97,42,24,9)
14	SRC B5	1'(8,8,7,6,F,3,85,29,1,2)
15	SRC A5	1'(6,7,7,8,R,43,37,69,27,29)
16	SRC A5A5	1'(5,3,3,1,M,40,40,40,25,10)
17	SRC A4	1'(5,6,5,1,M,40,40,40,25,10)
18	SRC A3	1'(11,5,5,41,R,7,52,98,6,12)
19	SRC A2	1'(2,4,3,1,M,20,10,10,10,15)

With reference to Figure 7.2.6.a 16% of the Active Nodes showed significantly high levels of Self-Similarity in their resource usage fluctuations (i.e. Hurst values above the 0.9 threshold). **The main observation of this work is that there wasn't a "Cascading Effect" present within this simulation instance of the network.**

The r^2 "goodness-of-fit" data for the simulation showed moderately high values for all of the Active Nodes (chapter 6; section 6.3.2).

7.3 Results Discussion

The graphical and tabulated (Appendix iv) results in the above case studies show that in some situations the network would consist of certain Active Nodes displaying Hurst values of above 0.9 (in resource usage fluctuations). **Where this is evident in a significant number of nodes, it has been found to indicate that Replicating Packets were producing a “Cascading Effect”.**

The graphs show the Hurst values of the Active Network simulated under a variety of input conditions.

The results show that nearly all nodes possess Hurst values that are significantly higher than that of randomly generated traces. This may be explained by the fact that Active Packets create deterministic resource usage in nodes as opposed to random patterns (for which Hurst values are about 0.5). Where these high values reach above the critical threshold of 0.9, the significance of which can be attributed to high resource fluctuations in nodes, there are grounds for further investigations regarding the types of Active Applications/Packets processed and the input scenario for the simulation.

Upon analysis of the simulation results (in the case studies), factors have been identified that significantly affect the Replicating Active Application/Package's ability to exhibit Self-Similarity and thus create a “Cascading Effect”. These factors are:

- The amount of resources an Active Replication Packet consumes within the node
- The number of Active Replication Packets initially injected into the network
- The amount of time for which the replication packets reserve resources (as part of the resource reservation feature of Active Networks).

If an Active Replication Packet requires a large percentage of node resources, we can expect a situation where the initial replicating packets very quickly consume the majority of the node resources. It is likely that replicated packets further down the process line (requiring the same amount of resources, from each node they encounter, as the original), will not have adequate resources to complete the Active process and pass through the network unaffected. The “Cascading Effect” would, in this situation, be self-limiting or fail to manifest itself. Furthermore, the Network would reach a point of

quasi-saturation in terms Active capability and performance. Case study 2 and case study 5 contain examples of this effect (Figure 7.2.2.a and Figure 7.2.5.a). The simulation in case study 2 was based on 1 Active Replication Packet being injected into the core of the network. The replications that followed required substantial amounts of node resources (i.e. 60% of each resource type) from each Active Node capable of processing them. The high resource requirements led to the network reaching the above-mentioned 'self-limiting' state. Similarly, the simulation in case study 5 held four initial replicating packets each requiring a combination of resources with a minimum of 58%. As a result of the high resource requirement and the number of individual Active replications present within the systems, the "Cascading Effect" failed to manifest. This was further compounded by the reservation of resources, for specific periods of time, by the replication packets. Case study 6 (Figure 7.2.6.a) showed a similar result to case study 5 (also with a randomly selected set of Active Packets/Applications as simulation inputs).

In comparison, the simulation in case study 1 had one Active Replication Packet/Application injected into the core that required a small amount of resources from each Active Node in the path of replication (i.e. 20% of each resource type). As a result, there was a significant number of Active Nodes displaying 'above 0.9' Hurst values in resource usage. A "Cascading Effect" was present within that instance of the system.

The 'general' path of the original Active Replication Packet, injected into the network, has **no effect** on the networks potential to exhibit a "Cascading Effect". It could be assumed that when the path is directed towards the centre and/or is directed along an edge dimension of the grid network, the number of nodes with 'above 0.9' Hurst values will be proportionally great. Case study 1 (Figure 7.2.1.a) presents this case where the initial path of the Active Replication Packet was directed towards the centre of the network. However, it is apparent from case study 3 (Figure 7.2.3.a) that, even when the path was directed away from the 'core', the number of affected nodes remains relatively stable.

It is worth noting that in some instances, without the presence of any Active Replication Packets in the network, a small number of Active Nodes will exhibit Hurst values of above 0.9. Case study 4 shows examples of this effect (Figure 7.2.4.a). Investigations revealed that these nodes were located in positions where they process an increased number of Active Packet/Applications (i.e. Active Application stream concentration points), thereby giving rise to high resource fluctuations and high Self-Similarity. For example, in case study 4, Active Nodes C3 (trace no. 13), B3 (trace no. 18) and B2 (trace no. 19) had a high concentration of Active Applications, which increased the amount of processing/memory handling tasks being performed (throughout the simulation at these nodes). The nodes processed in total 7, 6 and 7 Active Applications, respectively. This was reflected by above-0.9 Hurst values for these nodes. In comparison the Active Node A3 (trace no. 23) processed only 3 Active Applications for the entire duration of the simulation run. This resulted in a relatively low Hurst value of 0.7060. This phenomenon, on its own, was interesting and can be thought of as **secondary Emergence** (i.e. an additional facet of the discovered Emergence for this model).

7.4 Summary

In this chapter the author has re-emphasised the modelling considerations made with respect to the inputs/outputs of the system and the simulation. It proposes that the simplified inputs are integral to the high-level examination of an Active Network and in turn simplifies the process for the detection of Emergence.

The chapter also describes the structure of the input text file and the resulting output log file for a simulation run.

The second half of this chapter describes in detail various simulation ‘case studies’ that indicate the presence or absence of Emergence. The case studies are summarised as plots, which are also the results of the Self-Similarity calculation process – each simulation run produced one Emergence case study that consisted of 25 Hurst parameter values derived from the time-based Self-Similarity calculations of the 25 Active Nodes present within the system. Each case study included in the chapter provided indications of Emergence (if present) whilst the Results Discussion section (section 7.3) described the causal factors.

The chapter also describes the empirical process by which Emergence is identified - defining a key detection criterion (i.e. a Hurst value above 0.9 in a significant number of nodes). The detection criterion was established through the experimental case study analysis process, each of which reinforced the detection to a position where the potential Emergent outcome of the system could be ‘estimated’ prior to any simulation. The process of simulating random input scenarios, which contained (and didn’t contain) the Emergence Inducing Factor of self-replication, could have been continued indefinitely. However, it was apparent that case studies, after a certain number, added no exceptions to the Self-Similarity process or the Emergence criterion (for “Cascading Effects”). It was at this point that further simulations were deemed unnecessary.

C H A P T E R 8
C O N C L U S I O N S A N D
F U T U R E W O R K

8 Conclusions and Future Work

This chapter will present:

- A review of the Emergence definition and the Emergence detection criterion for the results produced by this work.
- The identification of the generic characteristics within the detected Emergent Behaviour.
- Further work in terms of enhancing the Active Network model.
- Further work in terms of developing alternative detection techniques
- Further work in terms of migrating the model to an entirely new methodology.

The key concepts in this chapter are: Emergence, Cellular Automata, cluster analysis, model enhancements

This work has attempted to identify Emergence as a measurable quantity in a highly connected network of 'intelligent' nodes. It has also succeeded in highlighting one particular instance of Emergence from a technique devised for the detection of patterns in data.

The conclusion of this work is that the Self-Similarity of resource usage fluctuations in Active Nodes (above a certain threshold) is an Emergence, by definition, and is a property that would not manifest/explain itself by the simple analysis of system dynamics - Emergence detection required a specific layer of abstraction of the network along with a specific detection algorithm.

Furthermore, it is apparent that even when self-replicating input scenarios were allowed to affect the network in an uncontrolled manner (a situation that could be perceived as detrimental to the network) they did not necessarily satisfy the Self-Similarity condition for Emergence. As a result, more attention was applied to the analysis of the dynamics of Active Replication Packets and other Active Packets. One can think of the lack of Self-Similarity condition, in these situations, is part of the overall Emergence within the system. This is because the phenomenon was not expected and required further investigation.

The root causes of the self-similar Emergent Behaviour were identified as the resource usage requirement of Active Replication Packets/Applications, the amount of time the resources are reserved as part of Global State Maintenance and the number of Active Replication Packets traversing across the network.

On reflection the root causes for the self-similar Emergent Behaviour seem obvious. However, the formal definitions of Emergence cater for the notion of 'hindsight' – the fact that Emergence is a behaviour that persists until a valid explanation is found, at which point it ceases to be Emergence.

The concern of this research work is the comprehension of the underlying dynamics of Active Networks, which would probably give rise to Emergence. Whilst this work was successful in proving Emergence exists in the Active Network model, it has highlighted only one particular example of Emergent Behaviour. It may be possible that the system holds many Emergent characteristics that could be discovered through other pattern detection techniques and algorithms. In general, there exists many types of Emergent

Behaviour; some closely coupled with system under investigation (not necessarily Active Networks related). The prime contribution of this work was to direct research towards the analysis of resources and resource usage within the system. The author proposes that there exists a strong link between resource usage in a distributed system and Emergent Behaviour. A 'clue' regarding Emergent Behaviour can be obtained by the analysis of fluctuations in resource usage - as they function as 'enablers' for the system's processes. The resource stores also function as 'points/places of contention' for system services thereby acting as reflectors of 'interesting' behaviour.

The modelling process accounts for the system being complex and being governed by theories of Complexity. Thus a systematic and piecemeal approach to the model construction had to be combined with a collective understanding of network behaviour. The author views the resultant 'collective behaviour' as patterns in the system. The algorithm developed to detect one type of Emergence is a step towards the understanding of all collective behaviours in the system.

Of the Emergent characteristics described in chapter 3 it is clear that the Self-Similarity Emergent Property is due to the creation of Positive Feedback structures (chapter 3; section 3.1.1). It is also apparent that the Emergent Property detected by this work is clearly linked with characteristics of Meta-balance (chapter 3; section 3.1.3) and Resonance (chapter 3; section 3.1.6).

Further work to this research could come in the form of extending the Active Network model in terms of additional elements, which would increase the accuracy of the model but would also add another layer of complexity (e.g. addition of variable time delays).

An extension to the model should be able to parameterise and simulate the concept of time (in a single step) as:

- Time taken for a packet to flow from one node to the next-hop node
- Time taken for a packet to move through the input/output queues of the current node
- Time taken for an Active Packet to be processed (code retrieval, verification, resource allocation and execution)

Time delays will provide an extra dimension to focus onto in order to highlight potential anomalous behaviour (i.e. Emergent Behaviour). It was decided that this research, in an

attempt to seek a balance between the development of a manageable system model and the analysis of an accurate/complex model, would disregard this element.

Additional techniques for the detection of Emergence, through the visualisation of resource usage fluctuations, can also be developed. The author proposes the use of Cluster Analysis Techniques to determine additional patterns in data that are indicative of Emergence.

Investigations of the “Cascading Effect” Emergence can be extended by incorporating alternate configurations of replicating schemes, increasing the number of Active Nodes being simulated and varying the topology of the network.

It may also be possible to migrate the modelling concepts to another methodology. For example, the author proposes the use of ‘Cellular Automata’, which would change the model from a static-node to a dynamic/virtual-node structure. Initial investigations on this concept have been positive and are described in section 8.1 .

The work done by this research can be considered as a precursor and a template for future detailed modelling of Active Networks (i.e. the detailed modelling process would be able to use the points highlighted by this research to build a better Active Network).

The results and findings of this project were presented at the “Multi-Service Networks Conference” (COSENNERS, 2004), held on the 8th of July in Abingdon, Oxford, UK.

8.1 Virtual Node Simulation

Observing Emergent Properties and self-organised structure is a complex task, considering that the 'structure' of Emergent Behaviour may not stay fixed to a set of nodes. It is possible that the structure may move within the boundaries of the modelled network. With this in mind and for the purposes of exposing Emergent Behaviour, one can remove the restriction of having Active Nodes in specific network topologies (even though, in practical terms, the Active nodes are in a fixed network).

Virtual Node Simulation (VNS) is a concept envisaged by the author, and may prove to be a viable addition to this work in terms of detecting Emergent Behaviour in Active Networks - VNS could be used to visually detect static or moving structures within the Active Network simulation environment. It can also facilitate the interpretation of these structures and the stages of structure formation.

VNS begins with the design of a cellular grid environment that depicts the scalable Active Network. End-stations are located at the periphery of the grid and each end-station has the capability to transmit and receive Active Packets/Applications (which are designated as moving blocks). The foundations for this type of simulation are taken from Cellular Automata (CA), which have been used in other research projects to successfully analyse various anomalous network behaviours.

At the heart of Cellular Automata, we consider the uniform lattice of cells to have local states, which are subject to a uniform set of rules. These rules drive the behaviour of the system, and in turn, set the particular state of a cell (i.e. the rules compute the next state of a cell as a function of its previous state and the states of surrounding cells). A moving block positioned within a cell is an abstraction of the state of that cell (i.e. the simulator displays the changing states of cells as moving blocks within the grid environment). An extension to this would be to allow the cells to preserve the history of state changes and calculate their next state based on it.

Cellular Automata describes the simulation environment in terms of a lattice and cells. The abstract view of changing states depicts the movement of packets within the lattice.

However, CA alone will not suffice in describing the complex behaviour of Active Networks because the Active Packets themselves are defined entities with specific objectives. The model must be in a position to describe the characteristics of packets along with the rule-set for each cell.

The full simulation framework will incorporate 'Multi-Agent' theory, which will allow the complete definition of an Active Packet (along with its behavioural dynamics). Each Active Packet will be represented by an autonomous 'Agent', which will be produced and consumed within the simulated space (i.e. within end-stations and the cellular lattice). Agents will carry complex rule-sets and objectives (they may also incorporate the static rule-set defined for the individual cells), and have sensors to perceive their local neighbourhood [Dij00]. A further ability would be for Agents to leave 'traces' of themselves at specific cells they visit, thereby affecting the local environment.

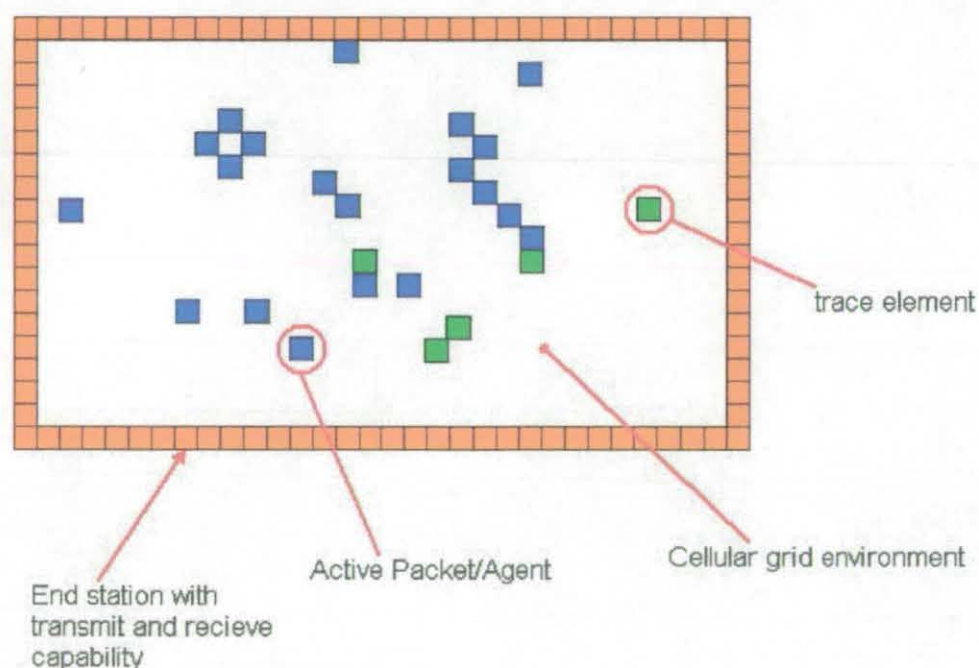


Figure 8.1.a: Virtual Node Simulation of an Active Network with a uniform lattice of cells

The Agent mobility and behavioural characteristics (anticipated or unplanned) are dependent on several factors:

- The static and dynamic goals of the Agent.
- The 'beliefs' of the Agent – a 'belief' is the internal, imperfect representation of the environment, held by the Agent, which includes the perceived states of other Agents.
- The rule-set
- The number of interactions between Agents
- The types of interactions between Agents (i.e. co-operative, competitive)
- The type of Agent

The concepts of Multi-Agents encompasses a large research area. Some of the concepts are either not relevant to Active Networks or are too strict in definition to be useful. It is therefore appropriate to reduce the Agent description to a minimum set of parameters.

One can define an Active Network Agent through $U = \langle R, A, F \rangle$, where:

- 'R' is a finite set of 'role identifiers'. It represents all possible roles (and combinations of which) an Agent can have. The 'role' of an Agent highlights the objectives and goals held within it.
- 'A' represents the activity agenda of an Agent i to achieve its goals $\{A_i\}$.
- 'F' represents the knowledge and information (Facets), held by an Agent, of its environment $\{F_i\}$. These facets include beliefs, awareness, experience, preference and choice. All of the facets are dynamic and are liable to change throughout a simulation [Dij00].

Agents (Active Packets) are programmed with certain capabilities, analogous to the Primitive Functional Components (PFCs) of the high-level modelling scenario. These capabilities are:

- Replicate themselves
- Merge into a single unit
- Transform themselves into new types
- Generate as new
- Leave trace elements at certain locations

- Harvest information from a local area and feedback to a calling application

The author proposes that Emergent Behaviour will not occur without the interaction of two or more Active packets and/or their trace elements.

An 'interaction' can be viewed, within the grid, as a collision between two or more packets and/or trace elements. At the point of impact, each Active Packet will execute certain rules and make decisions based on the local conditions (e.g. decisions based on collided packet type, trace element type, direction of movement, conflict in objectives, interest in co-operation, etc.).

In reality, Active Packets can only interact through a node; thus the point of impact is indicative of an Active Node being present at that location. In essence, all grid locations have the potential to be collision points and hence Active Nodes. However, the specific locations of Active Nodes are of little importance for a high-level simulation (i.e. the simulation should be independent of topology). The simulation should be capable of displaying Emergent Behaviour, some of which, will manifest themselves as stable, static/moving 'structures' (of Agents and their respective traces). Furthermore, through the rules governing the local state changes, a model can depict a global structure influencing local components (Figure 3.3.a).

The idea to use such an approach stems from the successes of 'Multi-Agents' in the detection and analysis of global phenomena in a wide variety of systems (e.g. traffic flow simulations, pedestrian behavioural analysis [Dij00]). However, currently this simulation method is in its initial investigation stages. Further work in this area may reveal the requirement of an application-specific simulator (which would have to be developed in-house through Object-Oriented programmes such as Java or C++).

R E F E R E N C E S

References

- [ACT] ACTIVATE Homepage, System Design Laboratory - Information and Computing Sciences (ICS) Division - SRI International,
<http://www.sdl.sri.com/projects/activate/>
- [Am94] Am, O., "Back to Basics: Introduction to Systems Theory and Complexity", Introductory web article, 1994,
<http://www.calresco.org/texts/backto.htm>
http://www.cuthb.plus.com/onar/back_to_basics.html
- [ANC] ANCORS Homepage, System Design Laboratory - Information and Computing Sciences (ICS) Division - SRI International,
<http://www.sdl.sri.com/projects/ancors/>
- [ANE] ANEP Homepage, Department of Computer and Information Science, University of Pennsylvania,
<http://www.cis.upenn.edu/~switchware/ANEP>
- [AWC] Adaptive Web Caching (AWC) Homepage, Computer Science Department - University of California, Los Angeles,
<http://irl.cs.ucla.edu/AWC/>
- [Bar97] Bar-Yam Y., "The Dynamics of Complex Systems (the advanced book: Studies in Nonlinearity series)", Addison Wesley Longman, August 1997, Chapter 0, Section 0.2 – 0.5, ISBN: 0-201-55748-7.
- [Ber00] Berson S., Braden B., Riciulli L., "Introduction to the Abone", Technical report, University of Southern California, Information Science Institute, June 2000, <http://www.isi.edu/abone/DOCUMENTS/ABoneIntro.pdf>
- [Bha97] Bhattacharjee S., Calvert K., Zegura E., "Active Networking and the End-to-End Argument", ICNP'1997: Proceedings of International Conference on Networking Protocols, Atlanta, Georgia, 28 – 31 October, 1997.

- [Bha98] Bhattacharjee S., Calvert K., Zegura E., "Congestion Control and Caching in CANES", ICC'1998: Proceedings of IEEE International Conference on Communications, Atlanta, Georgia, 7-11 June, 1998.
- [Bon97] Bonabeau E., Dessalles J. L., "Detection and Emergence", *Intellectica*, 1997/2, 25, pp. 85-94.
- [Bos99] Bossomaier T., Green D., "Patterns in the Sand: Computers, Complexity, and Everyday Life", Perseus Books, November 1, 1999, ISBN: 0-7382-0172-3.
- [Bra02] Braden B., Lindell B., Berson S., Faber T., "The ASP EE: An Active Network Execution Environment", DANCE'2002: DARPA Active Networks Conference and Exposition, San Francisco, CA., June 2002.
- [Bro01] Brown I., "End-to-end security in active networks", PhD thesis, London University, September 2001.
- [Bus01] Bush S. F., Kulkarni A., "Thought Communication", GE Global Research, Technical report no. 2001CRD062, 04 September 2001, <http://www.crd.ge.com/cooltechnologies/pdf/2001crd062.pdf>
- [Cam00] Campbell R. H., Liu Z., Mickunas M. D., Naldurg P., Yi S., "Seraphim: Dynamic Interoperable Security Architecture for Active Networks", OPENARCH'2000: Proceedings of IEEE Third Conference on Open Architectures and Network Programming, Tel Aviv, Israel, March 2000.
- [Cha] Chang S. K., Petri-Net and Augmented-Petri-Net, Course notes on Distribute Multimedia Systems, Department of Computer Science, Centre for Parallel, Distributed and Intelligent Systems (CPDIS), University of Pittsburgh, <http://www.cs.pitt.edu/~chang/365/2petri/p1.htm>

- [Cha97] Chapman N., "Petri Net Models", SURPRISE' 1997: Surveys and Presentations in Information Systems Engineering, Department of Computing, Imperial College London, May-June 1997, Stochastic Models of Manufacturing Systems - article 2.
http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol2/njc1/
- [Cru90] Crutchfield J. P., "Chaos and Complexity", Handbook of Metaphysics and Ontology, Philosophia Verlag, München, 1990.
- [Cru02] Crutchfield J. P., "What Lies Between Order and Chaos?", Art and Complexity by J. Casti (editor), Oxford University Press, 2002.
- [DAR] Defence Advanced Research Projects Agency (DARPA) - Advanced Technology Office (ATO) official website on Active Networks,
<http://www.darpa.mil/ato/programs/activenetworks/actnet.htm>
- [Dau00] Dautenhahn K., "Reverse Engineering of Societies - a biological perspective", AISB'2000: Proceedings of the symposium on Starting from Society - the Application of Social Analogies to Computational Systems, Birmingham, UK, 2000, pp. 15-20, ISBN: 1-902956-13-8.
- [Dec99] Decasper D., Parulkar G., Choi S., DeHart J., Wolf T., Plattner B., "A Scalable, High Performance Active Network Node", IEEE Network Magazine, January/February 1999.
- [Des] Design/CPN homepage, Department of Computer Science – DAIMI, University of Aarhus, Denmark, <http://www.daimi.au.dk/CPnets/intro/>
- [Dij00] Dijkstra J., Timmermans H. J. P., Jessurun A. J., "A Multi-Agent Cellular Automata System for Visualising Simulated Pedestrian Activity", Proceedings on the 4th International Conference on Cellular Automata for research and Industry, Karlsruhe, Germany, 4-6 October, 2000, pp. 29-36.
- [EME] EMERGENCE Homepage, "Emergence: A journal of Complexity issues in Organisations and Management",
<http://emergence.org/old/Whyemergence.html>

- [Gel00] Gelas J. P., Lefèvre L., "TAMANOIR: A High Performance Active Network Framework, Active Middleware Services (AMS)", Kluwer Academic Publishers, ISBN 0-7923-7973-X, August 2000.
- [Gol99] Goldstein J., "Emergence as a Construct: History and Issues", *Emergence: A journal of Complexity issues in Organisations and Management*, 1999, vol. 1, issue 1, pp. 49 – 72.
- [Gon02] Gonzalez R. C., Woods R. E., "Digital image processing", Prentice Hall publications, 2002, ISBN: 0201180758
- [Gra95] Graps A., "An Introduction to Wavelets", *IEEE Computational Science and Engineering*, Summer 1995, vol. 2, number 2, published by the IEEE Computer Society.
- [Har96] Hartman J., Manber U., Peterson L., Proebsting T., "Liquid Software: A New Paradigm for Networked Systems", Technical Report 96-11, Department of Computer Science, University of Arizona, June 1996, <ftp://ftp.cs.arizona.edu/xkernel/Papers/tr96-11.ps>
- [Her97] Herrmann J. W., Lin E., "Petri Nets: Tutorial and Applications", Presentation slides presented at the 32th Annual Symposium of the Washington Operations Research - Management Science Council, Washington, D. C., November 5, 1997, <http://www.isr.umd.edu/Labs/CIM/miscs/wmsor97.pdf>.
- [Hic00] Hicks M., Nettles S., "Active Networking Means Evolution (or Enhanced Extensibility Required)", *IWAN'2000: Proceedings of Active Networks: Second International Working Conference*, Tokyo, Japan, October 16-18, 2000, pp. 16 – 32.
- [Hic01] Hicks M., Moore J. T., Nettles S., "Compiling PLAN to SNAP", *IWAN'2001: Proceedings of the IFIP-TC6 Third International Working Conference*, September/October 2001.

- [Hic98] Hicks M., Kakkar P., Moore J. T., Gunter C. A., Nettles S., "PLAN: A Packet Language for Active Networks", International Conference On Functional Programming, Baltimore, September 27 - 29, 1998.
- [Kat00] Katabi D., Wroclawski J., "A framework for Scalable Global IP-Anycast (GIA)", ACM SIGCOMM'2000: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Stockholm, Sweden, 28 August - 01 September 2000, pp. 3 - 15, ISSN:0146-4833.
- [Kul01] Kulkarni A. B., Bush S. F., "Active Network Management and Kolmogorov Complexity", OPENARCH'2001: Proceedings of IEEE Open Architectures and Network Programming, Anchorage Alaska, April 27-28, 2001.
- [Kul99] Kulkarni A., Minden G., "Active Networking Services for Wired/Wireless Networks", IEEE INFOCOM'1999: Proceedings of the Conference on Computer Communications, New York, USA, March 1999.
- [Le00] Le L., Sanneck H., Carle G., Hoshi T., "Active Concealment for Internet Speech Transmission", The Second International Working Conference on Active Networks, Tokyo, Japan, October 2000.
- [Lel94] Leland W. E., Taqqu M. S., Willinger W., Wilson D. V., "On the Self-Similar Nature of Ethernet Traffic (Extended Version)", IEEE/ACM Transactions on Networking, February 1994, vol. 2, no. 1, pp. 1 - 15, ISSN: 1063-6692.
- [Leo97] Leon M., "Research funded for 'active IP networks'", Article written for InfoWorld Electric, InfoWorld Publishing Company, April 28, 1997, vol. 19, issue 17.

- [Liu00] Liu Z., Naldurg P., Yi S., Campbell R. H., Mickunas M. D., "Pluggable Active Security for Active Networks", PDCS'2000: 12th IASTED International Conference on Parallel and Distributed Computing and Systems, Las Vegas, Nevada, November 6-9, 2000.
- [Lo91] Lo, A. W., (1991). "Long-Term Memory in Stock Market Prices", *Econometrica*, 1991, vol. 59, issue 5, pp. 1279 – 313.
- [Mar99] Marshall I. W., Roadknight C., "A new Approach to Active Network Management", Policy Workshop 1999, Hewlett-Packard's European research centre, Bristol, UK., 15 – 17 November 1999.
- [Mau02] Maughan D., Forward of conference proceedings, DANCE'2002: DARPA Active Networks Conference and Exposition, San Francisco, CA., May 29 – 30 2002, ISBN: 0-7695-1564-9.
- [MATa] MATLAB documentation on Wavelets and the Wavelet Toolbox
<http://www.mathworks.com/access/helpdesk/help/toolbox/wavelet/>
- [MATb] MATLAB documentation on Wavelets and Detecting Self-Similarity
http://www.mathworks.com/access/helpdesk/help/toolbox/wavelet/ch03_ap6.html#996832
- [Men99] Menage P., "RCANE: A Resource Controlled Framework for Active Network Services", IWAN'1999: Proceedings of the First International Working Conference on Active Networks, 1999, Springer-Verlag, vol.1653, pp. 25-36.
- [Mes99] Meseguer J., Talcott C., "MAUDE: A Wide-Spectrum Formal Language for Secure Active Networks", Slides presented at DARPA Active Nets Briefing, March 18, 1999.
- [Mot03] Motulsky H., Christopoulos A., "Fitting Models to Biological Data using Linear and Nonlinear Regression: A practical guide to curve fitting", 2003, Oxford University Press, ISBN: 0195171802

- [Mur97] Murphy D., "Building an Active Node on the Internet", M.Eng Thesis, Massachusetts Institute of Technology, June 1997.
- [Naw95] Nawrocki D. N., "R/S Analysis and Long Term Dependence in Stock Market Indices", *Managerial Finance*, 1995, vol. 21, no. 7, pp. 78-91.
- [Ohi98] Ohira T., Sawatari R., "Phase transition in computer network traffic model", *Physical Review E*, vol.58 (1998), pp.193 –195, SCSL-TR-98-009.
- [Pat81] Peterson J. L., "Petri Net Theory and the Modelling of Systems" Prentice-Hall Inc, 1981, ISBN: 0-13-661983-5
- [Pso99] Psounis K., et al., "Active Networks: Applications, Security, Safety and Architectures", *IEEE Communications Surveys*, First Quarter, vol. 2, no. 1, 1999.
- [Raz00] Raz D., Shavitt Y., "Active networks for Efficient Distributed Network Management", *IEEE Communications Magazine*, vol. 38, March 2000, pp. 138 – 143.
- [Rei85] Reisig, W., "Petri Nets – An introduction", Springer-Verlag, Berlin, 1985, pp. 1 – 14, ISBN: 0-387-13723-8 & 3-540-13723-8
- [SAN] SANE/OS Homepage, Department of Computer and Information Science, University of Pennsylvania,
http://www.cis.upenn.edu/~switchware/sane_os/
- [Sal84] Saltzer H., Reed D. P., Clark D., "End-to-end arguments in system design", *ACM Transactions on Computing Systems*, 1984, vol. 2, no. 4.
- [Sav96a] Savetz K., Randall N., Lepage Y., "MBONE: Multicasting Tomorrow's Internet", IDG publishing 1996, Chapter 1, ISBN: 1-56884-723-8.
- [Sav96b] Savetz K., Randall N., Lepage Y., "MBONE: Multicasting Tomorrow's Internet", IDG publishing 1996, Chapter 3, ISBN: 1-56884-723-8.

- [Sch00] Schwartz B., Jackson A., Strayer T., Zhou W., Rockwell D., Partridge C., "Smart Packets: Applying Active Networks to Network Management", *ACM Transactions on Computer Systems*, February 2000, vol. 18, issue 1, pp. 67 - 88.
- [Sch99] Schwartz B. I., "Sprocket Language Description for the Smart Packets Project", BBN Technical Memorandum No. 1221, September 27, 1999, <http://www.ir.bbn.com/documents/techmemos/TM1221.pdf>
- [See] Seel R., "Emergence in Organisations", Online article, New Paradigm Consulting homepage, <http://www.new-paradigm.co.uk/emergence-human.htm>.
- [Smi97] Smith J. M., Farber D. J., Gunter C. A., Nettles S. M., Segal M. E., Sincoskie W. D., Feldmeier D. C., Scott Alexander D., "SwitchWare: Towards a 21st Century Network Infrastructure", a White Paper, 1997, www.cis.upenn.edu/~switchware/papers/sware.ps
- [Sol01] Solé R. V., Valverde S., "Information Transfer and Phase Transition in a Model of Internet Traffic", *Physica A: Statistical Mechanics and its Applications*, 2001, 289 (3-4), pp. 595-605, ISSN: 0378-4371.
- [Sto00] Stone J., Partridge C., "When the CRC and TCP Checksum Disagree", *ACM SIGCOMM'2000: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, Stockholm, Sweden, 28 August – 01 September 2000, pp. 309 – 319, ISSN:0146-4833.
- [Ten96] Tennenhouse D. L., Wetherall D. J., "Towards an active network architecture", *Computer Communication Review*, April 1996, vol. 26, no. 2.
- [Ten97] Tennenhouse D., Smith J., Sincoskie D., Wetherall D., Minden G., "A survey of Active Network Research", *IEEE Communications Magazine*, vol. 35(1), January 1997, pp.80 – 86.

- [Tsc97] Tschudin C. F., "Active Network Overlay Network (ANON)", RFC Draft, December 1997, <http://abone.ifi.unizh.ch/~anon/anon-rfc.html>
- [Tul01] Tullmann P., Hibler M., Lepreau J., "Janos: A Java-oriented OS for Active Networks", IEEE Journal on Selected Areas of Communication, March 2001, vol. 19, no. 3.
- [Wet98a] Wetherall D. J., Legedza U., Gutttag J., "Introducing New Internet Services: Why and How", IEEE Network Magazine, July/August 1998.
- [Wet98b] Wetherall D., Gutttag J., Tennehouse D., "ANTS, A toolkit for building and dynamically deploying network protocols", OPENARCH' 1998: Proceedings of IEEE Open Architectures and Network Programming, San Francisco CA., 3 - 4 April, 1998.
- [Wri00] Wright W. A., Smith R. E., Danek M., Greenway P., "A Measure of Emergence in an Adapting, Multi-Agent Context", ISAB'2000: Proceedings Supplement, Massachusetts, 2000, pp. 20-27, ISBN: 0-9704673-0-3.
- [Yam00] Yamamoto L., Leduc G., "An Agent-inspired Active Network Resource Trading Model Applied to Congestion Control", MATA'2000: Proceeding of the second International Workshop on Mobile Agents for Telecommunication Applications, Paris, France, September 18-20, 2000.
- [Yem96] Yemini Y., da Silva S., "Towards Programmable Networks". IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, L'Aquila, Italy, October 1996.
- [Yua02] Yuan J., Mills K., "Exploring Collective Dynamics in Communication Networks", The Journal of NIST Research, March - April 2002, vol. 107, no. 2, pp. 179-191.

- [Yue03] Yuen E., Lo S., Jha S., "Clack: An active network platform", ICT2003: 10th International Conference on Telecommunications, 23 February - 1 March, 2003, pages 77- 84, vol.1.

A P P E N D I X I

i. MATLAB Algorithm

```
% MATLAB algorithm for the generation of the Hurst value (using the R_S statistic)
% and R2 confidence value
% for one series of data (i.e. one Active Node)
% notes: this programme must be run for each of the 25 traces in order to calculate
% the complete set of Hurst values for a particular simulation run
%-----
clear;

a1 = 0;          % declare and initialise dummy variable for fscanf

% user input for datalog filename for reading
file_nm = input('Enter file name >','s');
% user input for specific Active Node Trace no.
Trace_no = input('Enter the trace no (1 to 25) >');

% open datalog file
file_id = fopen(file_nm,'r');

% scan datalog file line-by-line (row-by-row) and enter into 2-d array
for n = 1:499
    [array(n,:),a1] = fscanf(file_id,'%i',[1 25]);
end

% close datalog file
fclose(file_id);

sig = (array(:,Trace_no)).';          % extract from 2-d array the correct trace linked to the
                                     % Active Node

L = length(sig);          % length of data trace
R_S = zeros(1,L);         % R_S value declared and initialised

% main calculation for-loop to calculate a series of R/S values for
% different sample lengths
for n = 1:L
```

```

sig_n = sig(1:n);           % obtain a sample series from original
                             % trace
mean_sig_n = mean(sig_n);   % mean of sample series
S(n) = sqrt(var(sig_n)) + eps; % square-root of the variance of sample
                             % series. eps value is used to prevent
                             % divide-by-zero errors

% nested for-loop to calculate partial sum
for k = 1:n
    W(k) = sum(sig_n(1:k)) - k*mean_sig_n;
end
% calculation of R/S statistic for the sample series
R_S(n) = (max([0 W]) - min([0 W]))/S(n);
end

% generating a log-log plot
X = log10(1:L);
Y = log10(R_S + eps);
figure;
scatter(X,Y);    % draw a scatter plot

% regression analysis
% extract the positive R/S values for the regression analysis
% the negative section represents the disregarded 'startup' stage
a2 = Y(Y>0);
ana_range = (L - length(a2) + 1):L; % calculate the regression analysis
                                     % range
p = polyfit(X(ana_range),Y(ana_range),1); % calculate the linear regression line
% refresh the scatter plot with added regression line
Y_reg = polyval(p,X);
plot(X,Y,'o',X,Y_reg,'-');

% print Hurst value for the trace linked to the Active Node, which it the
% gradient of the linear regression line
H = p(1)
% R2 "goodness of fit" calculation
% used to obtain a confidence level for the regression analysis
error = Y(ana_range) - Y_reg(ana_range); % error calculation for the
                                           % specific

```

```
% regression analysis range
ss_reg = sum(error.^2);           % sum of the errors squared
null_residual = Y(ana_range) - mean(Y(ana_range)); % null residual
ss_tot = sum(null_residual.^2);   % sum of the null residual squared

% print R2 "goodness of fit" value for the trace linked to the Active Node
r_squared = 1 - (ss_reg/ss_tot)
```

A P P E N D I X I I

ii. Design/CPN Petri Net Diagrams

ii.1. Declarations

// GLOBAL DECLARATIONS AND DEFINITIONS (COLOSETS) OF VARIABLES USED IN THE PETRI NET MODEL

// COLORSETS (variable type definitions) for Active Packets

color AppID = int with 1..15; // Active packet IDs

color Dir1 = int with 1..8; // Direction Indicator 1

color Dir2 = int with 1..8; // Direction Indicator 2

color TTL = int; // Time-to-Live field

color Route = with M | R | C | F; // Routing Mechanism

color Memory = int with 0..100; // Active Packet MEMORY requirement

color Processor = int with 0..100; // Active Packet PROCESSOR requirement

color Buffer = int with 0..100; // Active Packet BUFFER requirement

color Release = int with 0..50; // reserved MEMORY resource release time

```

// other COLORSETS (variable type definitions)
color Queue = with Q_unit;           // for a queue control variable
color Timer_Control = with Time_unit; // for a timer control variable
color Timestamp = int;                // for a timestamp variable
color Timeflag = bool;               // for a check flag variable

// compound COLORSETS (variable type definitions)
color Time_holder = product Timestamp * Timestamp * Timestamp * Timestamp * Timestamp declare all; // timestamp holder
color Packet = product AppID * Dir1 * Dir2 * TTL * Route * Memory * Processor * Buffer * Memory * Release declare all; // Active Packet
color Xpacket = product AppID * Route * Memory * Processor * Buffer * Memory * Timestamp * Timeflag declare all; // Active Packet with timestamp and check flag

// VARIABLE declarations
var pkt, pkt2 : Packet; // Active Packet variables
var source : Packet ms; // Input Source variable
var app, app2 : AppID; // Active Packet application ID.
var dk1 : Dir1; // Active Packet Direction Indicator 1
var dk2 : Dir2; // Active Packet Direction Indicator 2
var ttl: TTL; // Active Packet Time-to-live counter
var vroute, vroute2 : Route; // Active Packet Routing Mechanism
var vmem, vmem2, vmem3, vmem4, vmem5, vmem6, vmem7 : Memory; // MEMORY resource variables

```

```

var vprocess, vprocess2, vprocess3, vprocess4, vprocess5, vprocess6 : Processor;    // PROCESSOR resource variables
var vbuff, vbuff2, vbuff3, vbuff4, vbuff5, vbuff6 : Buffer;                        // BUFFER resource variables
var lstore, lstore2 : Memory;                                                     // reserved MEMORY resource variables
var rcount : Release;                                                             // reserved MEMORY resource release time

var q1, q2, q3 : Queue;                                                           // queue control variables
var t1, t2, t3, t4, t5 : Timer_Control;                                           // timer control variables
var tstamp : Timestamp;                                                           // timestamp variable
var tflag : Timeflag;                                                            // check flag variable

var compound : Xpacket;                                                           // Active Packet variable
var a5,b5,c5,d5,e5 : Timestamp;                                                  // timestamp variables
var a6,b6,c6,d6,e6 : Timestamp;                                                  // timestamp variables

```

```
// LOCAL DECLARATIONS AND DEFINITIONS (COLOSETS) OF VARIABLES USED IN THE MERGE PACKET COMPONENT
```

```
// COLORSETS (variable type definitions)
```

```
color Mtimestamp = int;    // for a timestamp variable
```

```
color Firststate = bool;   // for a check flag variable
```

```
color Flag = bool;         // for a check flag variable
```

```
// compound COLORSETS
```

```
color Flag_holder = product Flag * Flag * Flag * Flag * Flag;    // check flag holder
```

```
color Mpacket = product AppID * Dir1 * Dir2 * Mtimestamp * Firststate;    // reduced information Control Packet
```

```
// VARIABLE declarations
```

```
var mapp, mapp2, mapp3 : AppID;    // Active Packet application ID.
```

```
var mdk1, mdk1_2, mdk1_3 : Dir1;    // Active Packet Direction Indicator 1
```

```
var mdk2, mdk2_2, mdk2_3 : Dir2;    // Active Packet Direction Indicator 2
```

```
var mtstamp, mtstamp2, mtstamp3, mtstamp4 : Mtimestamp;    // timestamp variables
```

```
var fstatus, fstatus2, fstatus3, fstatus4 : Firststate;    // check flag variables
```

```
var a1, b1, c1, d1, e1 : Flag;    // check flag variables
```

```

var a2, b2, c2, d2, e2 : Flag;           // check flag variables
var a3, b3, c3, d3, e3 : Flag;           // check flag variables
var a4, b4, c4, d4, e4 : Flag;           // check flag variables
var mcompound : Mpacket;                  // reduced information Control Packet variable

// LOCAL DECLARATIONS AND DEFINITIONS (COLOSETS) OF VARIABLES USED IN THE LOCAL STORAGE COMPONENT

// COLORSETS (variable type definitions)
color Ltimestamp = int;                   // for a timestamp variable
color St = bool;                          // for a check flag variable

// compound COLORSETS
color Lpacket = product Ltimestamp * Memory * Release * St; // reduced information Control Packet

// VARIABLE declarations
var ltstamp : Ltimestamp;                 // timestamp variables
var lcompound : Lpacket;                  // reduced information Control Packet variable
var a, b : Memory;                        // reserved MEMORY resource variables
var GO, GO2 : St;                         // check flag variables

```

```
// LOCAL DECLARATIONS AND DEFINITIONS (COLOSETS) OF VARIABLES USED IN THE DATA LOGGING COMPONENT
// includes GLOBAL REFERENCE variables used in code sections

// COLORSETS (variable type definitions)
color reg_name = int;           // for page instance value (ACTIVE NODE IDENTIFIER)

// VARIABLE declarations
var reg,reg2,reg3,reg4,reg5 : reg_name;    // variables holding the ACTIVE NODE IDENTIFIER value

// GLOBAL REFERENCE VARIABLE declarations
globref mk_0 = "null";           // not used
globref mk_1 = "null";           // variable to hold all datalog register location values as one string; written to output file "out.txt"
globref tpage_id = 0;             // variable to hold user selected page handle (i.e. Active Node page handle)
globref tplace1_id = 0;           // variable to hold user selected place handle (i.e. Mregister place handle)
globref tplace2_id = 0;           // variable to hold user selected place handle (i.e. Global_Memory_Store place handle)
globref tplace3_id = 0;           // variable to hold user selected place handle (i.e. Global_Processor_Store place handle)
globref tplace4_id = 0;           // variable to hold user selected place handle (i.e. Global_Buffer_Store place handle)

// file handles for the reading in of custom initial MEMORY, PROCESSOR and BUFFER values.
globref fh1 = TextIO.openIn "/home/elmsd2/design_cpn/activenetwork/src.txt";
```



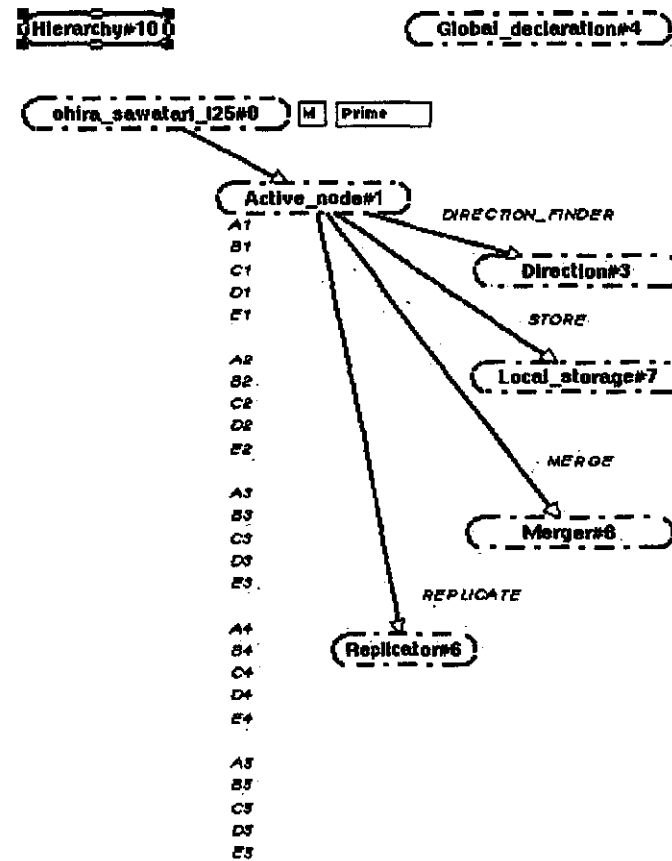
```
globref fh2 = TextIO.openIn "/home/elmsd2/design_cpn/activenetwork/src.txt";
globref fh3 = TextIO.openIn "/home/elmsd2/design_cpn/activenetwork/src.txt";

globref N = 0;                // while-loop counter
globref mem_val = "null"; // variable to hold the custom initial MEMORY value read in from file
globref pro_val = "null"; // variable to hold the custom initial PROCESSOR value read in from file
globref buf_val = "null"; // variable to hold the custom initial BUFFER value read in from file

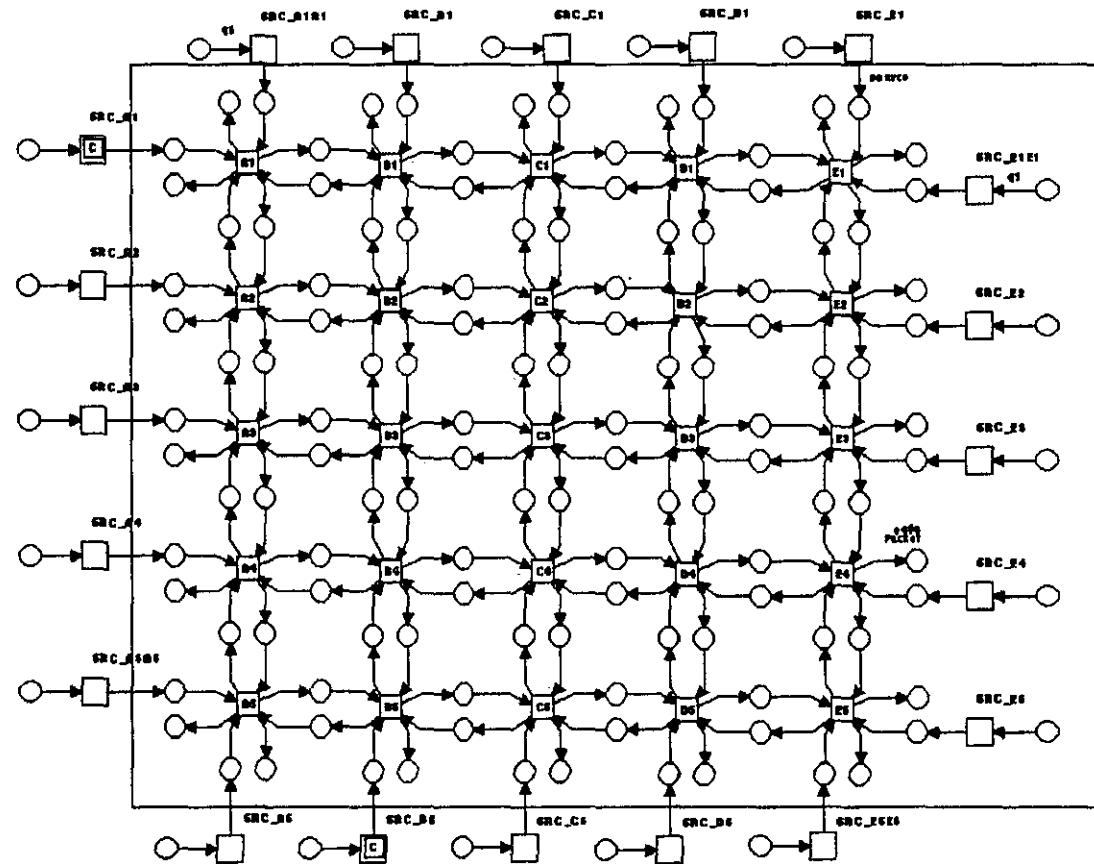
// datalog register locations which house the MEMORY resource values (linked to changes in place: Global_Memory_Store)
globref regloc0 = 100;
globref regloc1 = 100;
globref regloc2 = 100;
globref regloc3 = 100;
globref regloc4 = 100;
globref regloc5 = 100;
globref regloc6 = 100;
globref regloc7 = 100;
globref regloc8 = 100;
globref regloc9 = 100;
globref regloc10 = 100;
```

```
globref regloc11 = 100;  
globref regloc12 = 100;  
globref regloc13 = 100;  
globref regloc14 = 100;  
globref regloc15 = 100;  
globref regloc16 = 100;  
globref regloc17 = 100;  
globref regloc18 = 100;  
globref regloc19 = 100;  
globref regloc20 = 100;  
globref regloc21 = 100;  
globref regloc22 = 100;  
globref regloc23 = 100;  
globref regloc24 = 100;
```

ii.2. Model Hierarchy



ii.3. Top-level Active Network

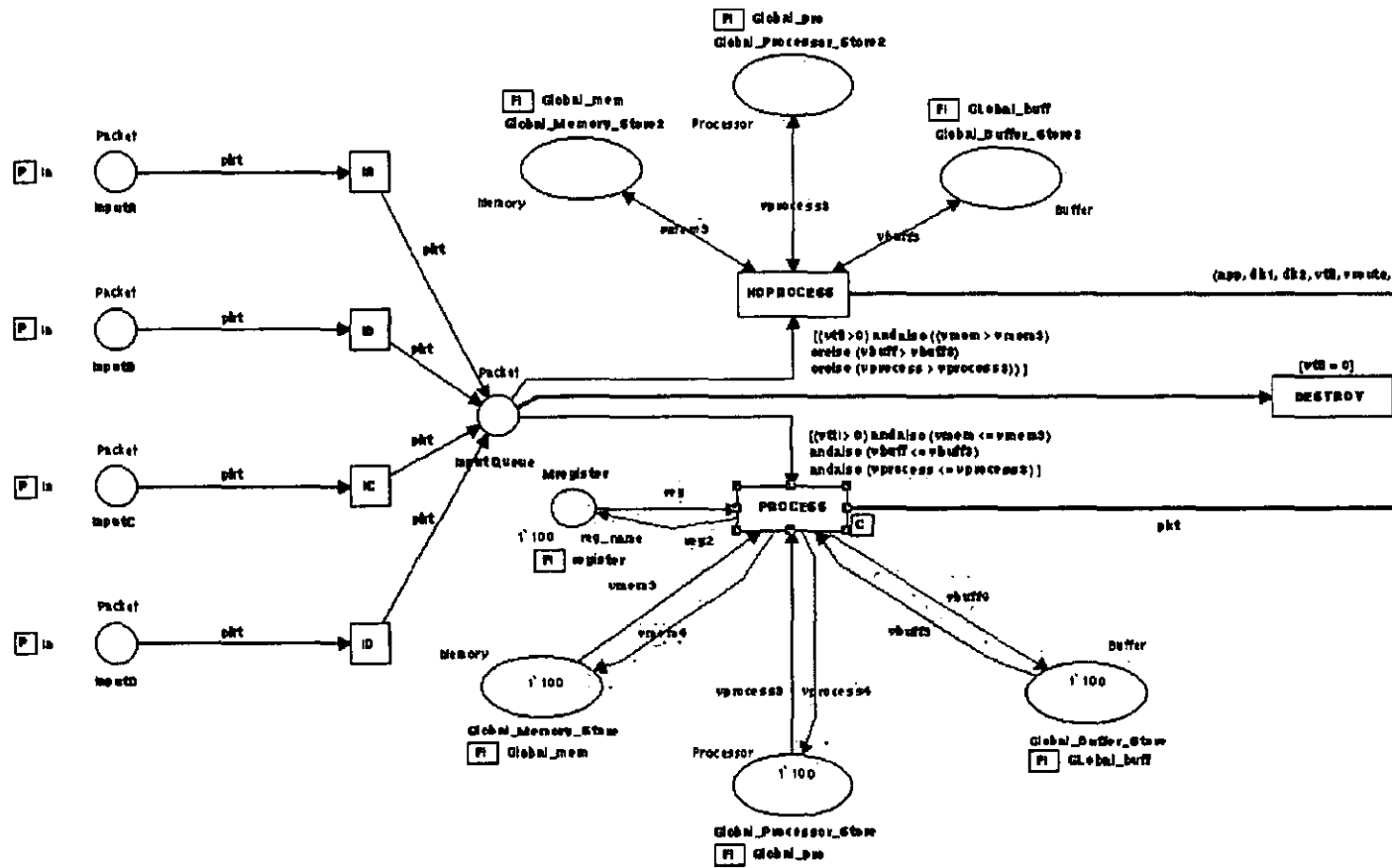


```
// Purpose: code for traffic generation at source SRC_A1
// Location: Petri Net transition: SRC_A1
// Description: the code reads the input source file "src.txt" to extract the correct line as input Active Packet
output source;
action
let
    // function definition to extract the entire input file as a list
    fun fileToList fileName =
    let
        // function definition
        fun f fh = if TextIO.endOfStream fh
        then
            let
                val _ = TextIO.closeIn fh
            in
                nil
            end
        else
            // formation of the list of input lines extracted from input source file
            // extract one line in sequence and append to list of other extracted lines
```

```
        (input_ms'Packet fh) :: (f fh)
in
    // function call to open text file
    f (TextIO.openIn fileName)
end;
in
    // function call
    // specifies correct input source file and line no.
    // the line no. is used to select the correct input line specific to the particular source
    // line no. 0 is the input for SRC_A5
    List.nth((fileToList "/home/elmsd2/design_cpn/activenetwork/src.txt"),0)
end;
```

ii.4. Active Node (section 1)

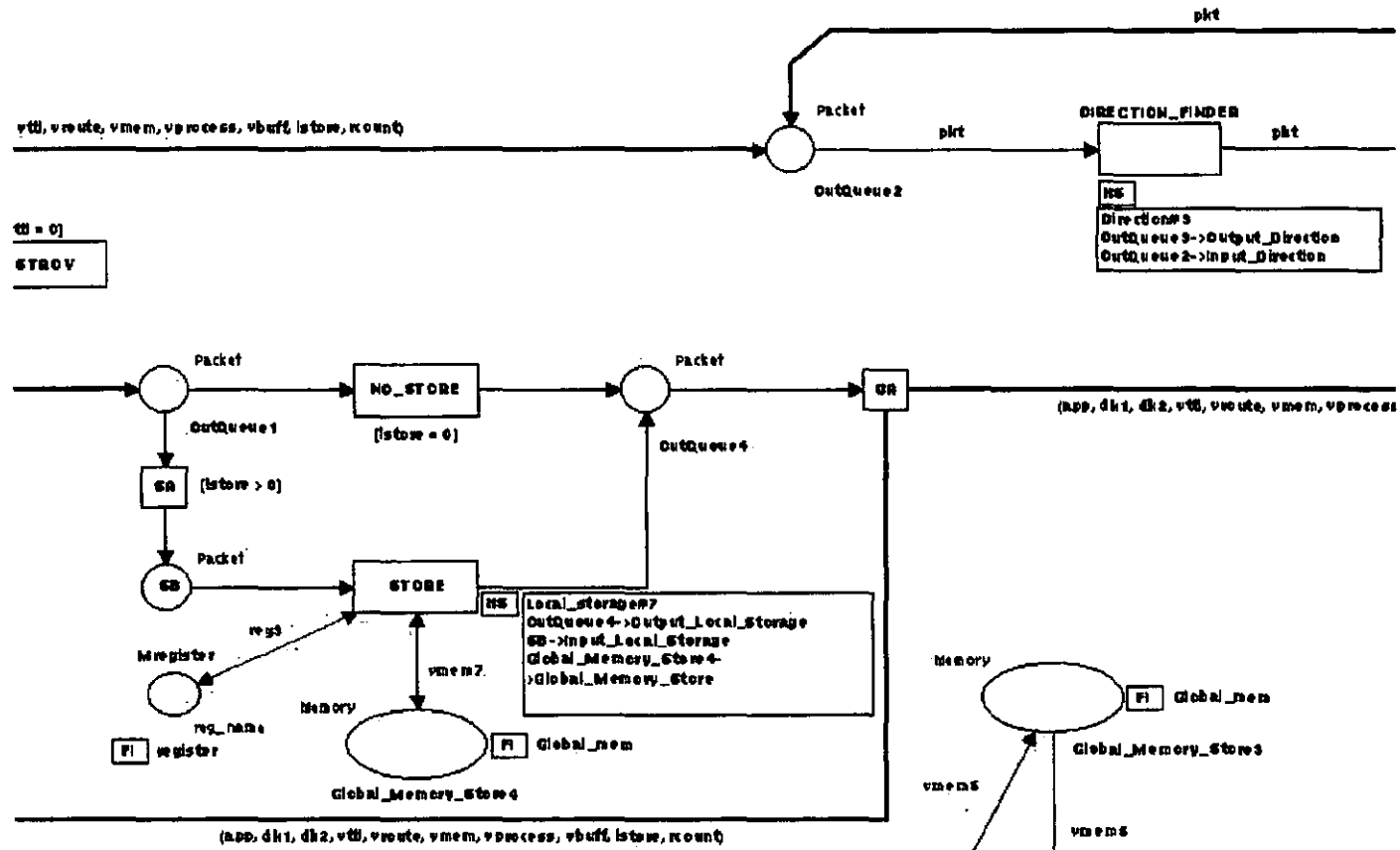
- The first section in the Active Packet process flow.
- Holds the 4 input ports and the decision to process, destroy or forward a particular Active Packet.



```
// Purpose: code to reduce the MEMORY, PROCESSOR and BUFFER resource values once the decision to process an Active
// Packet has been made
// Location: Petri Net transition: PROCESS
// Description: the code uses the resource requirement values held within the Active Packet to reduce the resource values in places: Global_Memory_Store, Global_Processor_Store
and Global_Buffer_Store. It also updates the regloc register location for the specific instance of Active Node
input (app,dk1,dk2,vttl,vroute,vmem,vprocess,vbuff,lstore,rcount,vmem3,vprocess3,vbuff3,reg);
output (pkt,vmem4,vprocess4,vbuff4,reg2);
action
// updates regloc location (linked to changes in place: Global_Memory_Store)
usestring["regloc"^makestring(reg)^":"="^makestring(vmem3-vmem)];
((app,dk1,dk2,(vttl-1),vroute,vmem,vprocess,vbuff,lstore,rcount),(vmem3-vmem),(vprocess3-vprocess),(vbuff3-vbuff),reg);
```

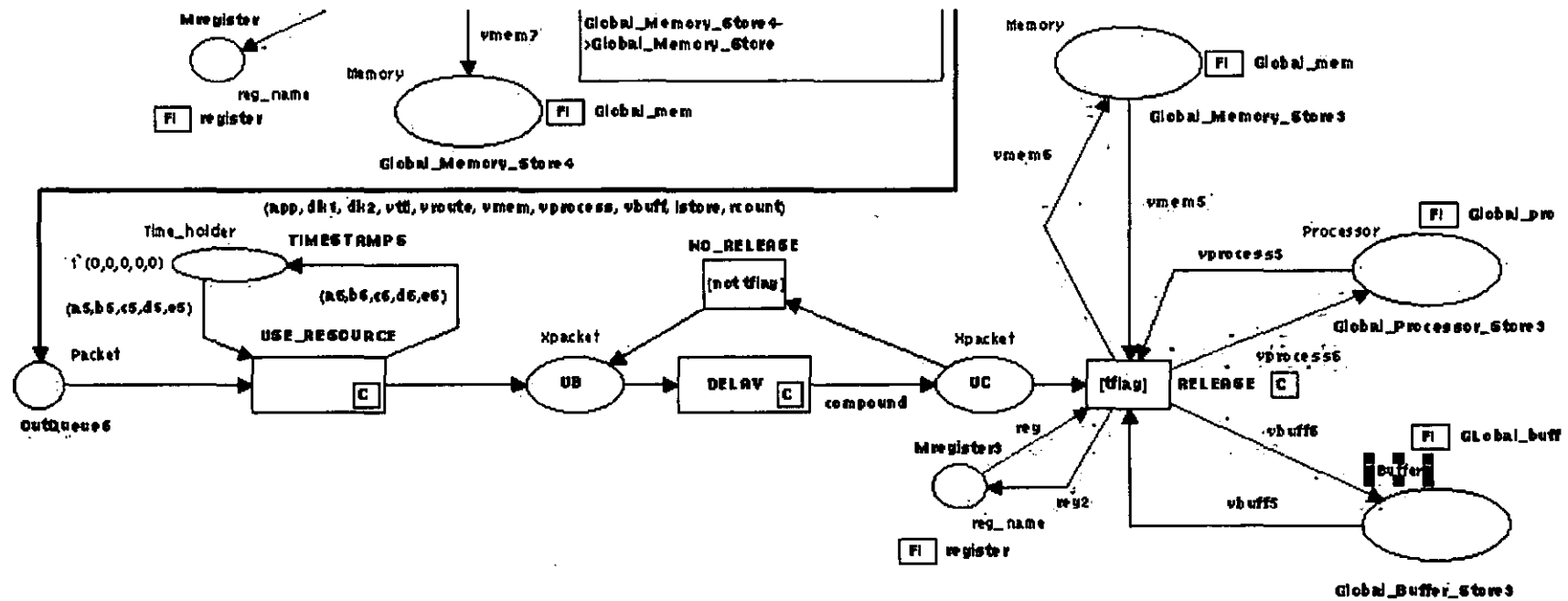
ii.5. Active Node (section 2)

- Positioned right of section 1.
- Holds decision-and-release mechanism for Global State Maintenance (Local Storage component) and the Direction Solver component.



ii.6. Active Node (section 3)

- Positioned below section 2
- Holds the Resource Release Mechanism for all Active Packets that are processed.



// Purpose: code to adjust the resource release time for MERGE Active Packets

// Location: Petri Net transition: USE_RESOURCE

// Description: adjustment of the resource release time for MERGE Active packet is necessary since the initial packets MERGE stream is delayed by 5 time steps. The initial Merge Active Packet follows normal resource release timings. Subsequent merge packets of a stream (that fall within the 5 time step period) will have their resources released immediately since they are merged with the initial packet.

input (app,dk1,dk2,vttl,vroute,vmem,vprocess,vbuff,lstore,rcount,a5,b5,c5,d5,e5);

output (app2,vroute2,vmem2,vprocess2,vbuff2,lstore2,tstamp,tflag,a6,b6,c6,d6,e6);

action

let

// function definition: the same procedure for each MERGE Active packet based on application id: 1 to 5

fun select_merge(APP,VROUTE,VMEM,VPROCESS,VBUFF,LSTORE,A5,B5,C5,D5,E5) =

case APP

// check if within the 5 time step period and not the initial Active Packet of the MERGE stream

of 1 => if ((step() < (A5 + 5)) andalso (0 < A5))

then

// release resource immediately: subsequent packet

(APP,VROUTE,VMEM,VPROCESS,VBUFF,LSTORE,(0-9),false,A5,B5,C5,D5,E5)

else

// release resource normally: initial packet

```

// the MERGE Active Packet is time stamped with the simulation step number
(APP,VROUTE,VMEM,VPROCESS,VBUFF,LSTORE,step(),false,step(),B5,C5,D5,E5)
| 2 => if ((step() < (B5 + 5)) andalso (0 < B5))
  then
    (APP,VROUTE,VMEM,VPROCESS,VBUFF,LSTORE,(0-9),false,A5,B5,C5,D5,E5)
  else
    (APP,VROUTE,VMEM,VPROCESS,VBUFF,LSTORE,step(),false,A5,step(),C5,D5,E5)
| 3 => if ((step() < (C5 + 5)) andalso (0 < C5))
  then
    (APP,VROUTE,VMEM,VPROCESS,VBUFF,LSTORE,(0-9),false,A5,B5,C5,D5,E5)
  else
    (APP,VROUTE,VMEM,VPROCESS,VBUFF,LSTORE,step(),false,A5,B5,step(),D5,E5)
| 4 => if ((step() < (D5 + 5)) andalso (0 < D5))
  then
    (APP,VROUTE,VMEM,VPROCESS,VBUFF,LSTORE,(0-9),false,A5,B5,C5,D5,E5)
  else
    (APP,VROUTE,VMEM,VPROCESS,VBUFF,LSTORE,step(),false,A5,B5,C5,step(),E5)
| 5 => if ((step() < (E5 + 5)) andalso (0 < E5))
  then
    (APP,VROUTE,VMEM,VPROCESS,VBUFF,LSTORE,(0-9),false,A5,B5,C5,D5,E5)

```

```
        else
            (APP,VROUTE,VMEM,VPROCESS,VBUFF,LSTORE,step(),false,A5,B5,C5,D5,step())
            // all other Active Packets are time-stamped with the simulation step number
            // resources released normally
            | _ => (APP,VROUTE,VMEM,VPROCESS,VBUFF,LSTORE,step(),false,A5,B5,C5,D5,E5)
in
    select_merge (app,vroute,vmem,vprocess,vbuff,lstore,a5,b5,c5,d5,e5)
end;
```

```

// Purpose: code to delay the releasing of resources for all Active Packets
// Location: Petri Net transition: DELAY
// Description: the delay is adjusted so that the resource release is timed to coincide with the exit of the Active Packet from the node. Each packet type (based on routing mechanism)
has different resource delay timings based on the time it spends within the Active Node and its sub components.
input (app2,vroute2,vmem2,vprocess2,vbuff2,lstore2,tstamp,tflag);
output (compound);
action
let
    // function definition to specify different delay values for different routing mechanisms
    fun delayvalue(VR) =
        case VR
        of F => 1      // FORWARD Active Packet delay in time steps
         | C => 0      // CONSUME Active Packet delay in time steps
         | M => 9      // MERGE Active Packet delay in time steps (may be adjusted with previous code)
         | R => 2      // REPLICATE Active Packet delay in time steps
in
    // check if release time is reached based on routing mechanism
    if (step () < (tstamp + delayvalue(vroute2)))
        then
            (app2,vroute2,vmem2,vprocess2,vbuff2,lstore2,tstamp,false) // delay not reached, set flag to false, loop back

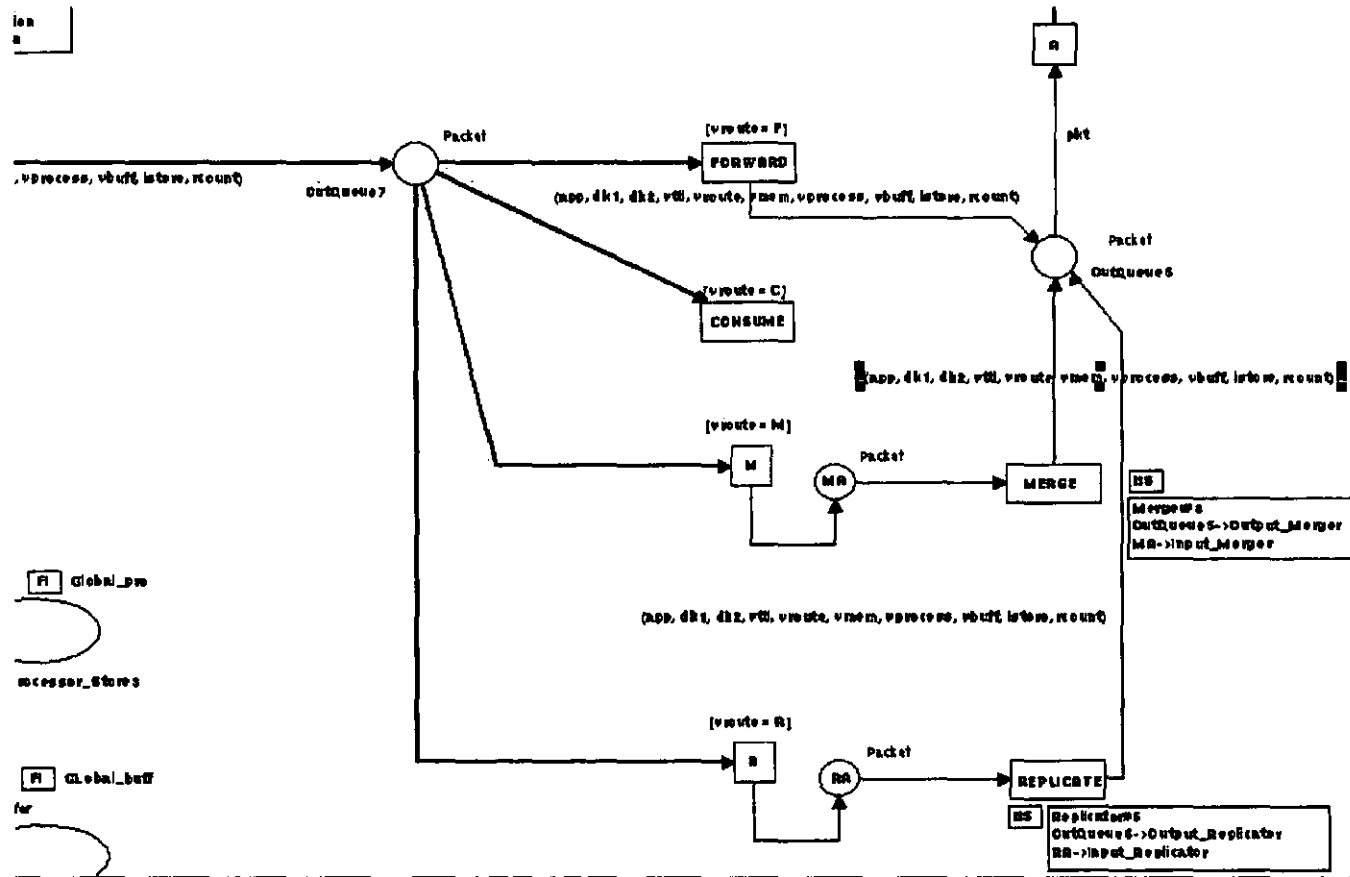
```

```
    else
        (app2,vroute2,vmem2,vprocess2,vbuff2,lstore2,tstamp,true) // delay reached, set flag to true, proceed to next stage
end;
```

```
// Purpose: code to increase the MEMORY, PROCESSOR and BUFFER resource values once the release time value has been reached (checks flag).
// Location: Petri Net transition: RELEASE
// Description: the code uses the resource requirement values held within the Active Packet to increase the resource values in places Global_Memory_Store, Global_Processor_Store
and Global_Buffer_Store. It also updates the regloc register location for the specific instance of Active Node.
input (app2,vroute2,vmem2,vprocess2,vbuff2,lstore2,tstamp,tflag,vmem5,vprocess5,vbuff5,reg);
output (vmem6,vprocess6,vbuff6,reg2);
action
// updates regloc location (linked to changes in place: Global_Memory_Store)
usestring["regloc"^makestring(reg)^":"^makestring(vmem5+(vmem2-lstore2))];
// note: MEMORY resources may not be fully restored due to the Global State Maintenance mechanism (Local Storage Component) reserving resources for additional periods of time
((vmem5+(vmem2-lstore2)),(vprocess5+vprocess2),(vbuff5+vbuff2),reg);
```

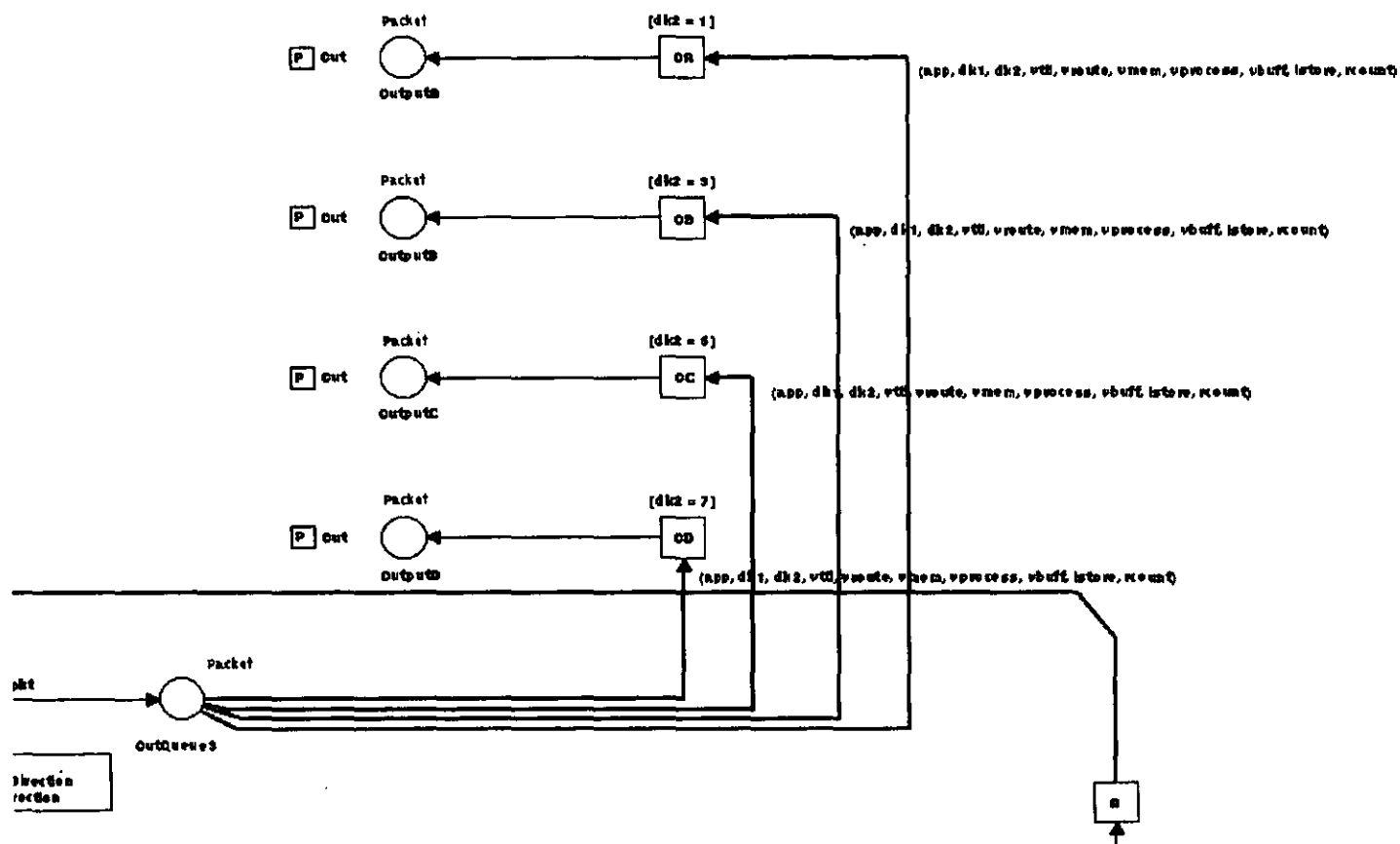
ii.7. Active Node (section 4)

- Positioned right of section 2.
- Differentiates Active Packets based on the 4 routing mechanisms.
- Holds the Merge Packet component and the Replicate Packet component.



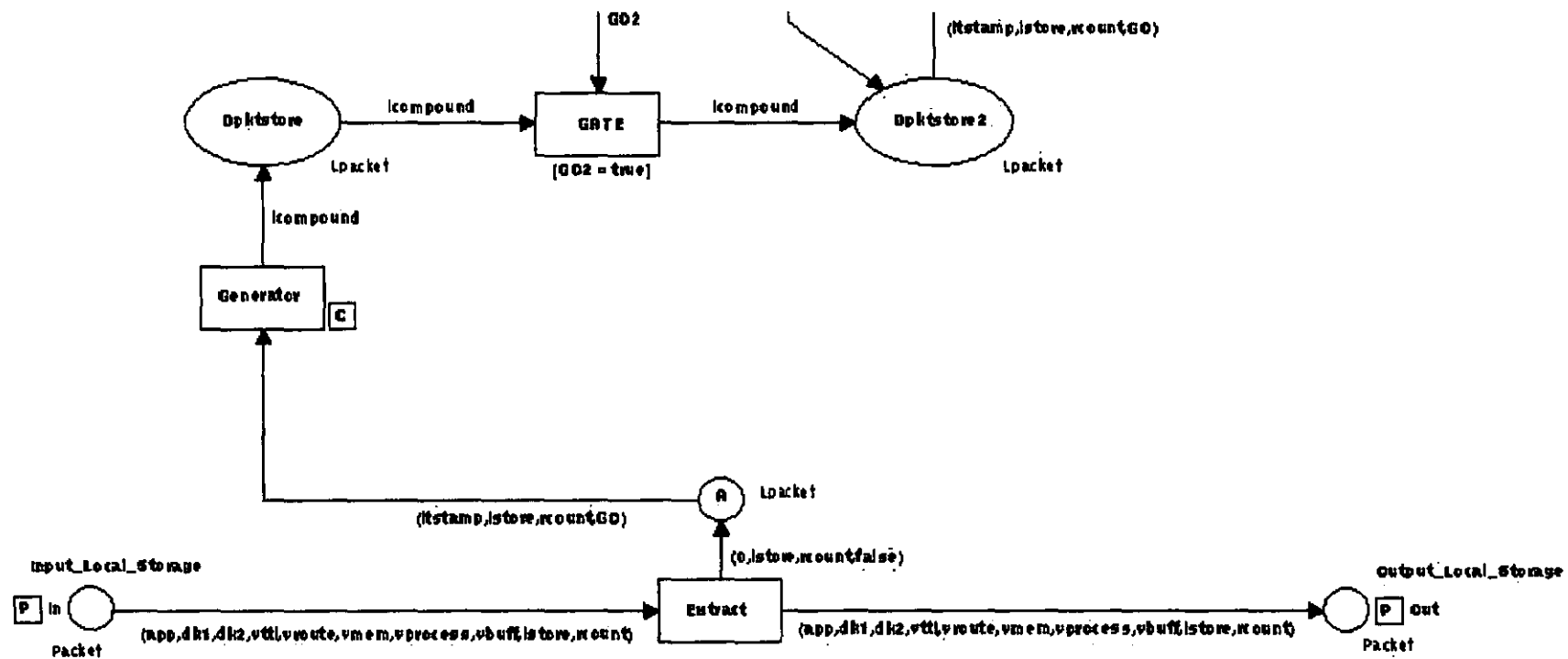
ii.8. Active Node (section 5)

- Positioned above section 4.
- Outputs Active Packets from one of 4 ports.
- The decision to output from a particular port is based on Direction Indicator 2 value, which is held with the Active Packet.
- As a pre-process, the Direction Solver Component modifies the Direction Indicator 2 value based on preset criteria.



ii.9. Local Storage Component in detail (section 1)

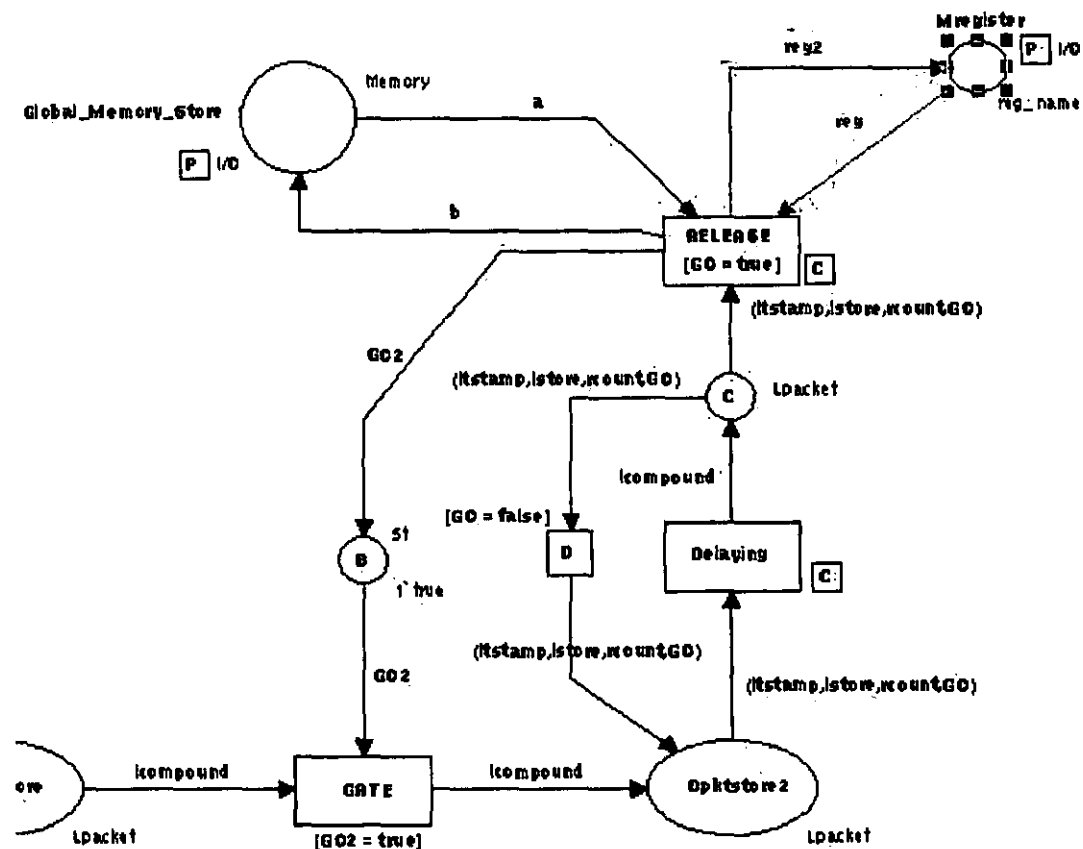
- The first section in the Active Packet process flow.
- The component describes the Global State Maintenance feature of the node (i.e Active Application resource reservation feature).



```
// Purpose: code to add a timestamp to packet in order to calculate the MEMORY resource storage delay
// Location: Petri Net transition: Generator
// Description: the code uses the current simulation step number as a timestamp, which is added to the rcount MEMORY storage time limit in order to calculate the MEMORY release
time in simulation time steps.
input (ltstamp,lstore,rcount,GO);
output (lcompound);
action
((step () + rcount),lstore,rcount,false);
```

ii.10. Local Storage Component in detail (section 2)

- Positioned above section 1.
- The section describes the delay-and-release mechanism for the reserved MEMORY resources.

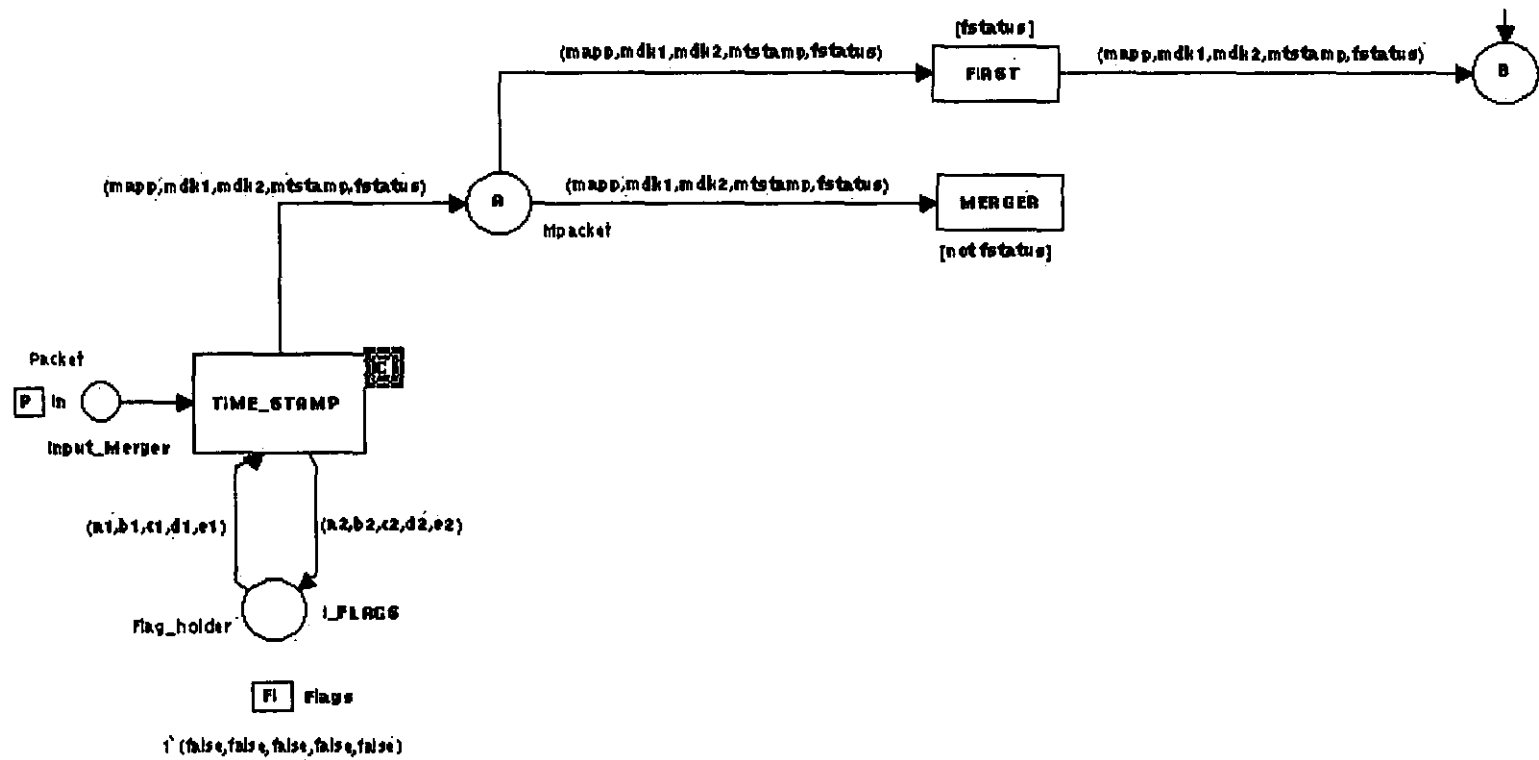


```
// Purpose: code to delay the release of MEMORY resources.
// Location: Petri Net transition: Delaying
// Description: the code uses a check to see if the current simulation time step is less than the timestamp and rcount combination.
input (ltstamp,lstore,rcount,GO);
output (lcompound);
action
let
    // function definition
    fun f (lts,ls,rc) = if (step () < lts)
        then
            (lts,ls,rc,false)          // release time limit not reached. Set check flag to false (to loop back)
        else
            (lts,ls,rc,true)           // release time limit reached. Set check flag to true (to proceed to next stage)
    in
        // function call
        f(ltstamp,lstore,rcount)
    end;
```

```
// Purpose: code to increase the MEMORY resource values once the release time value has been reached (checks flag).
// Location: Petri Net transition: RELEASE
// Description: the code uses the resource reserved value lstore, held within the Active Packet, to increase the resource values in place Global_Memory_Store (same as
Global_Memory_Store in Active Node). It also updates the regloc register location for the specific instance of Active Node.
input (a,lstamp,lstore,rcount,GO,reg);
output (b,GO2,reg2);
action
// updates regloc location (linked to changes in place: Global_Memory_Store in Active Node)
usestring["regloc"^makestring(reg)^":="^makestring(a+lstore)];
((a+lstore),true,reg);
```

ii.11. Merge Packet Component in detail (section 1)

- The first section in the Active Packet process flow.
- The component describes the Active Packet Merge feature.
- This section extracts the initial Merge Active Packet from subsequent packets.



// Purpose: check to see if the MERGE Active Packet is the 1st of the stream. It also adds a timestamp to each Merge Active Packet

// Location: Petri Net transition: TIME_STAMP

// Description: A check flag is set when the process identifies the initial Merge Active Packet. This flag is used to identify subsequent Merge Active Packets, of a particular application number, as being suitable for merging with initial Active Packet.

input (app,dk1,dk2,vttl,vroute,vmem,vprocess,vbuff,lstore,rcount,a1,b1,c1,d1,e1);

output (mapp,mdk1,mdk2,mtstamp,fstatus,a2,b2,c2,d2,e2);

action

let

// function definition: the same procedure for each MERGE Active packet based on application id: 1 to 5

fun checkflags (APP,DK1,DK2,A1,B1,C1,D1,E1) =

case APP

of 1 => if (A1 = false) // for application id: 1

then

(APP,DK1,DK2,step(),true,true,B1,C1,D1,E1) // initial packet detected. Check flag set to true

else

(APP,DK1,DK2,step(),false,A1,B1,C1,D1,E1) // initial packet not detected. Check flag remains false

| 2 => if (B1 = false) // for application id: 2

then

(APP,DK1,DK2,step(),true,A1,true,C1,D1,E1)

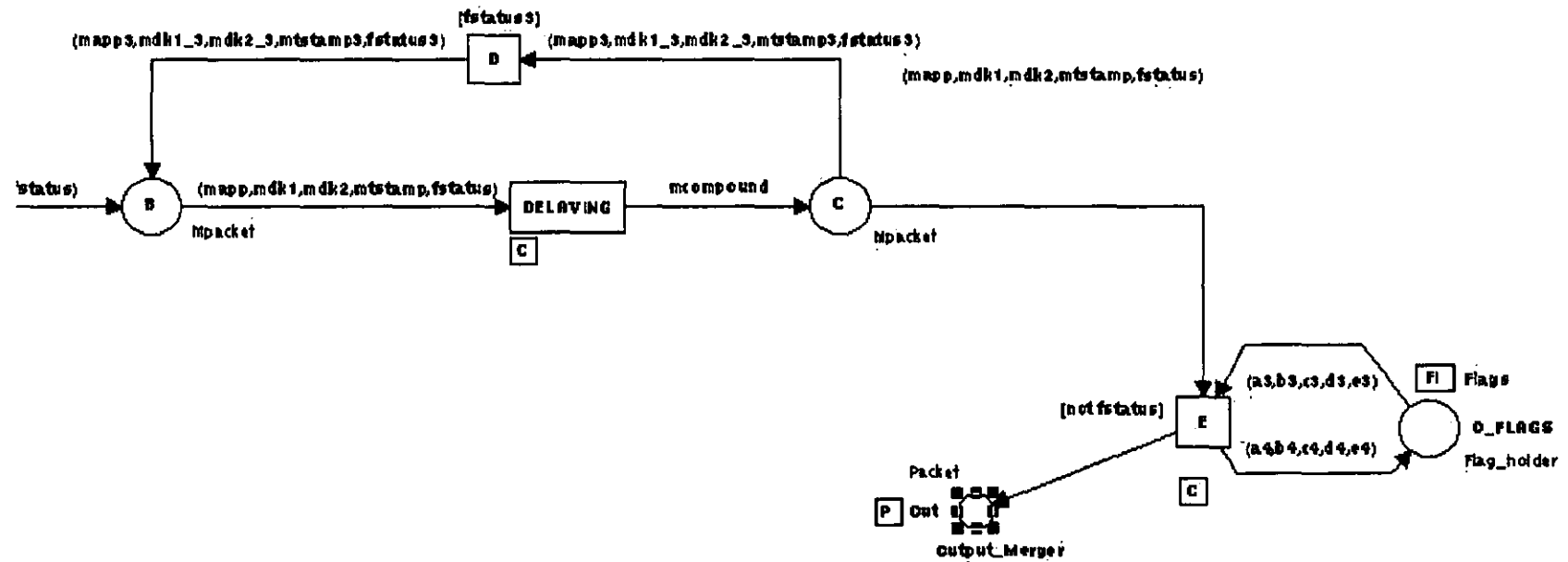
else

```

                                (APP,DK1,DK2,step(),false,A1,B1,C1,D1,E1)
| 3 => if (C1 = false)          // for application id: 3
    then
                                (APP,DK1,DK2,step(),true,A1,B1,true,D1,E1)
    else
                                (APP,DK1,DK2,step(),false,A1,B1,C1,D1,E1)
| 4 => if (D1 = false)          // for application id: 4
    then
                                (APP,DK1,DK2,step(),true,A1,B1,C1,true,E1)
    else
                                (APP,DK1,DK2,step(),false,A1,B1,C1,D1,E1)
| 5 => if (E1 = false)          // for application id: 5
    then
                                (APP,DK1,DK2,step(),true,A1,B1,C1,D1,true)
    else
                                (APP,DK1,DK2,step(),false,A1,B1,C1,D1,E1)
in
    // function call
    checkflags(app,dk1,dk2,a1,b1,c1,d1,e1)
end;
```

ii.12. Merge Packet Component in detail (section 2)

- Positioned right of section 1.
- The section describes the delay-and-release mechanism for the Merge Active Packet and the reset of the check flag.



```
// Purpose: code to delay the release of the initial Merge Active Packet.
// Location: Petri Net transition: DELAYING
// Description: the code uses a check to see if the current simulation time step is less than the timestamp + 5 simulation time steps.
input (mapp,mdk1,mdk2,mtstamp,fstatus);
output (mcompound);
action
let
    // function definition
    fun f (APP,DK1,DK2,TST,FST) =
    if (step () < (TST + 5))
        then
            (APP,DK1,DK2,TST,true)           // release time limit not reached. Set check flag to true (to loop back)
        else
            (APP,DK1,DK2,TST,false)          // release time limit reached. Set check flag to false (to proceed to next stage)
    in
        // function call
        f(mapp,mdk1,mdk2,mtstamp,fstatus)
    end;
```

```

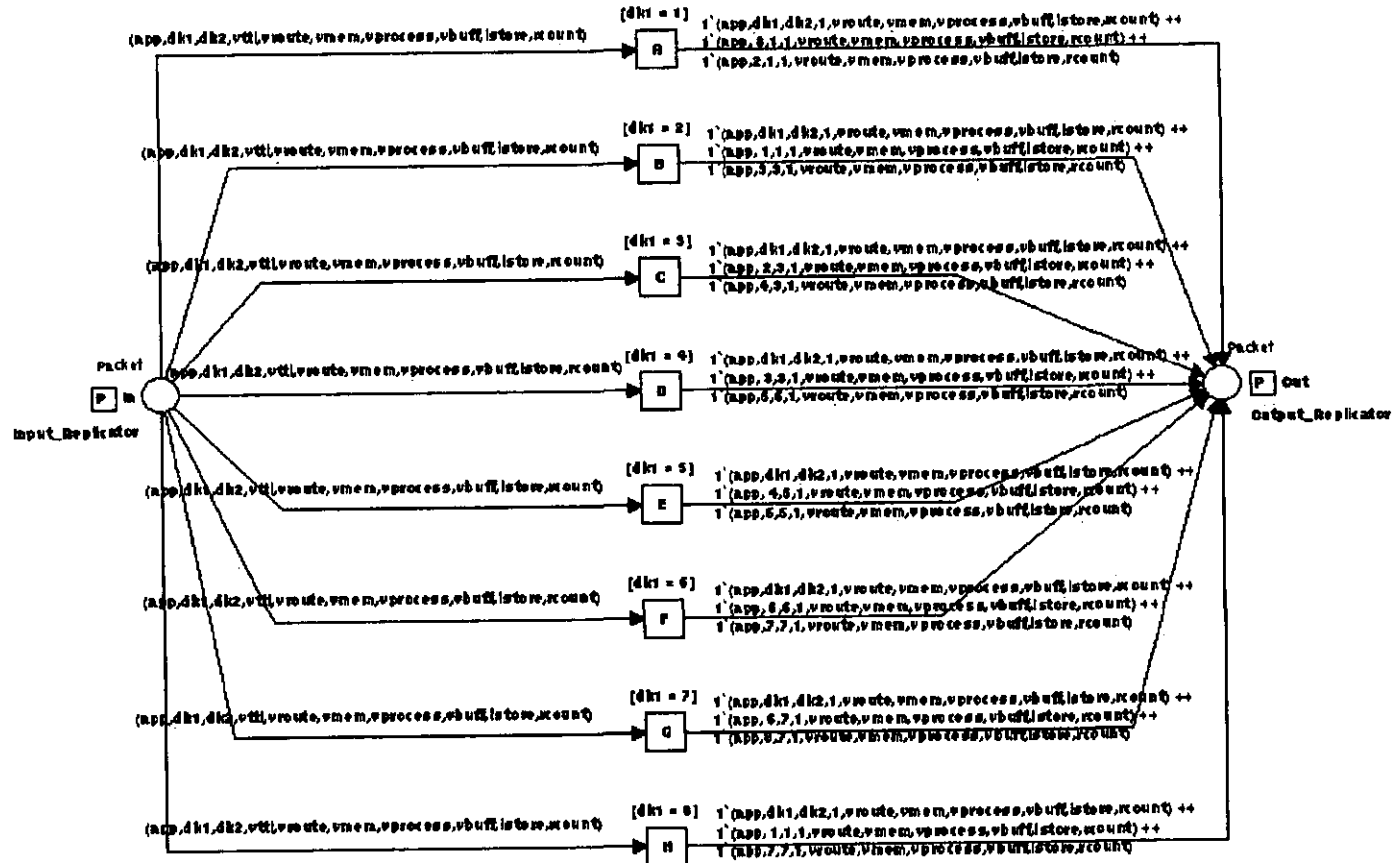
// Purpose: code to reset the specific check flag used in the selection of the initial Merge Active Packet, which has at this stage been released
// Location: Petri Net transition: E
// Description: a function is used to differentiate and reset the check flags of initial Merge Active Packets based on application id.
input (mapp,mdk1,mdk2,mtstamp,fstatus,a3,b3,c3,d3,e3);
output (app,dk1,dk2,vttl,vroute,vmem,vprocess,vbuff,lstore,rcount,a4,b4,c4,d4,e4);
action
let
    // function definition: the same procedure for each MERGE Active packet based on application id: 1 to 5
    fun checkflags2 (APP2,DK1_2,DK2_2,A3,B3,C3,D3,E3) =
        case APP2
        of 1 => (1,DK1_2,DK2_2,1,M,10,10,10,0,0,false,B3,C3,D3,E3)
         | 2 => (2,DK1_2,DK2_2,1,M,20,10,10,10,15,A3,false,C3,D3,E3)
         | 3 => (3,DK1_2,DK2_2,1,M,20,20,20,0,0,A3,B3,false,D3,E3)
         | 4 => (4,DK1_2,DK2_2,1,M,30,10,10,20,10,A3,B3,C3,false,E3)
         | 5 => (5,DK1_2,DK2_2,1,M,40,40,40,25,10,A3,B3,C3,D3,false)

in
    // function call
    checkflags2(mapp,mdk1,mdk2,a3,b3,c3,d3,e3)

end;
```

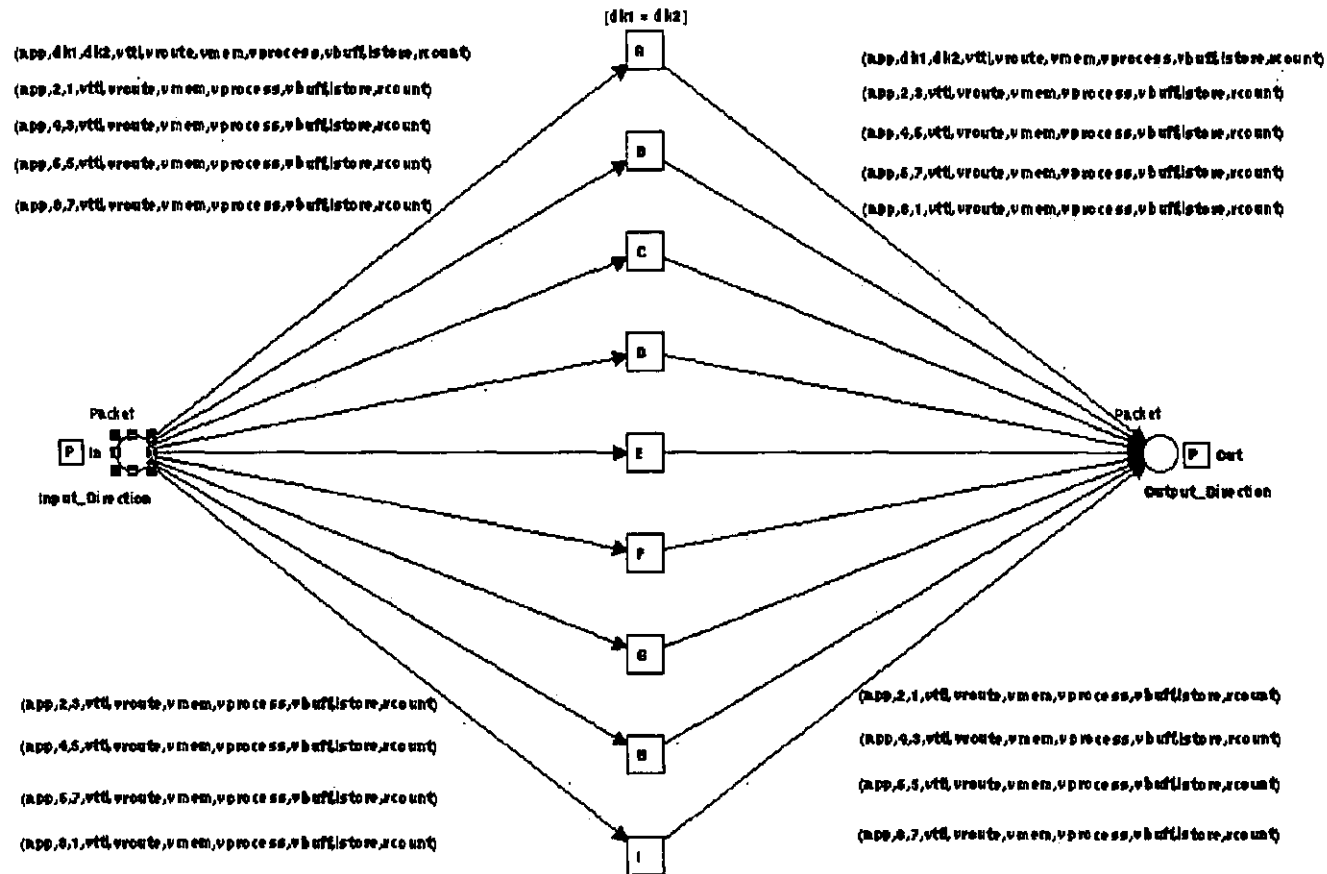
ii.13. Replicate Packet Component in detail

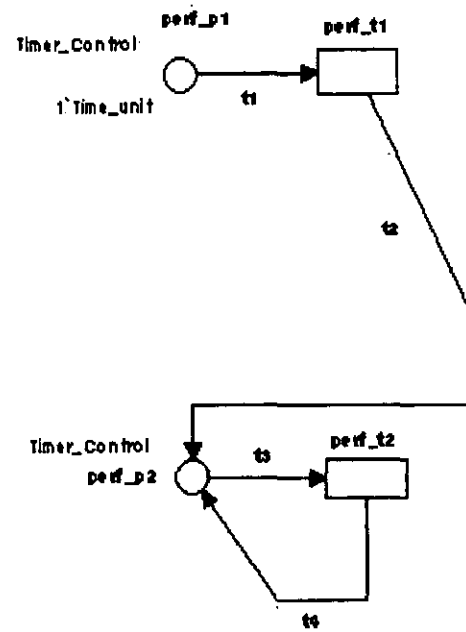
- The component describes the Replicate Packet feature, which replicates Active packets according to the prescribed scheme.
- Modifies the replicated Active Packets' Direction Indication 1 and Direction Indicator 2 values.



ii.14. Direction Solver Component in detail

- The component describes the Direction Solver feature used to modify the Direction Indicator 2 value so that an Active Packet exits the Active Node from the correct output port.



ii.15. Data logging component

```
// Purpose: code for the setup of the data logging process
// Location: Petri Net transition: perf_t1
// Description: the code requests inputs from the user to select which Petri Net places to track and sets up initial place values
input (t1);
output (t2);
action
// asks user to select the page (i.e. the Active Node component) which returns a page handle
tpage_id:= DSUI_AskUserToSelectPage();
DSStr_SetCurPage(!tpage_id);
// asks user to select page-instance-holder place which returns a place handle (user selects: Mregister)
tplace1_id:= DSUI_SelectObject{objtype=NODE_TYPE,override=false};
// asks user to select MEMORY place which returns a place handle (user selects: Global_Memory_Store)
tplace2_id:= DSUI_SelectObject{objtype=NODE_TYPE,override=false};
// asks user to select PROCESSOR place which returns a place handle (user selects: Global_Processor_Store)
tplace3_id:= DSUI_SelectObject{objtype=NODE_TYPE,override=false};
// asks user to select BUFFER place which returns a place handle (user selects: Global_Buffer_Store)
tplace4_id:= DSUI_SelectObject{objtype=NODE_TYPE,override=false};

// marks each page-instance-holder with the correct page instance (ACTIVE NODE IDENTIFIER)
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),0)),placeid=(!tplace1_id),mark="1`0"}];
```

```
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),1)),placeid=(!tplace1_id),mark="1`1"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),2)),placeid=(!tplace1_id),mark="1`2"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),3)),placeid=(!tplace1_id),mark="1`3"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),4)),placeid=(!tplace1_id),mark="1`4"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),5)),placeid=(!tplace1_id),mark="1`5"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),6)),placeid=(!tplace1_id),mark="1`6"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),7)),placeid=(!tplace1_id),mark="1`7"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),8)),placeid=(!tplace1_id),mark="1`8"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),9)),placeid=(!tplace1_id),mark="1`9"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),10)),placeid=(!tplace1_id),mark="1`10"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),11)),placeid=(!tplace1_id),mark="1`11"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),12)),placeid=(!tplace1_id),mark="1`12"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),13)),placeid=(!tplace1_id),mark="1`13"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),14)),placeid=(!tplace1_id),mark="1`14"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),15)),placeid=(!tplace1_id),mark="1`15"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),16)),placeid=(!tplace1_id),mark="1`16"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),17)),placeid=(!tplace1_id),mark="1`17"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),18)),placeid=(!tplace1_id),mark="1`18"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),19)),placeid=(!tplace1_id),mark="1`19"}];
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),20)),placeid=(!tplace1_id),mark="1`20"}];
```

```
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),21)),placeid=(!tplace1_id),mark="1'21"}];  
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),22)),placeid=(!tplace1_id),mark="1'22"}];  
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),23)),placeid=(!tplace1_id),mark="1'23"}];  
usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),24)),placeid=(!tplace1_id),mark="1'24"}];
```

```
// initialises datalog register locations which house the MEMORY resource values (linked to changes in place: Global_Memory_Store)
```

```
regloc0:= 100;  
regloc1:= 100;  
regloc2:= 100;  
regloc3:= 100;  
regloc4:= 100;  
regloc5:= 100;  
regloc6:= 100;  
regloc7:= 100;  
regloc8:= 100;  
regloc9:= 100;  
regloc10:= 100;  
regloc11:= 100;  
regloc12:= 100;  
regloc13:= 100;
```

```
regloc14:= 100;  
regloc15:= 100;  
regloc16:= 100;  
regloc17:= 100;  
regloc18:= 100;  
regloc19:= 100;  
regloc20:= 100;  
regloc21:= 100;  
regloc22:= 100;  
regloc23:= 100;  
regloc24:= 100;
```

```
// initialises text file handles for the reading in of custom initial MEMORY, PROCESSOR and BUFFER values.
```

```
fh1:= TextIO.openIn "/home/elmsd2/design_cpn/activenetwork/initmemval.txt";  
fh2:= TextIO.openIn "/home/elmsd2/design_cpn/activenetwork/initproval.txt";  
fh3:= TextIO.openIn "/home/elmsd2/design_cpn/activenetwork/initbufval.txt";
```

```
N:=0;  
mem_val:="null";  
pro_val:="null";
```

```

buf_val:="null";

// marks MEMORY, PROCESSOR and BUFFER values for each instance of Active Node with custom initial values taken from text files.
// initialises places: Global_Memory_Store, Global_Processor_Store, Global_Buffer_Store
// this is an optional process not used in simulations. All MEMORY, PROCESSOR and BUFFER initial values are set to 100
while (!N < 25) do (
    mem_val:= TextIO.inputLine (!fh1);
    pro_val:= TextIO.inputLine (!fh2);
    buf_val:= TextIO.inputLine (!fh3);

    usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),!N)),placeid=(!tplace2_id),mark="1"^(!mem_val)}}];
    usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),!N)),placeid=(!tplace3_id),mark="1"^(!pro_val)}}];
    usestring[GetChangeMarkingCode{instid=(List.nth(GetPageInsts(!tpage_id),!N)),placeid=(!tplace4_id),mark="1"^(!buf_val)}}];

    N:=(!N+1)
);

Time_unit;

```

```

// Purpose: continuous loop for the logging of MEMORY resource values of all Active Nodes.
// Location: Petri Net transition: perf_t2
// Description: at each pass of the loop the code writes (as one string) the regloc values for all instances of Active Node.
input (t3);
output(t4);
action
// make the string with all the regloc values
mk_1:= makestring(step())^"...^makestring(!regloc0)^", "^makestring(!regloc1)^", "^makestring(!regloc2)^", "^makestring(!regloc3)^", "^makestring(!regloc4)^", "^makestring
(!regloc5)^", "^makestring(!regloc6)^", "^makestring(!regloc7)^", "^makestring(!regloc8)^", "^makestring(!regloc9)^", "^makestring(!regloc10)^", "^makestring(!regloc11)
^", "^makestring(!regloc12)^", "^makestring(!regloc13)^", "^makestring(!regloc14)^", "^makestring(!regloc15)^", "^makestring(!regloc16)^", "^makestring(!regloc17)^", "^makestring
(!regloc18)^", "^makestring(!regloc19)^", "^makestring(!regloc20)^", "^makestring(!regloc21)^", "^makestring(!regloc22)^", "^makestring(!regloc23)^", "^makestring(!regloc24);

let
    // open file and write one line
    val outstr = TextIO.openAppend ("/home/elmsd2/design_cpn/activenetwork/log/out.txt")
    val a = TextIO.output (outstr,!mk_1)^"\n"
    val _ = TextIO.closeOut outstr
in
    Time_unit
end;
```

A P P E N D I X I I I

iii. Petri-Net Simulators Reviewed

Name	Features		Environment
	Petri-Net types supported	Components	
ALPHA/Sim <i>Commercial</i> <i>(academic discount)</i>	High-level Petri-Nets Petri-Nets with Time	Graphical Editor Token Game Animation Fast Simulation Simple Performance Analysis	SunOS Solaris MS Windows NT
Artifex <i>Commercial</i> <i>(academic discount)</i>	High-level Petri-Nets Petri-Nets with Time	Graphical Editor Token Game Animation Fast Simulation Simple Performance Analysis Report generator C code generator	Sun HP Silicon Graphics Linux MS Windows
CPN-AMI <i>Free of charge</i>	High-level Petri-Nets Place/Transition Nets	Graphical Editor Fast Simulation State Spaces Place Invariants Transition Invariants Structural Analysis Services for modular modelling	Sun Linux Macintosh

DaNAMICS <i>Commercial</i>	High-level Petri-Nets Stochastic Petri-Nets	Graphical Editor Token Game Animation Fast Simulation State Spaces Place Invariants Transition Invariants Structural Analysis Simple Performance Analysis Advanced Performance Analysis	Java
Design/CPN <i>Free of charge</i>	High-level Petri-Nets Petri-Nets with Time	Graphical Editor Token Game Animation Fast Simulation State Spaces Simple Performance Analysis Interchange File Format	Sun HP Silicon Graphics Linux
GreatSPN <i>Commercial (free for academic purposes)</i>	High-level Petri-Nets Stochastic Petri-Nets Petri-Nets with Time	Graphical Editor Token Game Animation Fast Simulation State Spaces Condensed State Spaces Place Invariants Transition Invariants Structural Analysis Advanced Performance Analysis	Sun Linux

INCOME Process Designer <i>Commercial</i> <i>(free for</i> <i>academic</i> <i>purposes)</i>	High-level Petri-Nets Stochastic Petri-Nets Petri-Nets with Time	Graphical Editor Token Game Animation Fast Simulation Transition Invariants Net Reductions Structural Analysis Simple Performance Analysis Advanced Performance Analysis Interchange File Format Interfaces to workflow engines, CASE tools, integrated document management, process monitoring	Sun HP Silicon Graphics Linux MS Windows Java
Moses Tool Suit <i>Free of</i> <i>charge</i>	High-level Petri-Nets Stochastic Petri-Nets Petri-Nets with Time	Graphical Editor Token Game Animation Fast Simulation User-extendable	Sun Linux MS Windows Java

PACE <i>Commercial</i> <i>(academic discount)</i>	Object-oriented PNs High-level Petri-Nets Place/Transition Nets Stochastic Petri-Nets Petri-Nets with Time	Graphical Editor Token Game Animation Fast Simulation Net Reductions Fuzzy Modelling	Sun MS Windows
PetriSim <i>Free of charge</i>	High-level Petri-Nets Place/Transition Nets Petri-Nets with Time	Graphical Editor Fast Simulation	MS DOS
RENEW <i>Free of charge</i>	Object-oriented PNs High-level Petri-Nets Place/Transition Nets Petri-Nets with Time	Graphical Editor Token Game Animation Fast Simulation Interchange File Format	Java

TimeNET <i>Commercial</i> <i>(free for academic purposes)</i>	High-level Petri-Nets Place/Transition Nets Stochastic Petri-Nets Petri-Nets with Time	Graphical Editor Token Game Animation Fast Simulation State Spaces Place Invariants Structural Analysis Simple Performance Analysis Advanced Performance Analysis Interchange File Format	Sun Linux
Visual Object Net ++ <i>Free of charge</i>	Place/Transition Nets Petri-Nets with Time	Graphical Editor Token Game Animation Fast Simulation Structural Analysis Simple Performance Analysis Supports object hierarchies	MS Windows

A P P E N D I X I V

iv. Case Study Tabulated Results

iv.1. Case Study 1

Trace no	Active Node	Hurst value	r^2 value
1	e5	0.8189	0.9701
2	e4	0.7984	0.9520
3	e3	0.8648	0.9628
4	e2	0.8936	0.9657
5	e1	0.9146	0.9818
6	d5	0.9006	0.9938
7	d4	0.8822	0.9760
8	d3	0.7201	0.9261
9	d2	0.7248	0.9211
10	d1	0.9062	0.9723
11	c5	1.0451	0.9489
12	c4	0.9780	0.9663
13	c3	0.6299	0.8575
14	c2	0.6509	0.8998
15	c1	0.8624	0.9627
16	b5	0.9680	0.9899
17	b4	1.0112	0.9863
18	b3	0.9633	0.9615
19	b2	0.7425	0.9846
20	b1	0.8473	0.9669
21	a5	0.9794	0.9828
22	a4	1.0288	0.9801
23	a3	0.9962	0.9743
24	a2	0.8165	0.9732
25	a1	0.8398	0.9687

iv.2. Case Study 2

Trace no	Active Node	Hurst value	r ² value
1	e5	0.7358	0.9261
2	e4	0.6898	0.9619
3	e3	0.6845	0.9532
4	e2	0.7365	0.9800
5	e1	0.8036	0.9813
6	d5	0.6742	0.9700
7	d4	0.6913	0.9146
8	d3	0.4947	0.8364
9	d2	0.5873	0.9284
10	d1	0.7939	0.9877
11	c5	0.6876	0.9738
12	c4	0.5522	0.8096
13	c3	0.7783	0.9012
14	c2	0.5768	0.8273
15	c1	0.7434	0.9626
16	b5	0.6799	0.9449
17	b4	0.5126	0.9045
18	b3	0.5381	0.8842
19	b2	0.5163	0.8537
20	b1	0.5482	0.8183
21	a5	0.6942	0.9479
22	a4	0.6596	0.9411
23	a3	0.6765	0.9468
24	a2	0.6511	0.9033
25	a1	0.6337	0.9128

iv.3. Case Study 3

Trace no	Active Node	Hurst value	r ² value
1	e5	0.7700	0.9736
2	e4	0.8383	0.9580
3	e3	0.8926	0.9789
4	e2	0.8708	0.9670
5	e1	0.8834	0.9783
6	d5	0.9176	0.9914
7	d4	0.9126	0.9796
8	d3	0.8025	0.9841
9	d2	0.7125	0.9264
10	d1	0.8656	0.9693
11	c5	1.0063	0.9619
12	c4	1.0057	0.9643
13	c3	0.6187	0.9406
14	c2	0.8176	0.9720
15	c1	0.8779	0.9787
16	b5	0.9878	0.9751
17	b4	1.0160	0.9701
18	b3	1.0561	0.9508
19	b2	0.7821	0.9568
20	b1	0.8756	0.9779
21	a5	0.9735	0.9893
22	a4	0.9615	0.9835
23	a3	0.9482	0.9898
24	a2	0.8948	0.9692
25	a1	0.8629	0.9721

iv.4. Case Study 4

Trace no	Active Node	Hurst value	r ² value
1	e5	0.7030	0.9888
2	e4	0.7672	0.9889
3	e3	0.7740	0.9894
4	e2	0.8969	0.9673
5	e1	0.7898	0.9920
6	d5	0.7358	0.9811
7	d4	0.7188	0.9786
8	d3	0.7908	0.9921
9	d2	0.8372	0.9838
10	d1	0.6771	0.9908
11	c5	0.8021	0.9771
12	c4	0.7969	0.9790
13	c3	0.9329	0.9760
14	c2	0.8605	0.9809
15	c1	0.7009	0.9824
16	b5	0.7526	0.9820
17	b4	0.8570	0.9786
18	b3	0.9356	0.9783
19	b2	0.9381	0.9780
20	b1	0.8579	0.9799
21	a5	0.5945	0.9795
22	a4	0.7822	0.9901
23	a3	0.7060	0.9860
24	a2	0.7748	0.9921
25	a1	0.8389	0.9797

iv.5. Case Study 5

Trace no	Active Node	Hurst value	r ² value
1	e5	0.7038	0.9353
2	e4	0.7193	0.9473
3	e3	0.8329	0.9760
4	e2	0.8209	0.9698
5	e1	0.8711	0.9739
6	d5	0.7020	0.9326
7	d4	0.7323	0.9704
8	d3	0.7966	0.8765
9	d2	0.7976	0.8676
10	d1	0.7286	0.9387
11	c5	0.8281	0.9863
12	c4	0.7044	0.9631
13	c3	0.7205	0.9299
14	c2	0.8717	0.9255
15	c1	0.7494	0.9601
16	b5	0.7747	0.9806
17	b4	0.6585	0.9304
18	b3	0.4985	0.8770
19	b2	0.8684	0.9719
20	b1	0.8611	0.9758
21	a5	0.8517	0.9712
22	a4	0.7092	0.9634
23	a3	0.7766	0.9634
24	a2	0.8216	0.9724
25	a1	0.8927	0.9733

iv.6. Case Study 6

Trace no	Active Node	Hurst value	r^2 value
1	e5	0.9316	0.9828
2	e4	0.9349	0.9824
3	e3	0.8694	0.9802
4	e2	0.7924	0.9840
5	e1	0.7884	0.9576
6	d5	0.8214	0.9671
7	d4	0.7627	0.9550
8	d3	0.8567	0.8652
9	d2	0.7512	0.9048
10	d1	0.7855	0.9615
11	c5	0.8389	0.9654
12	c4	0.7789	0.8956
13	c3	0.6083	0.8504
14	c2	0.6131	0.7901
15	c1	0.7967	0.9600
16	b5	0.9171	0.9727
17	b4	0.8758	0.9865
18	b3	0.6292	0.8659
19	b2	0.5154	0.7209
20	b1	0.7033	0.9259
21	a5	0.7858	0.9450
22	a4	0.9213	0.9802
23	a3	0.8162	0.9534
24	a2	0.7060	0.8714
25	a1	0.7269	0.9105

