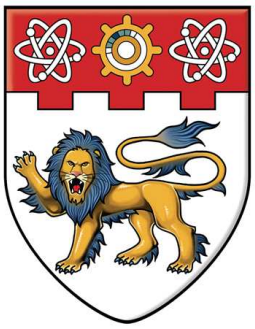

NANYANG TECHNOLOGICAL UNIVERSITY

Anti-fragile Internet with Autonomous Swarm Networks

Lua Rui Ping

School of Computer Engineering

2014

Anti-fragile Internet with Autonomous Swarm Networks

Lua Rui Ping

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2014

Abstract

Internet services and traffic is growing at an exponential rate. They are however vulnerable to flash crowds and Distributed Denial of Service attacks. Existing techniques are difficult to scale and have limited effectiveness. Most of them addresses specific attacks and do not provide wider coverage. We explore concepts of anti-fragility, autonomic and swarm computing to address these problems. We propose a decentralized and iterative overlay structure to reduce disruptions and risks in large networks. An autonomic system features (1) Self-configuration, (2) Self-healing, (3) Self-optimization and (4) Self-protection. For each of the features, we propose appropriate mechanisms. We show how each component is integrated to addresses simple and sculpted attacks. Fast flux session binding allows clients to contact gateway nodes in overlay networks. Auto-structure describes how the network builds itself iteratively. Auto-sensing estimates global traffic flows effectively. Auto-resistance deploys traffic scrubbers in real-time to mitigate illegitimate traffic. Auto-flow optimization increases network throughput through route selection. Our system filters traffic that does not conform to expectations of end point servers. Application level attacks will trigger auto deployment of filters. Meanwhile, self-management strategies constantly optimize and heal the underlying network. Simulations were ran to demonstrate effectiveness of the above features. These were then integrated into a prototype. This prototype was deployed to a large number of computers to demonstrate how each feature performs. Results from our deployment has shown that we can mitigate application level attacks effectively. We are also able to perform real-time optimization of our network by varying the number of active nodes and their respective pools. We show how swarm algorithms such as IWD can be used to perform distributed traffic management. This allows us to allocate traffic effectively and increases survivability of the network.

Acknowledgments

I would like to thank my Ph.D. adviser Associate Professor Ng Wee Keong for supporting me over the past two years. His unwavering encouragements and support has been invaluable. He has kept me looking towards the future when distractions were aplenty. His knowledge and experience helped guide my efforts towards the completion of this dissertation. As my informal adviser, I would like to thank Professor Lua Kim Teng for distilling my thoughts and keeping them relevant. His help in ensuring sensibility has gone a long way to shape how I approach problems. Special thanks to Professor Yow Kin Choong my undergraduate and initial postgraduate supervisor. His enthusiasm and heart warming demeanor brings comfort even to the most lost Ph.D. student. Late evening discussions about anything and everything with him have been both an exciting and rewarding experience. I would also like to thank members of Emerging Research Laboratory and Data Intensive Scalable Computing Laboratory for their support and help.

Heartfelt thanks to friends I met in SCE who have became life-long friends, especially Ben, Joyce, Ronnie, Ezra and + 3 more. Their unwavering belief in me for close to a decade has empowered me to accomplish things I have not thought possible. When I had doubts in myself, they didn't. When I started believing in myself, they started exaggerating. I would also like to thank people whom were more then just friends and their company and support from one period to another will always be a treasured memory in time to come.

I want to thank a group of loyal supporters and cell group members whom helped me put my theories into practice either through academic discussions or sheer muscle power. The memories of wiring up close to a hundred computers and servers to my bedroom to test my wildest theories would always be a highlight of my studies. The increase in ambient temperature and noise, coupled with large

power and network lines crisscrossing across levels of the house shall not easily be forgotten.

Deepest gratitude to my family, my father, mother, Tse Min, Seow Chin and Shiang Chien for putting up with me all these years. Whom have shaped my life in ways I cannot imagine. For my father whose mind I was inspired from. For my mother who kept me loved and fed. For Tse Min who sacrifices herself to debug my code and sometimes my mind. For Shiang Chien who had to patiently lower himself to my level to facilitate discussions. For Seow Chin for being an indispensable sister. Thank you.

And lastly, the many others whom I have not mentioned, for without would not have made all this possible. This belongs to all of us.

- Ruiping, 2014

Contents

1 Introduction **9**

1.1 Fragile giants 9

1.2 Stressors - Distributed Denial of Service Attacks and Flash Crowds 10

1.3 Exploring Anti-fragility 13

1.4 Problem Statement 14

1.5 Autonomous Swarm Networks 15

1.6 Publications 17

1.7 Thesis Organization 17

2 Literature Review **20**

2.1 Overview 20

2.2 Preventive measures 20

2.2.1 Attack prevention 20

2.2.2 DoS Prevention 25

2.3 Reactive measures 27

2.3.1 Detection 27

2.3.2 Response 29

2.4 Virtual networks and alternative topologies 31

2.4.1 Overlay Networks 31

2.4.2 Indirection infrastructures 33

2.5 Summary 34

3 Autonomous Swarm Networks **38**

3.1 Overview 38

3.2 Introduction 38

3.3 Overview of Autonomous features 39
3.4 Mitigating multiple attack vectors 43
3.5 Summary 44

4 Autonomous Network Architecture **46**
4.1 Overview 46
4.2 Fast-flux session binding 48
4.2.1 Round-Robin DNS 48
4.2.2 Fast flux service network 50
4.2.3 Session binding 51
4.3 Auto-Structure 52
4.3.1 Seed and Directory services 52
4.3.2 Relays and Neighbour lists 54
4.4 Auto-Sensing (K-iterative sampling) 57
4.4.1 Introduction 57
4.4.2 Simulations 60
4.5 Auto-Sleeping 66
4.5.1 Sleeping and waking nodes 66
4.5.2 Simulations 68
4.6 Auto-Resistance 76
4.6.1 Network funnelling 76
4.6.2 Resisting abnormal traffic flows 76
4.6.3 Simulations 78
4.7 Summary 80

5 Swarm Networking **82**
5.1 Overview 82
5.2 Network and Traffic models 82
5.2.1 Network structure and dynamics 84
5.2.2 Route selection 86
5.2.3 Wardrop Equilibria 87
5.2.4 Optimization through selfish behavior 90
5.3 Allocation and Queues 91

5.4 Network variability and fair usage . . . 93
5.4.1 Highest Capacity (Absolute) . . . 95
5.4.2 Highest Capacity (Proportional) . . . 98
5.5 From tokens to utilization measure . . . 101
5.5.1 Proportional allocation (Exponential) . . . 101
5.5.2 Proportional allocation (Linear) . . . 102
5.5.3 Additional simulations (Capacities: 0.8-1.2,0.5-2.0) . . . 103
5.5.4 Sampling intervals . . . 105
5.6 Water flow . . . 106
5.6.1 Swarm Intelligence . . . 106
5.6.2 Water and Soil inspiration . . . 107
5.6.3 Algorithm . . . 108
5.6.4 Simulations . . . 109
5.7 Summary . . . 111

6 Prototype **113**
6.1 Overview . . . 113
6.2 Software approach . . . 113
6.2.1 Event driven, HTTP Servicing . . . 113
6.2.2 Applying Representational State Transfer . . . 115
6.2.3 Refresh intervals and Timings . . . 117
6.3 Setup . . . 117
6.3.1 Hardware and Networking . . . 117
6.3.2 Generating traffic and benchmarking . . . 121
6.3.3 Servers . . . 122
6.4 Reinforcement, Resistance and Flows . . . 122
6.5 Flow Optimizations . . . 125
6.5.1 Control Scenarios . . . 125
6.5.2 Improving flows . . . 129
6.5.3 K decisions and Auto Sleeping . . . 131
6.6 Summary . . . 131

7 Conclusions and future work **135**
7.1 Discussions 135
7.2 Future work 138
7.3 Final thoughts 142

A Network variability and Congestion **160**

B Additional Waterflow performance **163**

Chapter 1

Introduction

1.1 Fragile giants

With each passing day, Internet transacts higher and higher volumes of data. This increase is brought upon by the exponential growth of web applications and service worldwide. Cisco reports that within the next five years, global IP traffic is expected to reach an annual run rate of 1.6 zettabytes (10^{21} bytes) [38]. Network infrastructures has to constantly evolve to meet this challenge. However, provisioning for this increase is insufficient without addressing complex Quality of Service requirements.

Internet packet routing mechanism has evolved to become highly efficient. However, it is made volatile by the unpredictability stemming from its users. The number of cascades and run-away chain reactions increases with the growth of complexities built into the systems. Volatile request rate results in web servers having significant performance fluctuations. Traffic may fluctuate due to changing popularity of content or localized events. The most severe and unpredictable increase in requests is categorized as flash crowds. A flash event can be predicted if the web site is aware of the possibility of its occurrence[1]. While web services can attempt to provision for predictable flash events, they often fail to predict the demand accurately. Hence offsetting much of the advantages to the said prediction.

While flash crowds are made up on legitimate users, Distributed Denial of Ser-

[1]For example, a predictable flash event would be the streaming of World Cup 2014

vice, DDoS are malicious attacks. DDoSes are attempts made to make a machine or network resource unavailable to its intended users. It generally consists of efforts to temporarily or indefinitely interrupt or suspend services of a host connected to the Internet. A recent report from NSFOCUS [10] based on 244,703 DDoS events in 2013 shows that majority of attacks were short, small in total attack size and frequently repeated against the same target.

Observations by NSFOCUS have shown that in 2013, on average 27.9 attacks occur every hour 91.1% lasts less than 30 minutes. Almost 79.8% of these attacks were less then 50 Mbps and only 0.63% were more then 4 Gbps. Neustar and IDG Research Services [81] suggests that the average hourly attack cost is USD 100K for companies with on-line marketing or commercial web presence (70% of survey participants were involved in e-commerce operations). The impact of DDoS attacks is felt most in the areas of customer service followed by brand damage [10].

The situation is made particularly dire with the ease of hiring botnets to perform DDoS attacks. A fairly typical rate for DDoS botnet rentals is about $200 for 10,000 bot agents per day[2] Some providers offer DDoS services at substantially lower rates depending on geographical and performance requirements. From above, we can infer how vulnerable and fragile today's critical web services are.

1.2 Stressors - Distributed Denial of Service Attacks and Flash Crowds

As reported by Akamai [29] in the first quarter of 2014, Table 1.1 describes the various ports used in the attack.

In Q1 of 2014, Americas continue to account for 49% (139) of all attacks followed by Asia Pacific region with 31% (87) and the remaining 20% (57) from Europe, Middle East and Africa (EMEA). In Singapore, nearly 50% (43) of attacks were concentrated on financial institutions and government sites[3].

The vast majority of attacks experienced by Akamai occurs in layers 5-7 of the TCP stack. These attacks requires much less traffic to be effective. Most

[2]Botnet rental rates are taken from Ghost Market
[3]Customer reports DDoS on Akamai Networks

Table 1.1: Top 10 ports under attack, Akamai

Rank	**Port**	**Ports**	**Q1'14**
1	445	Microsoft-DS	14.0%
2	5000	Universal Plug and Play	12.0%
3	23	Telnet	8.7%
4	80	WWW (HTTP)	8.0%
5	443	SSL (HTTPS)	2.9%
5	3389	Microsoft Terminal Services	2.8%
6	1433	Microsoft SQL Server	2.3%
7	22	SSH	2.0%
8	8080	HTTP Alternate	1.5%
9	135	Microsoft-RPC	1.0%
$\geq$10	-	Others	45%

recently in February, an increase in DDoS using Network Time Protocol (NTP) was observed. Attackers spoofs source IP addresses and sends a small query to a vulnerable NTP server which generates a large amount of response data to the spoofed addresses. This is known as an amplification and reflection attack. The amplification of this particular attack is 556.9x[4] This means that an attacker with access to a 1 Gbps connection could theoretically generate more than 500 Gbps of replies to its target. CloudFlare, a content delivery network and security provider recently reported suffering an NTP reflection attack exceeding 400 Gbps [89]. This is 100 Gbps more than the Spamhaus attack in 2013 [90] which was predominately made of DNS reflection attacks.

It has been reported by the end of 2013, majority of DDoS attacks lasts less than 30 minutes, a trend that was observed similarly in 2012. Practitioners speculate that 30 minutes duration is the industry threshold for most entities to respond to DDoS attacks effectively. Low-rate DDoS attacks below 50 Mbps continue to dominate. These attacks tend to be in form of application layer attacks or hybrid attacks which has the potential of disrupting unprepared service providers. Low-rate attacks are preferred as a larger attack will both cost money and increase its unwanted exposure. Hence to minimize risks to themselves, attackers prefer smaller

[4]Report by US-CERT advisory [4].

attacks. Small attacks are also used as 'scout' attacks to identify vulnerabilities before they begin to attack them repeatedly. More than 50% of these victims suffers from multiple attacks.

Alongside the growth of low-rate DDoS attacks, amplifications are widely adopted. This attack vector uses the asymmetry of protocol communication to magnify attack traffic. Protocols including DNS, NTP and SNMP are good candidates to be used in amplification attacks. Figure 1.1 shows a DNS amplification attack[5]. Very little resources are required to deliver large crippling attacks. The attacker sends a signal to his botnet (1). These compromised hosts sends resolution queries to their respective DNS (2-8) with spoofed return addresses. The reply is then sent to the spoofed (victim) address as a malicious DDoS attack (9). These attacks can be mitigated by network filters such as our proposed overlay network.

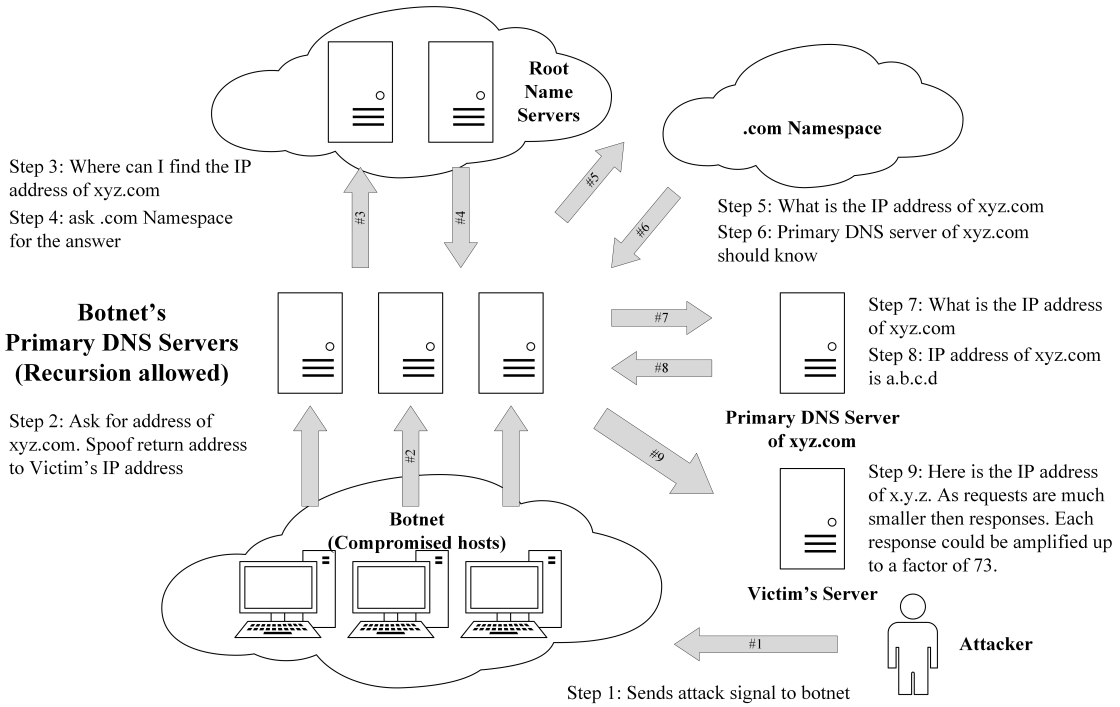

Figure 1.1: DNS Amplification attack

[5]Adapted from nilog's diagram

1.3 Exploring Anti-fragility

With network systems becoming increasingly complex, outcome of interactions becomes harder to predict. Taleb introduces fragility and anti-fragility [110, 109]. Fragility is related to how a system suffers from variability of its environment beyond a certain threshold, while anti-fragility refers to when it benefits from this variability. Systems and services are made robust to a certain level of variability and stress but fail when this level is exceeded. They are particularly fragile towards the uncertainty about the distribution of these stress-ors. Hence any model error and uncertainties increase the probability of any particular service dipping below the designed robustness level and resulting in a collapse. On the opposite, the natural selection of an evolutionary process is particularly anti-fragile. A more volatile environment increases the survival rate of robust communities and eliminate those other that is highly dependent on environment parameters. Similarly, designing a system that follows an evolutionary process allows it to exploit uncertainty and unpredictability to discover superior operational configurations. Table 1.2 lists commonly known fragile, robust and anti-fragile items in the same domain.

Table 1.2: The fragile, robust and anti-fragile

Field	**Fragile**	**Robust**	**Anti-fragile**
Errors	Hates mistakes	Mistakes are perceived as information	Loves mistakes
Regulations	Rules	Principles	Virtue
Medicine	Additive treatment (Medications)		Subtractive treatment (Removing harm)
Psychological	Post-traumatic stress		Post-traumatic growth
Greek mythology	Sword of Damocles[6]	Phoenix[7]	Hydra[8]

[6] An allusion to imminent and ever-present peril faced by those in positions of power. Describes fragility in powerful systems.

[7] Ability to regenerate and respawn upon death. A phoenix is robust.

[8] When each head was cut off, it grew two more. Hydra becomes increasingly powerful as more harm is applied.

Anti-fragility can be associated commonly by the effects of hormesis and mithridatization. Hormesis is a term for favourable biological responses to low exposures to toxins and stressors. A pollutant or toxin showing hormesis thus has the opposite effect in small doses as in large doses. Hormesis is now generally accepted as a real and reproducible biological phenomenon [75], while mithridatism refers to the wilful exposure to toxins in an attempt to build immunity against them. However this addresses a single entity and its response to these stressors, as we build networks of entities, we need to expand this scope of anti-fragility.

Evolution itself is a highly anti-fragile process. It benefits from stressors, randomness, uncertainty and disorder. While individual organisms are relatively fragile, the gene pool takes advantage of shocks to enhance its fitness. This can be viewed in terms of populations. Within a given population, there can be both winners and losers. Where the weak perish and the strong flourish. Overtime, the population gets stronger overall. At a higher level, populations in a cluster of populations (or ecosystems) also experience similar phenomena. Weaker populations shrink while those they are stronger will produce greater numbers. This trend usually benefits the higher level organization.

This hierarchical layering operates iteratively and can be found easily. For example, a tree has many branches, and these look like small tress. This is a manifestation of what is called fractal self-similarity described by mathematician Benoit Mandelbrot. This can also be found in cells. A cell has a population of inter-cellular molecules; in turn the organism has a population of cells, and the species has a population of organisms. A strengthening mechanism for the species comes at the expense of some cells, all the way down and all the way up as well. This iterative structure is incorporated into our system design later.

1.4 Problem Statement

Networks are growing larger and more complex. This increases the burden of operators to manage and protect them. Several methods have been proposed to mitigate DDoS attacks. However, they are (1) difficult to scale and have (2) limited effectiveness. They also only address (3) specific attacks and do not provide wider coverage.

1.5 Autonomous Swarm Networks

To address the stressor of flash crowds and DDoS, we propose an *autonomous swarm network*.

Firstly, knowing that larger structures tend to be more fragile than smaller ones. We start off by decentralizing control and operations. Each components which are iteratively constructed, any single point of failure will not lead to a larger collapse. This leads us to build a swarm network.

To cope and benefit from the variability of the environment, a system should self-organize and adaptive. Centralized designs are vulnerable to model error and subsequent operational uncertainty. Feedback control loops are effective in maintaining quality of service despite fluctuations in network conditions. Feedback control systems can then be developed further to become autonomic systems.

An autonomous system [56, 55] consists of the following features.

- **Self-configuration** - Automatic set-up of components
- **Self-healing** - Discovery and correction of faults
- **Self-optimization** - Monitoring and control of resources
- **Self-protection** - Proactive identification and mitigation of attacks

Contributions

We propose mechanisms addressing features listed in Table 1.3.

Table 1.3: Features and proposed mechanisms

	Mechanism	**Config**	**Healing**	**Optimize**	**Protect**
1	Fast flux session binding	Yes	Yes		
2	Auto structure	Yes	Yes		
3	Auto sensing	Yes		Yes	
4	Auto resistance				Yes
5	Auto flow optimization	Yes	Yes		

Fast-flux session binding (1) provides a mechanism for clients to contact gateway nodes in the overlay networks. By varying responses by our name servers, we are able to allocate gateway nodes as required. We are also able to use various allocation strategies to adapt to incoming traffic. Auto structure (2) describes how the network builds itself iteratively. This technique gives us self-healing properties by maintaining a pre-defined service level with available resources. Auto sensing (3) describes an algorithm for us to estimate global traffic flows effectively and eventually allowing us to complete a feedback control loop. Auto resistance (4) allows us to deploy dynamic real-time filters that reduces illegitimate traffic. Finally auto flow optimization (5) increases network throughput through an intelligent relay node selection strategy.

We then run simulations illustrating benefits of the above features. A prototype encompassing all the above is then deployed using 44 computers with participants from the public. As the system encounters fluctuations, its structure allows it to discover new and sometimes better configuration or routing strategies. This results in the system benefiting from variability hinting on anti-fragility described before.

For example, given the population of nodes in an overlay network. If incoming traffic always remain consistent and beneath a threshold, we cannot identify which nodes are weak or poor performing. However, with variations, response time would vary differently with respect to the node's performance. Hence we are able to pick out poorly performed node and perform appropriate re-configurations to optimize the network. Variations in network conditions also forces the network to change its routing topology over time. This allows us to discover non responsive nodes and remove them, enhancing overall system.

1.6 Publications

Published works

- Ruiping Lua, Chin Han Wah, Wee Keong Ng, 'Cornstarch Effect: Intensifying flow resistance for increasing DDoS attacks in Autonomous Overlays" *Consumer Communications and Networking Conference (CCNC), 2014 IEEE*, 2014

- Lua, Rui-ping, and Wee Keong Ng. "Autonomic swarms for regenerative and collaborative networking." *Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th International Conference Conference on.* IEEE, 2013.

- Lua, Ruiping, and Kin Choong Yow. "Mitigating ddos attacks with transparent and intelligent fast-flux swarm network." *Network, IEEE* 25.4 (2011): 28-33.

Submitted for review

- Ruiping Lua and Wee Keong Ng, 'Antifragility and Autonomic Swarm Networks', *EAI Endorsed Transactions on Collaborative Computing*

1.7 Thesis Organization

This report is organized into 7 chapters. The relationship between the different chapters is illustrated in Figure 1.2.

- **Chapter 1** - Introduction to the fragility of web services and a summary on current DDoS attack trends. We also briefly describe our work to address the above problem.

- **Chapter 2** - Literature Review of various DDoS defense strategies

- **Chapter 3** - Autonomous Swarm Networks introduces our proposed solution. We describe the road map of features developed to realize an Autonomous Computing Architecture.

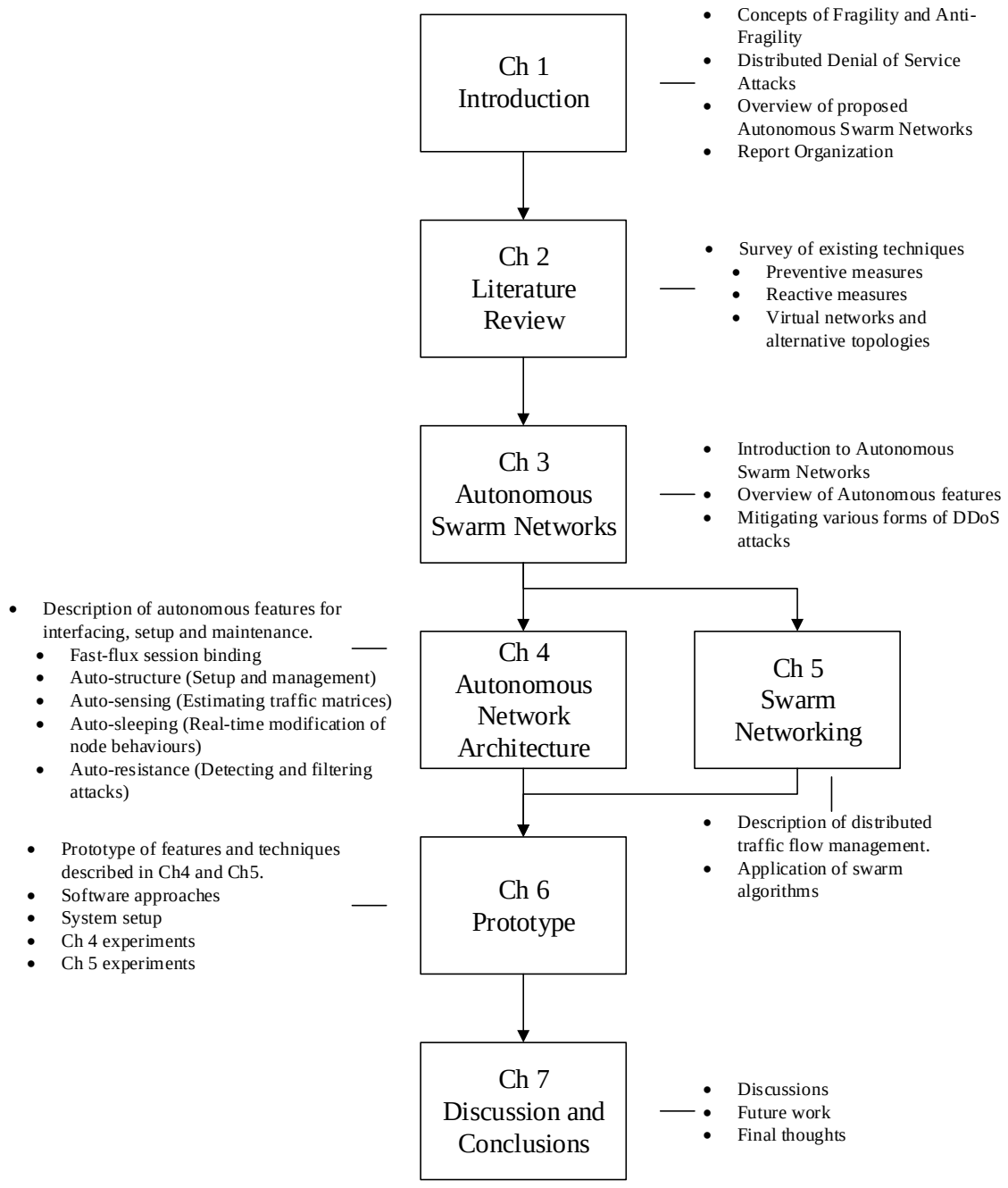

Figure 1.2: Thesis organization

- **Chapter 4** - Autonomous Network Architecture describes various techniques including Fast-flux session binding to allow clients to seamlessly access the overlay network. Using fast-flux networks, we are able to iteratively construct and vary network configurations to adapt to various conditions.
- **Chapter 5** - Swarm networking describes how swarm algorithms can be used for decentralized route selection and flow optimization.
- **Chapter 6** - We build a prototype encompassing strategies and techniques discussed above. We show how each component benefits the system as a whole.
- **Chapter 7** - Discussion about related works, future work and concluding remarks

Chapter 2

Literature Review

2.1 Overview

DDoS defence mechanisms can be categorized by its type of activities: (1) preventive, (2) reactive and (3) virtual networks and alternate topology. A taxonomy of DDoS defence strategies is shown in Figure 2.1 [78, 130].

The type of activity can be either preventive or reactive. In preventive mechanisms, they attempt to eliminate or reduce the possibility of DDoS attacks. They also enable the victims to endure the attack without denying services to legitimate clients. Reactive mechanisms try to mitigate the effects of the DDoS attack by detecting an attack and responding to it. The goals of such techniques are to detect DDoS attacks as early as possible and then respond accordingly. Alternative topologies provides different network infrastructures to make them more resilient to DDoS attacks.

2.2 Preventive measures

2.2.1 Attack prevention

Prevention techniques typically involve modifying systems and protocols on the Internet to reduce and eliminate the possibility of damage from a DDoS attack. Attack prevention measures can be classified as securing the DDoS target by im-

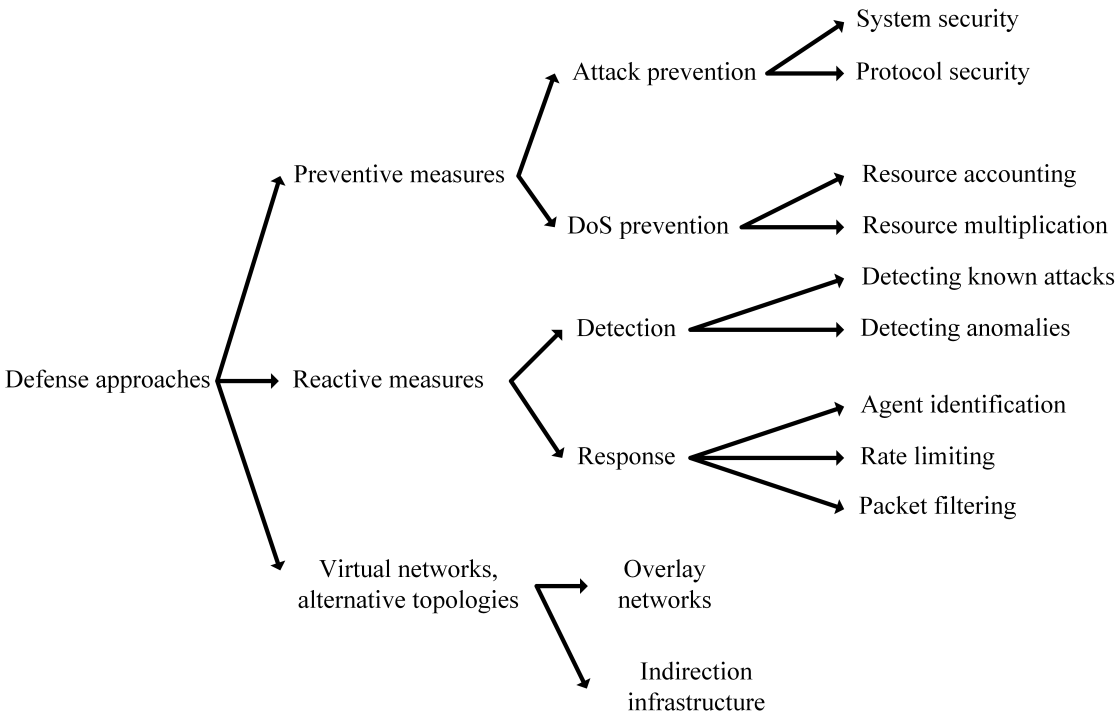

Figure 2.1: Taxonomy of DDoS defence approaches

proving (1) system security or (2) protocol security.

System security

Measures can be taken to secure individual systems. These approaches increase overall Internet security and prevent illegal access to a machine. DDoS are typically conducted using compromised systems. By securing these systems, the number of attackers that could be recruited is reduced. These measures started with Tripwire [59] which monitor access to a server. Tripwire is a host-based intrusion detection system. However, rather than detecting intrusions at the network interface level, it detects changes to file system objects. Tripwire continuously scans the file system objects and compares it with stored hashes in a database. Any changes to the files are reported to the user. A further survey on related IDS development is published by Sandhu et al [95].

System and software patches are also used extensively to address vulnerabilities in systems. Lillich et al [21, 68] first pioneered and patented the mechanism used to providing patches within a computer operating system without the need of re-compiling, relinking or rewriting the code file. Taylor et al then formalized the first software patch architecture [112, 83] which is used widely today to manage patches within an operating system. They continued to develop self-healing architectures [24] and techniques for runtime software adaptations [84, 113].

Worm defence systems curtails propagation of malwares. Work has been carried out to model and detect sophisticated camouflaged worms (C-WORMS) [129, 67]. Reactive anti-body defence generates an inoculation in response to the worm. The inoculation when applied will protect the host from infection [126]. An example of this is using signatures to filter out worm traffic [117] early in its infection phase. Local containment focuses on containing a locally infected machine from sending malicious packets into the network. After worms are contained, systems such as Phagocytes [72] provide computational puzzles to further restrain infected hosts.

Protocol security

Steps can be taken to secure vulnerabilities in protocol design. Many protocols are designed for efficiency and reliability, with little or no security. Examples such as

TCP are vulnerable to TCP SYN attack. TCP SYN attack takes advantage of state retention of TCP for some time after receiving a SYN message. Malicious clients retains enough server resources through bogus half-connections. This exhausts resources required to establish new legitimate connections [28]. Approaches such as SYN caching and cookies mitigates DDoS attacks [62, 134]. Further hardening of these protocols is proposed by Feng et al [32] through establishing security foundations. Protocols can be secured by the following measures.

1. Authentication and Validation
2. Network puzzles
3. Protocol scrubbing and screening

Authentication and Validation Many key agreement protocols [44, 131, 118] are proposed to succeed Internet Keys Exchange [43] (and IKEv2 [53]) to secure protocols. This arrangement allows the responder to verify that it is in a round-trip communication with a legitimate correspondent or user. Liu et al [69, 70] proposes a packet passport system to securely authenticate the source of a packet. Bi et al [14] proposes using source address validations to provide a transparent network service to ensure that packets received and forwarded hold an authenticated source IP address. This is achieved by building a Validation Rules (VR) table that associates each incoming interface of the router with a set of valid source address blocks. Spoofed addresses that do not belong to an authentication IP block are filtered.

Network puzzles Feng et al [30, 31] proposed cryptographically based countermeasure against DoS attacks. When a server comes under attack, it distributes small cryptographic puzzles to the requesting clients. To complete its request, a client must solve the puzzle correctly. This prevents clients from conducting server resource depletion attacks cheaply. They propose implementing this countermeasure within the network protocol stack in order to provide effective protection. Huang et al [48] introduces a resource auction to bid for greater resources. This mechanism filters out users who are not committed to service. Waters et al also propose using a bastion [120] approach to distribute network puzzles to a group of

servers. This out-sourcing of puzzles prevents DoS attacks on the puzzle mechanism as a point of compromise. An example of a TCP network puzzle is shown in Figure 2.2 [30] where options are specified to allow puzzle questions and answers to be included. These puzzles uses willingness and ability of clients to solve them as a crude indication of intent to ensure only legitimate requests are serviced.

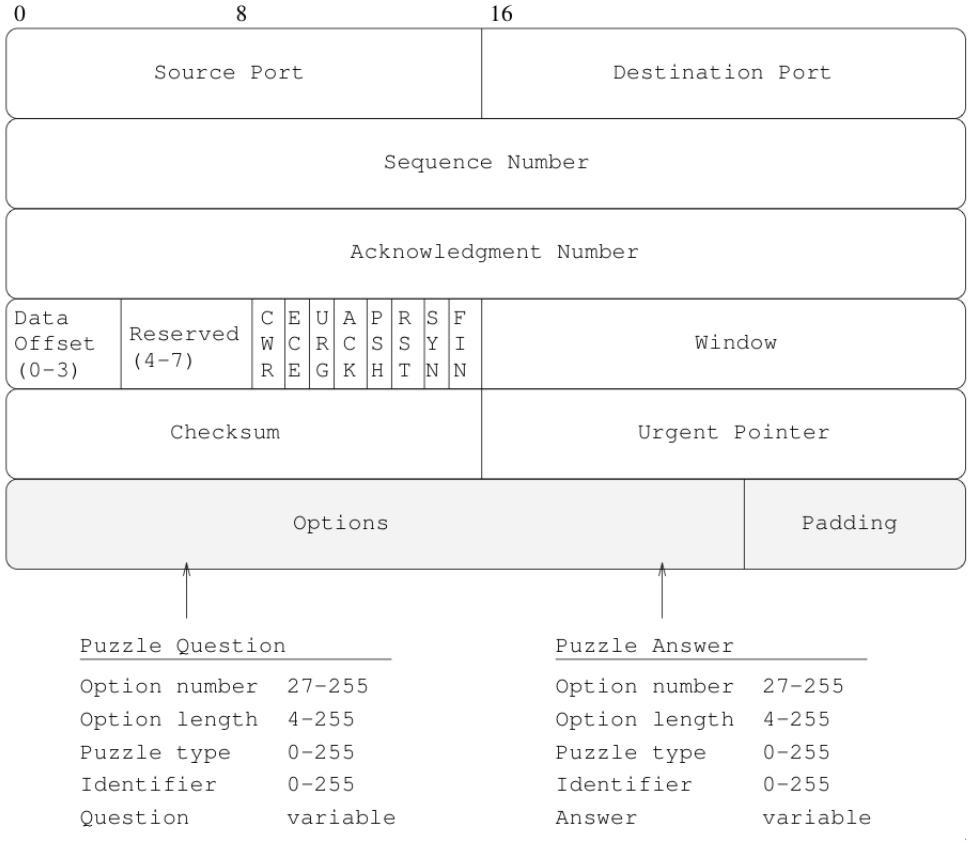

Figure 2.2: Example of a Network puzzle

Protocol scrubbing and screening Malan et al [74] proposes using protocol scrubbers to remove networks attacks at the transport layers. A transport scrubber mitigates transport attacks by removing protocol ambiguity. This allows any downstream Intrusion Detection System to function with assurance. Transport scrubber tries to convert ambiguous network flows into well-behaved network flows. Implementing a transport scrubber can eliminate insertion and evasion at-

tacks that use ambiguities to subvert detection. Attackers are known to use "stack fingerprinting" to identify vulnerable hosts in the Internet. Watson et al [121] proposes using scrubbers to remove markers in network traffic that can be used for malicious "stack fingerprinting" purposes. Protocol scrubbers are widely used to mitigate DDoS traffic [41].

2.2.2 DoS Prevention

Prevention techniques allow the victim to endure DDoS attacks. This can be done through (1) resource accounting or (2) resource multiplication.

Resource accounting

Resource accounting involves managing the server's resources, and providing fair service to legitimate users. Earlier systems experience DoS attacks through the emergence of benign processes. These processes consume CPU resources, depriving other processes of fair CPU usage. A DoS protection base can use a waiting-time policy for malicious processes. Such a waiting-time policy follows the Yu-Gligor model [77]. Resource monitor algorithm and policies follows closely a state-transition model of a resource allocation system. Since then, resource accounting systems such as Escort [105] have expanded to combat network attacks.

Miyoshi et al [80] uses Resource Control Lists (RCLs) to specify protection policies on resources, similar to access control on files. They provide the ability to control the amount of CPU and bandwidth each user is able to consume. This prevents a malicious and unauthorized client from occupying significant resources on a server. Garg et al [35] proposed the implementation of a QoS regulator residing between public networks and the end servers as shown in Figure 2.3. The QoS regulator maintains the state of resource usage at the network layer. Traffic regulation policies are enforced across the traffic class based on resource usage. This technique is then expanded to router QoS [111] and user-centric approaches [49].

There are mechanisms that only allow legitimate clients to communicate with the servers. In an architecture proposed by Anderson et al [125], a client must obtain "permission to send" from the destination. The receiver provides the sender

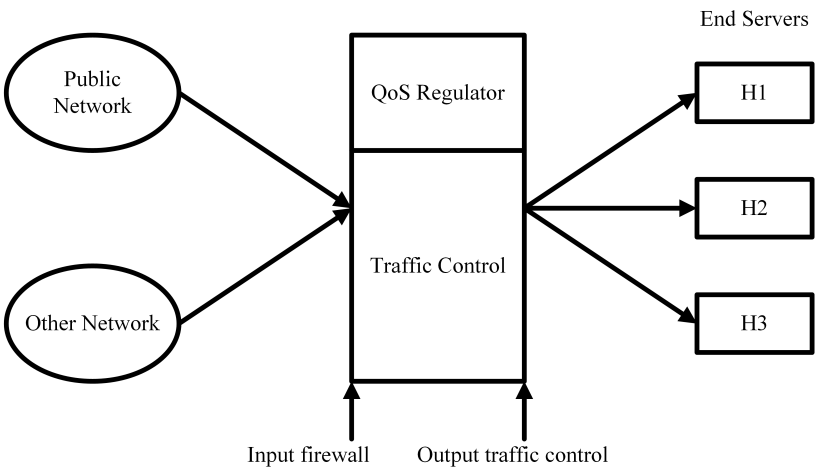

Figure 2.3: QoS Regulator in Network

with tokens upon authentication. The sender then makes use of these tokens to send requests to the server. This allows the server to authenticate if the requests are legitimate and discard the rest. Another example is the use of Keromytis et al's Secure Overlay Services (SOS) [58]. SOS provides message routing capability to authenticated clients while intensively filtering traffic at protected network edges. Anderson continues to improve on SOS by separating the overlay routing and filtering to allow fine-grained control over them (Mayday[8]). This allows different balances of security and performance that achieves better security or performance then pre-existing systems. Gilgor [37] demonstrates how user-server agreements can be negotiated by a third-party before the client is able to communicate with the server. This is similar to resource brokerage.

Resource multiplication

Resource multiplication / replication adds more resources to the victim network. Examples of resource replication includes using Content Delivery Networks to endure DDoS attacks. Some organizations implement large pools of servers with a load balancer to ensure their service availability.

2.3 Reactive measures

Reactive techniques attempts to mitigate the DDoS attack by detecting the attack and responding to it. Strategies to detect the attack can be classified as (1) Detecting known attack patterns and (2) Detecting network anomalies. After an attack has been detected a number of responses are available. Response strategies include the following.

1. Agent identification
2. Rate-limiting
3. Filtering
4. Reconfiguration

2.3.1 Detection

Detecting known attack patterns

Such detection techniques require a database with detailed information of known attacks. Such systems monitor communications to detect attack patterns. Snort [94] by Roesch is one of the most commonly used tools for Intrusion Detection. Snort is a rule-based traffic collection engine. It is a libpcap-based packet sniffer that can be used as a lightweight network packet sniffer and logger. It uses rule-based logging to perform content pattern matching to detect malicious network activity. Bro [87] is a similar IDS tool written by Paxon. Researchers such as Coit et al [22, 9] have continued to develop content matching algorithms that allow faster string matches. They have shown that a Boyer-Moore[1] approach is able to reduce the computation time required to match packet content to a large ruleset. This speedup allows for large rulesets describing a large number of malicious network activities to be used in real-time. However, these systems are only able to detect known attacks, and are unable to identify emerging attacks.

[1]Boyer-Morre string search's efficiency comes from the fact that with each unsuccessful attempt to find a match, that information gained from the attempt is used to rule out as many positions of the text as possible [16].

Detecting network anomalies

Systems detect anomalies by constantly comparing the traffic with an established normal traffic model or expected system performance. Gil et al proposes MULTOPS [36] as a data-structure for bandwidth attack detection. Using MULTOPS, network devices can detect bandwidth attacks by the significant difference in packet rates between the client and the server. Packets are defined to be malicious if they are destined for a host from which too few packets are coming back. This is based on an observation that the packet rate of traffic moving in one direction is proportional to the traffic rate coming from the opposite direction. Mirkovic et al proposes D-WARD [79] as a self-regulating reverse feedback system. D-WARD detects traffic anomalies by retaining traffic information in a flow hash table. Each flow record contains statistics on TCP, UDP and ICMP messages. D-WARD compares traffic with established normal flow models. After each observation, the flows are classified as normal, suspicious or attack. Mahajan et al's Pushback [73] compares the observed loss rate with the expected loss rate to detect an attack. The pushback mechanism was then adopted to use in Boston's airport[101].

Subsequent research uses signal processing and other statistical methods to detect network anomalies. Lakhina et al [61] and Li et al [63] propose using Principal Component Analysis (PCA) to separate normal flows from anomalous flows. By measuring traffic between links, they are able to determine (1) when the anomalies occur, (2) identify the anomalous flow within other flows and (3) volume of anomalous traffic. Callegari et al [17, 18] use wavelet analysis to detect anomalies. They organize the data into high frequency signals and low frequency signals using time as the independent variable. This decomposition process allows them to isolate anomalies and other events in the time series.

Gu et al [40] shows that a network anomaly detection using maximum entropy estimation. Their approach builds a baseline distribution model by learning a density model from the training data. The model is then compared to the observed network traffic. As the relative entropy of the observed network traffic class increases, the chances of anomalous traffic increases respectively. Nychis et al [82] suggests expanding beyond port and address distribution for anomaly detection. They propose using behavioural metrics and flow size distribution as features for

detection. Researchers such as Ming Li [64] documents the Hurst, H parameter of traffic under DDoS, which is used as a detector. Ming Li explains and shows that the averaged H of abnormal traffic tends to be smaller then normal traffic through Fourier analysis [65].

Soule et al [104] proposes using filters and statistical methods to detect anomalies. They construct a Kalman filter to filter out normal, expected traffic. They adjust their filters using a non-linear combination of recent measurement data and their prediction model. Using hypothesis testing, they conduct statistical analysis of the observed network traffic to detect anomalies. Such anomaly detection approaches allow the system to detect previously unknown attacks. However such approaches are also susceptible to false alarms and may improperly categorize legitimate traffic.

2.3.2 Response

Agent identification

When an attack has been detected, tracing mechanisms are used to gather information about the attack source. Agent identification usually has to take place before any other response mechanisms can be utilized.

Agent identification is commonly done through trace back mechanisms [42, 6] such as IP marking or sending ICMP traceback messages. In IP marking schemes, the routers randomly write path information into the packets during forwarding. This is similar to Router Stamping by Doeppner et al [26] . For example, the edge-sampling algorithm writes edge information into the packets. The routers reserves two IP addresses describing the start, end and distance field. When the router decides to mark a packet, it writes it's own IP address into the start field and writes zero into the distance field. If the distance field already zero, it can be inferred that the previous router marks the packet. In this scenario, the router writes it's own IP address into the end field. If the router does not want to mark the packet, it will increment the distance field. In some schemes such as Deterministic Distance Packet Marking (DDPM) by Lee et al [86], a relative distance counter is used instead of an absolute value. Using these marking schemes, victims can reconstruct the attack graph and identify the attackers. Marking on Demand

[128] by Yu et al further improves detection. Snoeren et al proposes Source Path Isolation Engine (SPIE) [107] which uses hash encoding of the IP addresses to lower computation overhead and achieve higher precision. This was later extended to improve performance by Hilgenstieler et al [45].

Alternatively, packet logging requires each router to digest and append the entire travel log of the packet. Hsu et al proposes Path Information Caching and Aggregation (PICA) [47] that records paths of packet streams in fix-length path messages. This allows very precise and easy decoding of attack paths. However, this is cost prohibitive as the packet travels further and further, the amount of path history increases linearly. Gong et al [39] proposes implementing a hybrid of packet marking and packet logging to reduce storage overhead and improve access time. More recent traceback schemes such as RIHT by Yang et al [124] attempts to optimize performance and storage requirements.

Rate limiting

Rate limiting techniques attempts to mitigate DDoS attacks through (1) preventing the spread of DDoS agents and (2) reduce attack traffic to the victim. When an attack is under-way, the victim and attacker's network can impose a limit on the rate of transmission. Rate-limiting does not stop packets from reaching the victim, but is used to make the situation manageable. This is patented [19] and widely used by service providers today.

D-WARD proposed by Mirkovic et al [79] places DDoS defence systems near the source of attacks. They are installed at the source router and serves as a gateway to the larger Internet. D-WARD monitors the behaviour of each peer the source network communicates with and looks for signs of DDoS attack. If a peer is suspected of conducting DDoS attacks, a rate-limit is imposed. Pushback suggested by Mahajan et al [73] moves the implementation of rate-limiters towards the routers. Pushback is a cooperative mechanism that a congested router asks its adjacent upstream routers to rate limits the incoming traffic. This has the effect of pushing the rate-limits closer to the source of the attack traffic, thus minimizing the effects of the DDoS attack.

Packet filtering

Filtering mechanisms use the information from the detection mechanism to filter out the attack streams. Early example of filtering mechanisms such as NetBouncer [114] proposed by Thomas et al provides a basic framework to filter incoming traffic. NetBouncer classifies incoming traffic into 3 action categories: (1) accept and transmit the packet, (2) discard the packet and (3) challenge the sender of the packet. When the legitimacy of the packet is unknown, NetBouncer sends an ICMP echo request to the sender. If the sender does not respond timely, future packets from this host is discarded. Other filtering mechanisms use Deterministic Packet Marking (DPM) / Flexible Deterministic Packet Marking Scheme (FDPM) mentioned in IP Traceback strategies to determine if the packet is legitimate. Using the markings in the different IP packets, Xiang et al is able to determine how DDoS traffic comes from multiple sources and aggregates at one destination. Neural networks can be used to determine attack patterns and subsequently filter them out [122, 52]. Systems such as Arbor Network's Peakflow and Riverbed provide commercial solutions to remove attack traffic. Filtering mechanisms are susceptible to accidental removal of legitimate packets due to incorrect packet marking by the detection model.

2.4 Virtual networks and alternative topologies

These approaches change the topology of the victim network by either adding more resources or isolating systems under attacked or proposing new mechanisms for communications. These are categorized as (1) overlay networks and (2) indirection infrastructures. Reconfiguration overlay networks allows distributed Internet applications to detect and recover from path outages and periods of degraded performance.

2.4.1 Overlay Networks

Andersen et al proposes a Resilient Overlay Network (RON) [7] as an architecture that allows distributed Internet applications detect and recover from network outages. The RON operates on the application layer. Each RON node monitors

the performance of its links and uses the performance to determine how packets are routed. Andersen et al's RON is illustrated in Figure 2.4. Traffic is directed through Node 1 and forwarded to Node 3 via Node 2 (or any intermediate nodes). This served as a generic overlay network from which later works are based on.

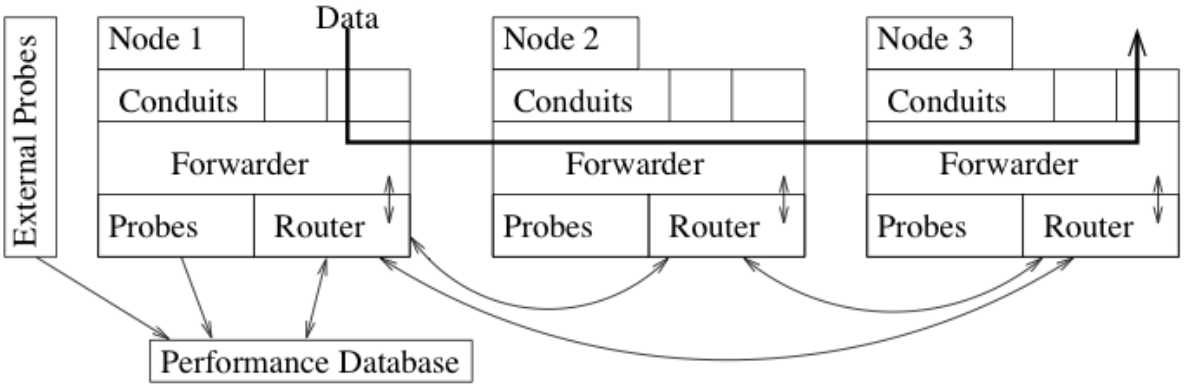

Figure 2.4: Resilient Overlay Network

Overlay networks can be designed to be both social and locality aware [33]. Other researchers such as Touch et al [116] have created multiple virtual networks in their overlays. Touch et al extends the X-bone overlay to provide multiple layered networks to increase the network's resilience to DDoS attacks.

Overlay networks uses various mechanisms to satisfy Quality of Service (QoS) requirements of their system. For example, Subramanian et al proposes OverQoS [108] to enhance the best-effort service of our Internet. They propose a Controlled Loss Virtual Link (CLVL) mechanism (a hybrid between FEC and ARQ) to place an upper bound on traffic losses. Through this mechanism, they are able to smoothen packet losses, prioritize certain packets and provide guarantees in respect to bandwidth and losses.

Other overlay networks such as Tapestry [132] by Zhao et al provides Decentralized Object Location and Routing (DOLR) interfaces to route messages between end-points. DOLR virtualizes the location of resources and through replication of endpoints is able to ensure message delivery in an unreliable network. Recently, flat identifiers are used to improve addressing objects with high mobility [23]. Policy based name routing also addresses scalability of such large networks [91].

Conventional overlay networks provide the packet forwarding service to end-hosts that traditional internet routing cannot make use of. Overlay networks can also be used to filter out malicious traffic. Keromytis et al [57] proposed SOS with the goal of routing only good traffic to servers. Only traffic from good users/clients is allowed. Clients make use of an overlay network to reach the servers. The built-in redundancy of SOS, as well as the secrecy of how packets are forwarded in the network, contributes to its resistance against DDoS attacks. However, this is a purely target-side solution, which can still be potentially overwhelmed by a bandwidth attack at its access points. A spread-spectrum-like communication model spreads their packets randomly across all access points. Alongside these packets, a token is used for authentication. The system prevents the attacker from attacking specific overlay nodes by eavesdropping on clients [115, 88].

2.4.2 Indirection infrastructures

Indirection infrastructures such as publish-subscribe [34] provides an alternate means for web services to communicate with one another. Stoica et al [106, 60] propose a multi-purpose Internet Indirection Infrastructure (I3) by providing rendezvous-based communication for the server and clients. Stoica et al pairs client's data with an identifier. The servers use a trigger to indicate their interest in packets. This infrastructure does not require the sender to be aware of the receiver and vice versa. Figure 2.5 shows an Indirection infrastructure.

(1) Client looks up address of the server. (2) An address is returned. (3) Client requests page/applet from static server. (4) Applet is loaded in the client. (5) Applet sends request to mailboxes. (6) Server retrieves requests from mailboxes. (7) Server deposits responses into mailboxes. (8) Client/Applet retrieves response from mailboxes.

Dixon et al [25] proposes Phalanx that consists of three main components. Phalanx relies on utilizing a swarm that can match an attacking bonnet in strength. It makes use of simple packet mailboxes to allow users to store and pick up messages. The servers in the network must explicitly request a packet from the mailbox for it to be delivered. Lastly, Phalanx uses authentication mechanisms to ensure that data stored in its network is legitimate. However, Phalanx's message interchange

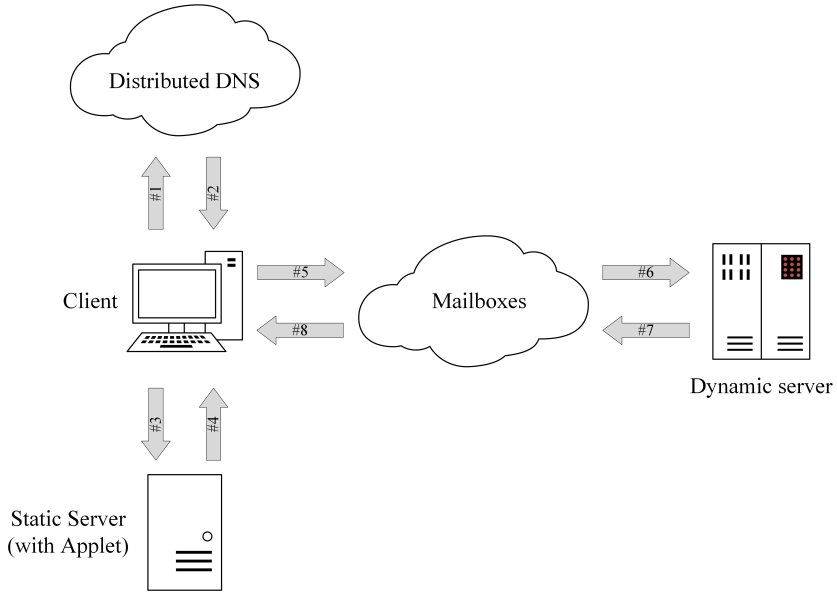

Figure 2.5: Indirection infrastructure

mechanism requires extensive modifications of existing services.

2.5 Summary

The following summaries relevant work to our study.

Preventive measures

- Attack prevention
  - System security
    - Tripwire [59], Further IDS survey [95]
    - Patching [21, 68, 112, 83],
    - Self-healing [24]
    - Software adaptations [84, 113]
    - Detecting worms [129, 67] and filtering [126, 117, 72]
  - Protocol security
    - Syn attacks, caching and cookies [28, 62, 134]

        * Hardening protocols [32]
        * Key agreement protocols [44, 131, 118, 43, 53]
        * Packet passports [69, 70]
        * Source address validations [14]
        * Network puzzles [31, 30]
        * Resource auction [48]
        * Puzzle outsourcing [120]
        * Protocol scrubbing [74, 121, 41]

- DoS prevention
    - Resource accounting
        * Resource accounting and waiting [77]
        * Resource accounting system, Escort [105]
        * Resource control lists [80]
        * QoS regulators [35, 49, 111]
        * Permission to send [125]
        * Secure Overlay Services [58, 8]
        * Negotiating client and server agreements [37]

Unfortunately, cyber-attacks will not be fully prevented. While the above approaches increases the difficulty to launch DDoS attacks, work should also be on surviving an attack when it happens.

Reactive measures

- Detection
    - Detecting known attacks
        * Snort [94]
        * Bro [87]
        * Content matching for IDS [22, 9]
    - Detecting network anomalies

        * MULTOPS [36]
        * D-WARD [79]
        * Pushback [73]
        * Principle Component Analysis [61, 63], Wavelets [17, 18], Entropy estimation [40, 82]
        * Hurst parameter for detection [64, 65]
        * Filters and other statistical methods [104]

- Response
    - Agent identification
        * Traceback mechanisms [42, 6]
        * Router stamping [26]
        * Packet marking [86, 128, 107, 45]
        * Packet logging [47, 39, 124]
    - Rate limiting
        * General/Patent use of rate-limiting [19]
    - Packet filtering
        * NetBouncer [114]
        * Neural Networks for pattern detection [122, 52]

Detection methods described are very useful form of network intelligence. They can increase the survivability of a network by providing previous lead time for administrators to prepare for an incoming attack. Rate limiting and packet filtering mechanisms provide insufficient protection as an attack is already under way.

Virtual networks, alternative topologies

- Overlay networks
    - Resilient Overlay Networks [7]
    - Social and locality aware overlays [33]

  - Multiple virtual networks [116]
  - Best effort through OverQoS [108]
  - Object, name routing overlays [132, 23, 91]
  - Filtering routing and SOS [57]
  - Spectrum communications [115, 88]

- Indirection infrastructures
  - Publish-subscribe architecture [34]
  - Internet Indirection Infrastructure [106, 60]
  - Phalanx [25]

Overlay networks provides a secured infrastructure for communications. They provide network abstraction that allows a network to be configured in a manner that is suitable for various requirements. This flexibility makes it a candidate for our proposed DDoS solution.

Alternatively, indirection infrastructure while effective in eliminating application level attacks to the servers requires extensive modification of existing services.

Chapter 3

Autonomous Swarm Networks

3.1 Overview

In this chapter, we briefly introduce concepts used in designing our solution. We then provide a high level technical overview to show how features proposed are aligned with the requirements of an autonomous system. Coverage against various types of DDoS attacks is also described to highlight advantages of this system.

3.2 Introduction

Autonomic computing focuses on developing a system capable of self-management. Kephart describes this to consist of (1) Self-configuration, (2) Self-optimization, (3) Self-healing and (4) Self-protection [56].

Our network is designed for scalability while maintaining reliable data delivery. To achieve the above, we propose our distributed system with following considerations.

1. **Iterative composition** - An architectural design that allows formation of large scale networks in which each constituent network is composed in a similar structure.

2. **Atomization / Polymorphism** - Functions are made as simple and modular to allow for various ways of re-composition for performing multiple roles.

This iterative structure allows the network to scale to any size. It enhances the robustness of the network. As all the nodes are performing similar functions, when subsets of nodes are disrupted, the rest of the nodes will take over and continue its operations. This allows rapid self-healing of the network and minimizes the downtime. Functions of each nodes are reduced to encourage atomization. This atomization allows each node to take on various roles as required. Individual node does not need to perform complex or large calculations to determine the best route for relaying requests. Instead each node is governed by simple rules and takes input from its surrounding network environment. This leads to the design of a swarm network.

3.3 Overview of Autonomous features

In the course of our study, we have developed the following features.

1. Fast-flux binding - Section 4.2
2. Auto structure - Section 4.3
3. Auto sensing - Section 4.4
4. Auto sleeping - Section 4.5
5. Auto resistance - Section 4.3
6. Auto flow optimization - Chapter 5

The following Figure 3.1 shows how each technique addresses requirements of an autonomous system. Figure 3.2 shows the communication layout of our overlay networks. Exchanges between external clients or users with the swarm are shown on the left. Traffic is then forwarded by the gateways through relays to exits. Exit nodes then contact the respective servers on the right. Responses from servers are then sent to the exit nodes which forwards them back to the gateway and their respective clients. Communications can be categorized as (1) Inter-Swarm - A and (2) Intra-Swarm - B.

The following categorizes each feature developed to an autonomous trait and their use in the overlay network.

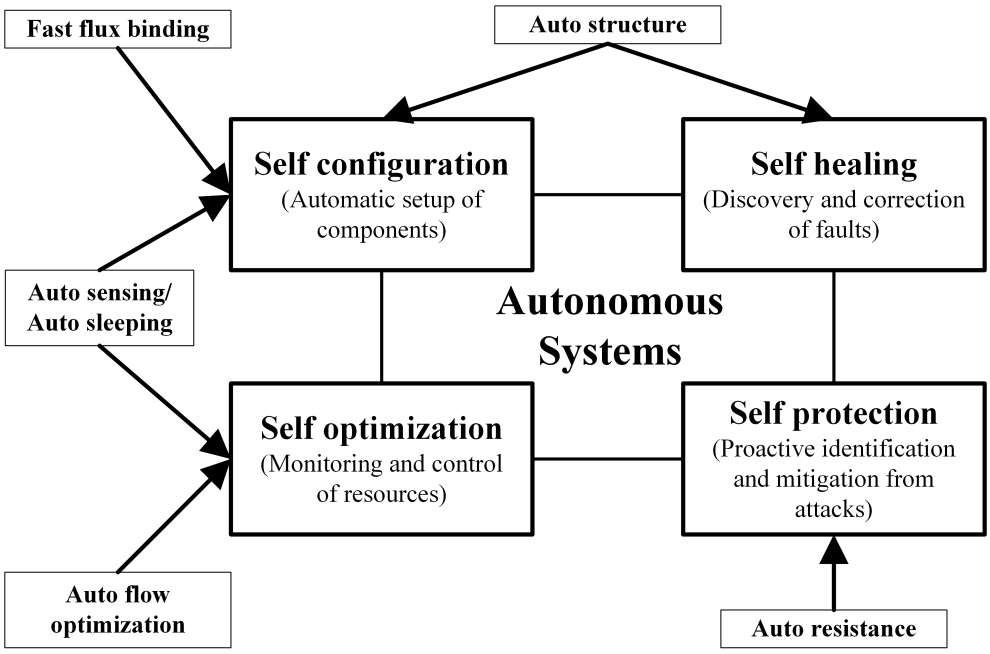

Figure 3.1: Categorizing autonomous features

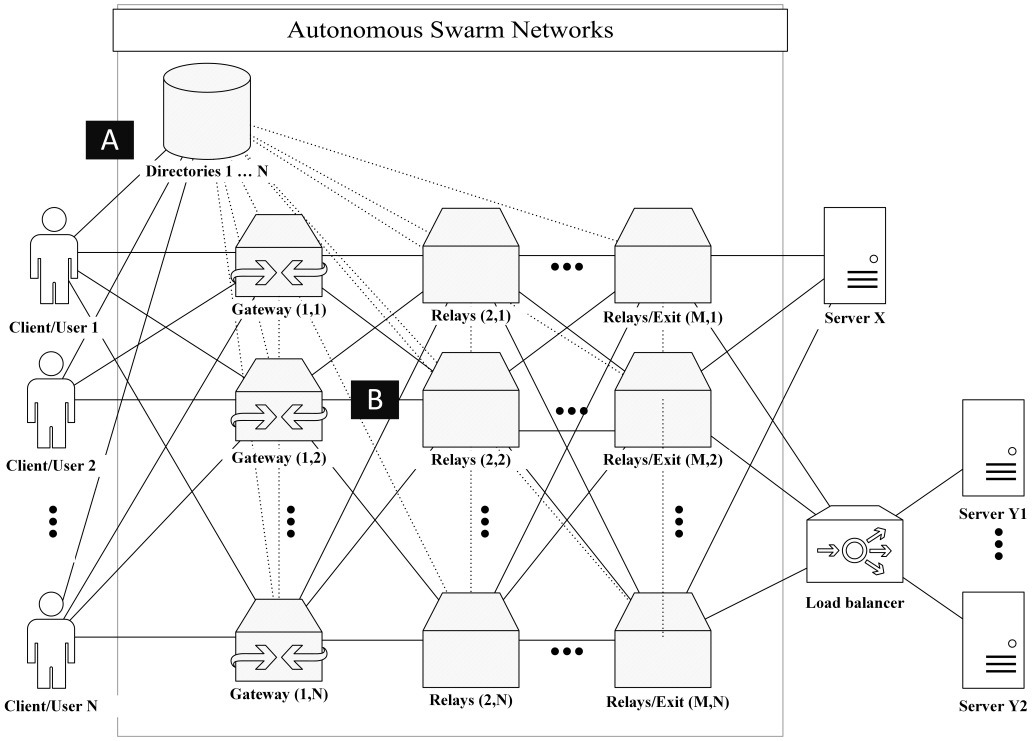

Figure 3.2: Network Communications

Fast flux session binding (A, B) - Section 4.2

Category: *Self-configuration, Self-healing*

Through expanding techniques first demonstrated in fast-flux service networks, we dynamically bind clients to our gateway nodes. As clients resolves desired domain name, it is directed to our gateway nodes. Our gateway nodes relay requests across our overlay network and its respective responses. This allows clients to access the overlay network seamlessly.

Non-conforming requests such as ICMP or UDP destined for a TCP/HTTP application server will be dropped at the gateways. As the number of such requests increases, host table at the Domain Name Server can be modified to reinforce the gateway nodes to cope with increasing attacks. Our Fluxing Domain Name Servers, FDNS periodically checks availability of the gateway nodes and removes non responsive dead nodes and reinforce[1] if necessary to maintain quality of service.

Auto structure (A, B) - Section 4.3

Category: *Self-configuration, Self-healing*

Nodes are able to morph into various roles depending on situational requirements. This allows the network to form its own mesh network without operator's intervention. As each node builds its peer list from each other, the network maintains a high level of resilience even when large number of nodes are experiencing denial of service attacks. This mechanism also allows the network to quickly 'heal' itself when encountering unresponsive nodes.

We are also able to perform real-time changes to pool sizes to increase or reduce public exposure of the network. This allows it to cope with various types of attacks.

Auto sensing, Auto sleeping (B) - Section 4.5

Category: *Self-configuration, Self-optimization*

To perform reconfiguration or optimizations, we need to gather statistics about the network. In large networks, aggregating instantaneous traffic from all nodes is not feasible. This is due to the following factors. (1) Large number of nodes

[1]Reinforcement involves reallocating nodes from other layers.

will require similarly large samples to be taken. Hence introducing large volume of traffic as overhead. (2) Querying large number of nodes cannot be performed in quick successions as it might cause a denial of service within itself. Reducing the query rate will results in out-of-date statistics.

To address this issue, we propose iterative aggregation to form an estimate for global traffic. At the lowest level $k = 0$, each node samples itself. At $k = 1$, each node aggregate its neighbours' $k = 0$ estimates. This is built upon iteratively $k = 2...N$ depending on neighbour and overall network size. This allows us to have a good estimate of global traffic quickly and considerably inexpensive.

Using these estimates, we show that it can be used to optimize the usage of the network. When network load and utilization are low, nodes are asked to sleep. This conserves power and increases efficiency. As the network experiences higher loads, sleeping nodes can be awaken.

Auto resistance (B) - Section 4.6

Category: *Self-protection*

In a distributed network such as this, detecting intermittent or low rate of attacks is difficult. This is due to a (1) lack of oversight authority and (2) load balancing effects. Placements of detectors and scrubbers pose a sensitivity and capability trade off problem.

The network exposes its gateway nodes to the public. To provision for incoming attacks, larger proportion of nodes will be allocated to the gateway (entry) layer. This creates a general funnel shaped overlay network $N_{Layer1} \leq N_{Layer2} \leq N_{Layer3}$. Our FDNS load balances attack traffic across all available gateway nodes. This prevents the gateway nodes from detecting sizeable attack volume. Nodes do not consolidate traffic analyses within the same layer. This avoids the large overhead incurred by sending or receiving traffic histories from large number of peers. However, these attacks will aggregate as it nears the exit layer where there are lesser quantity of nodes. This aggregation will allow nodes nearer to the exit layer to detect attacks much easier. Therefore, if we allocate detectors nearer to the gateway layer, we would have more computational and network resources, but lower sensitivity to attacks. Vice-versa, allocating detectors nearer to the servers would

have higher sensitivity but lesser resources.

To address this, we propose a mechanism for the network to self-determine its optimal allocation of detectors and scrubbers. Starting from the ones nearer to the servers, as more malicious traffic is detected, an inexpensive trigger is used to recruit nodes from preceding layers to start scrubbing. This provides on-demand resources to manage an on-going attack.

Auto flow optimization (B) - Chapter 3

Category: *Self-configuration, Self-healing*

As traffic is routed through the overlay network, it encounters challenges. (1) Inefficient allocation occurs when nodes and links of various performance are used. Without an intelligent allocation, traffic is bottlenecked at lower performing nodes. This problem is exacerbated when nodes select routes without knowledge of other nodes. Any allocation bias towards higher performing nodes will result in an unintentional denial of service attack within the network.

This is addressed by modelling consumption of nodes and links through our swarm algorithm, Waterflow. Using this approach, we can avoid congestions in a decentralized manner. This allows us to optimize flows in between link update intervals hence increasing overall efficiency of our network.

3.4 Mitigating multiple attack vectors

With the above in mind, we propose to mitigate various types of attacks in the following manner. Figure 3.3 shows where each type of attack can be mitigated.

Dropping non-conforming traffic flows, A

Non-conforming traffic includes sequential port scans, ICMP and any transmission that is not specified by the servers. For example, if the destination of a request is a HTTP server on port 80. Any other traffic on other ports will be dropped at the gateway nodes. This addresses generally unsophisticated DDoS attacks.

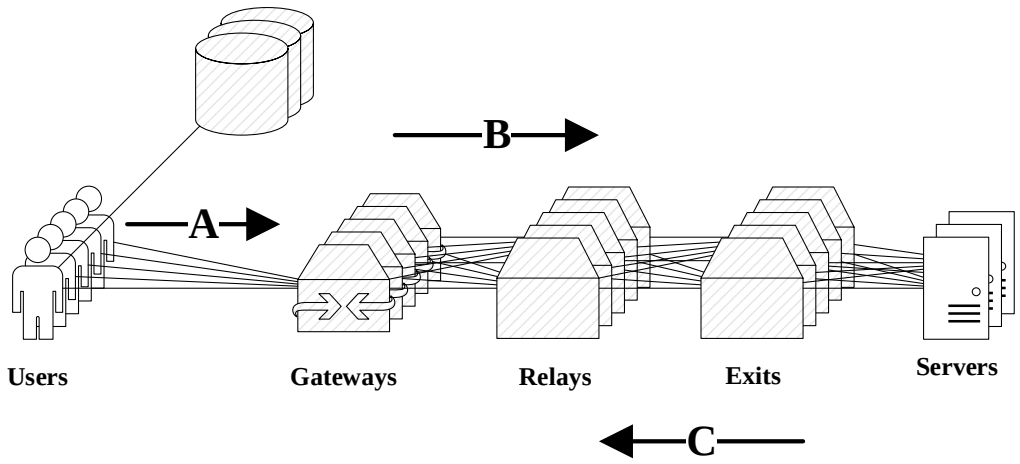

Figure 3.3: Mitigating multiple attack vectors and surfaces

Efficient route selection and routing, B

Crafted requests accepted by the gateway nodes are balanced throughout the overlay network, avoiding congestions. At the exit layer, load balancing spreads the requests to one or more servers. Auto-flow optimizations avoid non responsive nodes and heal the network as necessary. This allows us to extend serviceability of the network for as long as possible to address crafted attacks.

Resisting abnormal traffic flows, C

As requests start to aggregate at the exit layer, various techniques can be used to determine if any particular traffic flow is malicious. For example, source-destination identification can be used to detect and address low-rate DDoS attacks by subsequently blacklisting them from the overlay. This is made possible with our auto-deployment of traffic detectors and scrubbers.

3.5 Summary

In this chapter we elaborated on how an iterative design can allow for rapid self-healing, role polymorphism and scalability. We provide a listing of features pro-

posed and show how they align with an autonomous system. Finally, we highlight how our proposed design will mitigate both network and application level DDoS attacks.

Chapter 4

Autonomous Network Architecture

4.1 Overview

In this chapter, we describe the various techniques we used in creating and managing our overlay network. Figure 4.1 shows how proposed techniques addresses requirements at different stages of the node's life cycle.

Fast-flux session binding provides a mechanism for users to access their content. Content providers do not need to modify their existing service to make use of our overlay networks. Fast flux session binding is elaborated in Section 4.2. Fast-flux session binding allows the building up of the overlay network iteratively. Nodes also share their neighbour lists among themselves resulting in a robust network that is resilient against large node disruptions. These characteristics are further described in Section 4.3.

After the overlay network is raised, for further optimizations to take place, we require sensing mechanism to complete the control loop. An iterative sampling approach is used to estimate global network matrices is shown in Section 4.4. With an estimate of global traffic, we can also perform reconfiguration of the network. We can ask nodes to sleep or wake up depending on perceived network load as described in Section 4.5. Lastly, allocation of detectors and scrubbers are determined in a distributed matter. Starting near the servers, an inexpensive trigger is used to

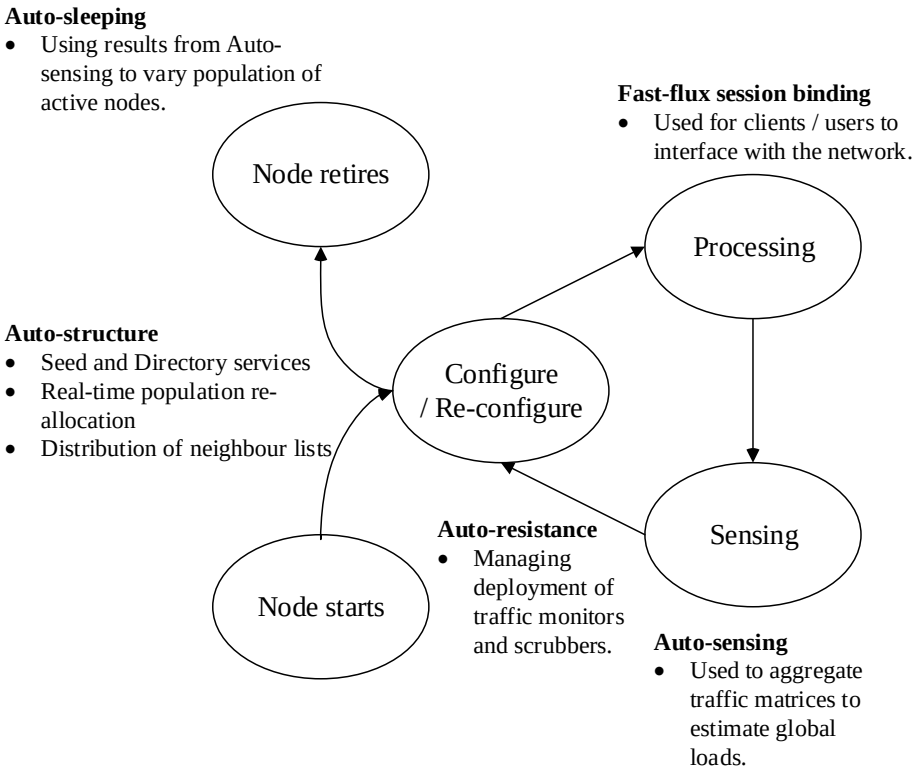

Figure 4.1: Autonomous techniques and node's life cycle

push back scrubbers towards the entry gateways. This on demand recruitment minimizes waste and described further in Section 4.6.

4.2 Fast-flux session binding

We use fast-flux session binding service to provide inter-swarm communication. Components involved are shown shaded in Figure 4.2. User can be any web client and gateway are nodes that respond their incoming requests. Directories correspond to DNSs used to resolve addresses.

We describe how DNS plays a role in dynamic retrieval of web content. We subsequently show how we extend these techniques to provide session binding for our Autonomous Swarm Network. We then implement a single-machine Java simulation that contains each of these elements. An in-built request generator[1] is used. A prototype is then built and described in Chapter 6. Our prototype includes our own DNS implementation and gateway servers while Apache Bench is used to generate incoming traffic.

4.2.1 Round-Robin DNS

Fast-flux service networks shares similarities with Round-Robin Domain Name Service (RRDNS) [3] and Content Distribution Networks (CDN) in their characteristics. They are single service that is hosted from many different IP addresses. A Fast Flux Service Network (FFSN) uses rapidly changing DNS entries to build a resilient hosting infrastructure. This collection of proxies redirects traffic to a central site where the content is hosted. Disabling any of the proxies will not affect the availability of the website itself. The FFSN returns a different set of IP addresses for a particular DNS entry in the same manner as a RRDNS. This results in greater resilience to any service disruption.

A Round-Robin DNS responds to each DNS request with multiple DNS `A` records[2]. Using Domain Information Groper (DIG), the following DNS response is

[1]Requests are distributed uniformly among all gateway nodes. This is to illustrate the viability of the binding mechanism.

[2]Resource Record Type A: Address record, returns a 32 bit IPv4 address, most commonly used to map host names to an IP address of the host. Defined in RFC 1035[1]

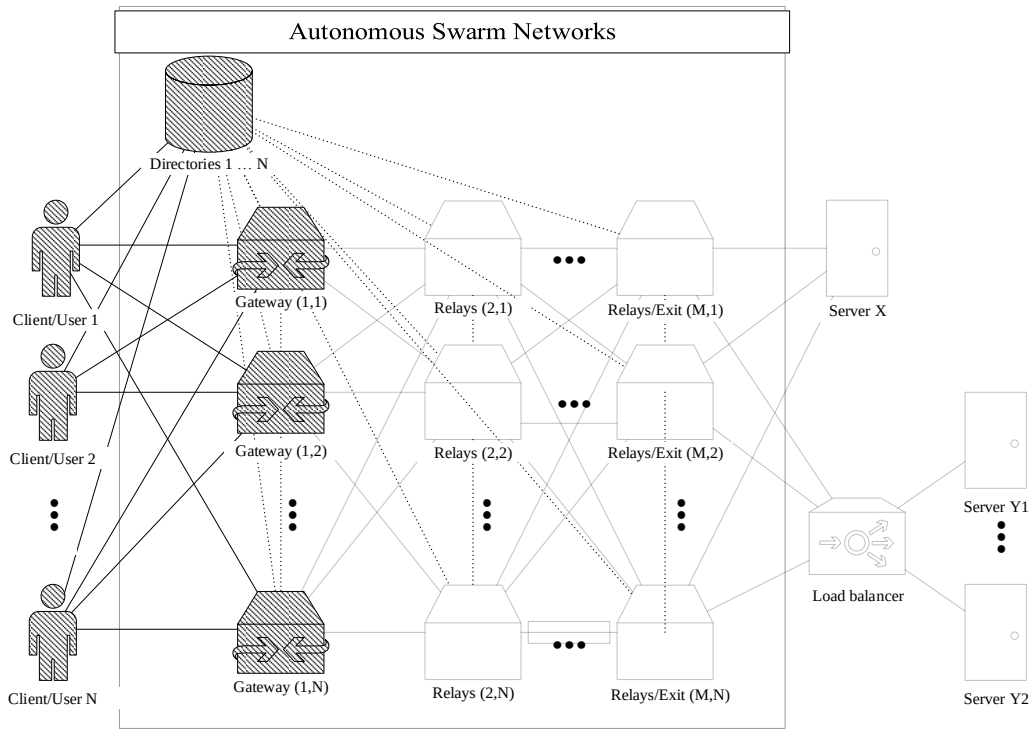

Figure 4.2: Components of fast flux session binding

listed.

```
;; QUESTION SECTION:
;google.com.                    IN      A

;; ANSWER SECTION:
google.com.             180     IN      A       74.125.235.19
google.com.             180     IN      A       74.125.235.20
google.com.             180     IN      A       74.125.235.16
google.com.             180     IN      A       74.125.235.17
google.com.             180     IN      A       74.125.235.18
```

Figure 4.3: Answer from DNS query of Google.com

All these `A` records returns IP address of Google's servers with the same content. The DNS client (or user) will then decide which record to use to locate the resource. Every `A` record has a Time To Live (TTL) parameter that determines how long an `A` record will remain valid. The Networking Working Group recommends using `TTL` values of 1 to 5 days to maximize DNS caching benefits. Shaikh et al [97] shown that small `TTL` values have negative effects. It is therefore important to balance responsiveness against any extra client latency. It is worthy to note that if a DNS query is performed before expiry of `TTL`, the DNS lookup returns the same set of `A` records in a different order. Holz et al have shown that 33% of Alexa Top 500 list's domains uses some form of RRDNS [46]. Through setting `TTL` parameters and the return IP addresses, we can control how users interact with the network.

4.2.2 Fast flux service network

RRDNS and CDN returns several IP address in response to a DNS request. The service network is considered to be online as long as one of the addresses responds correctly. The fast flux domain returns a number of `A` records from a large pool of reliable/unreliable machines as proxies to route an incoming request to another machine. Using this technique, we can build a resilient, robust, one/multi-hop overlay network.

The following Figure 4.4 shows an example of a fluxing domain name [93]. A query was made to resolve the address of `divewithsharks.hk` and a set of IP addresses was returned. However, a second query was made 20 minutes later

returned a different set of IP addresses. This phenomenon is known as a fast-flux network.

```
Query 1

;; WHEN: Sat Feb 3 20:40:04 2007 (~30 minutes/1800 seconds later)
divewithsharks.hk. 1800 IN A 24.85.102.xxx [xxx.vs.shawcable.net] NEW
divewithsharks.hk. 1800 IN A 69.47.177.xxx [d47-69-xxx-177.try.wideopenwest.com] NEW
divewithsharks.hk. 1800 IN A 70.68.187.xxx [xxx.vf.shawcable.net]
divewithsharks.hk. 1800 IN A 90.144.43.xxx [d90-144-43-xxx.cust.tele2.fr]
divewithsharks.hk. 1800 IN A 142.165.41.xxx [142-165-41-xxx.msjw.hsdb.sasknet.sk.ca]

divewithsharks.hk. 1800 IN NS ns1.world-wr.com.
divewithsharks.hk. 1800 IN NS ns2.world-wr.com.

ns1.world-wr.com.  85248 IN A 66.232.119.xxx [HVC-AS - HIVELOCITY VENTURES CORP]
ns2.world-wr.com.  82991 IN A 209.88.199.xxx [vpdn-dsl209-88-199-xxx.alami.net]

Query 2

 ;; WHEN: Sat Feb 3 21:10:07 2007 (~30 minutes/1800 seconds later)
divewithsharks.hk. 1238 IN A 68.150.25.xxx [xxx.ed.shawcable.net] NEW
divewithsharks.hk. 1238 IN A 76.209.81.xxx [SBIS-AS - AT&T Internet Services] Response Received!
divewithsharks.hk. 1238 IN A 172.189.83.xxx [xxx.ipt.aol.com] NEW
divewithsharks.hk. 1238 IN A 200.115.195.xxx [pcxxx.telecentro.com.ar] NEW
divewithsharks.hk. 1238 IN A 213.85.179.xxx [CNT Autonomous System] NEW

divewithsharks.hk. 1238 IN NS ns1.world-wr.com.
divewithsharks.hk. 1238 IN NS ns2.world-wr.com.

ns1.world-wr.com.  83446 IN A 66.232.119.xxx [HVC-AS - HIVELOCITY VENTURES CORP]
ns2.world-wr.com.  81189 IN A 209.88.199.xxx [vpdn-dsl209-88-199-xxx.alami.net]
```

Figure 4.4: Fast-flux example

As observed, when repeated queries are performed on a fast-fluxing domain name, different resolution replies are given. Fast-flux Service Networks can be very large, consisting of thousands or tens of thousands of systems.

4.2.3 Session binding

Figure 4.5 shows how our session binding takes place.

When a client connects to the server, it tries to resolve the domain name of the service provider (1). As the ownership of the server's domain is delegated to the swarm network, it will return the IP address of an entry node (2, 3). The entry node will forward the client's request across the swarm network (4). The packet is then routed through the swarm network until it has reached an exit node closest to the server. The exit node will then forward the request to the server.

The server responds to the exit node, which passes the reply through the swarm back to the entry node. Entry node then returns the result back to the client (6). This arrangement allows both parties to function without any changes to existing protocols. Hence, we are able to provide a seamless and hidden transport network that increases the robustness of current web services.

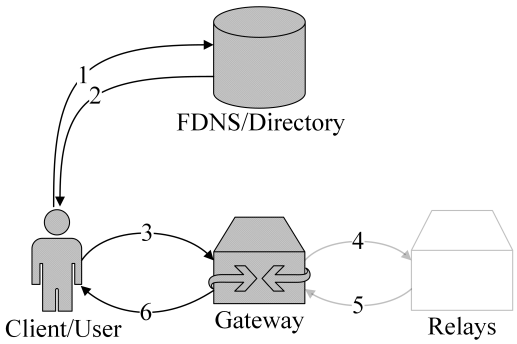

Figure 4.5: Fast Flux Session Binding

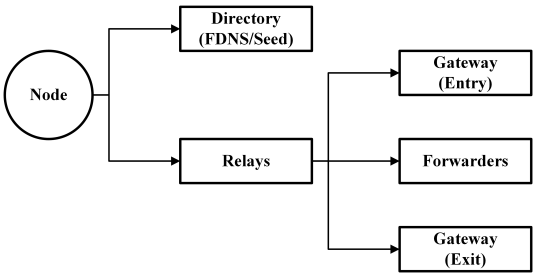

Figure 4.6: Node Roles

4.3 Auto-Structure

The nodes in the network have the following roles shown in Figure 4.6. Each nodes can morph into any required roles depending on the situational requirements. As a node starts, it decides if it will be a FDNS/directory or a relay. Using spatial information, the node can then further specialize itself to be either a gateway (entry/exit) or forwarder node. This allows the system to rapidly heal and recover from disruptions.

4.3.1 Seed and Directory services

The fluxing domain name server (FDNS) described previously is a key component to how the network forms over time.

When a node starts, it begins by querying a pre-fixed domain name. The domain name resolution request will be iteratively forwarded to our fluxing domain name server. If there is no response from a valid FDNS node, it will promote itself to become a FDNS/seed node. Subsequent queries from each node will register

itself with the FDNS/seed and receive a response directing it to morph into a relay node. Hence the first FDNS node is known as a seed node, as it facilitates all registrations that come after it.

The seed node keeps a table of all registered nodes. This has 2 uses. (1) Serving initial list of peers to relays. After registration, seeds will provide relays with a list of peers to build their neighbour list. This significantly shortens the time required to build their list. (2) Providing a host table to be used for FDNS services. Gateway (entry) nodes are selected from this host table. The number of gateway nodes can be constrained depending on the total number of nodes available. For example for an overlay network requiring strict separation between entry and exit using at least 3 layers can have a $1:1:1$ allocation. Hence the number of nodes to be used as gateway (entry) can be $\frac{1}{3}$ respectively with nodes of homogeneous performance. This can be changed to suit varying types of performance in the node population and other overlay requirements.

Real-time population re-allocation

We can also increase the number of gateway nodes to compensate for increasing non conforming attack traffic. Before any reconfiguration can take place, we assume there's a mechanism that provides real-time estimates of performance and utilization of the network. These approximations are integral in determining the current network conditions and are elaborated in a later section.

Signals to reconfigure pool sizes can come from either (1) Directory level (top-down) or (2) Relay level (bottom-up). Seeds operating at the directory level can sample their relay nodes to have estimate on how each pool is performing. The pool size can then be varied to maintain certain parameters. For example, in a given $1:1:1$ network of homogeneous nodes, if the nodes in the gateway pool (1st pool) is experiencing higher loads then other (2nd and 3rd) pools, we can infer that it is experiencing significant amount of non-conforming attack traffic that is not forwarded to other pools. Asymmetric overloading of any single pool will reduce overall performance of the network, hence more nodes can be re-allocated from other pools[3].

[3]Re-allocation must take place within constraints to avoid other pools from starvation

At the gateway/relay level, nodes can also report on the volume of attack traffic. At certain thresholds they can send a trigger to its neighbours or peers to perform the reconfiguration. However, congestion and network slow-downs are frequent occurrences that happen without any attack and will be undetected. Similarly to the technique performed by seeds, relays can also perform respective sampling to estimate global performance. This is shown to be more efficient than gateway/seed (top-down) approach in the following section.

4.3.2 Relays and Neighbour lists

After nodes become relays, they will begin building their neighbour list. They receive neighbour information from (1) initial list from Seed/Directory, (2) neighbour list of neighbours and (3) neighbours who relayed information through them.

A node's neighbour list is constantly updated when new nodes make contact. When a node in the neighbour list becomes un-contactable, it is removed. When the node receives an initial list of neighbours, it proceeds to establish contact with them. However, the initial list from the directories is typically small to conserve bandwidth. To further expand the list, each node can request for their neighbour's immediate neighbour list. This expands each node's neighbour list without overloading the directories.

However, special care should be taken in the allocation of neighbours to avoid chronologically biased configurations. This effect is particularly pronounced in the early phases of network set-up. In the beginning, when N the total number of nodes is less or approximately same as n neighbour list size, newly registered nodes will frequently receive similar or identical neighbour list. Nodes that are registered earlier appears in higher frequencies then those that comes later. As N grows to $N >> n$, this effect is reduced. However, these artefacts are still present in the earlier nodes. As nodes are not aware of how many nodes have them on neighbour list, a skewed allocation and discovery process will cause traffic aggregation at the earlier nodes and reduce overall performance. Having many similar neighbours list will result in a problem as illustrated in Figure 4.7. In this scenario, multiple nodes have the same next-hop peer. This is described as an unsympathetic selection problem whereby a node favouring a particular peer does

not take into consideration that other peers also do the same. Hence resulting in premature congestion in higher performance or favoured nodes.

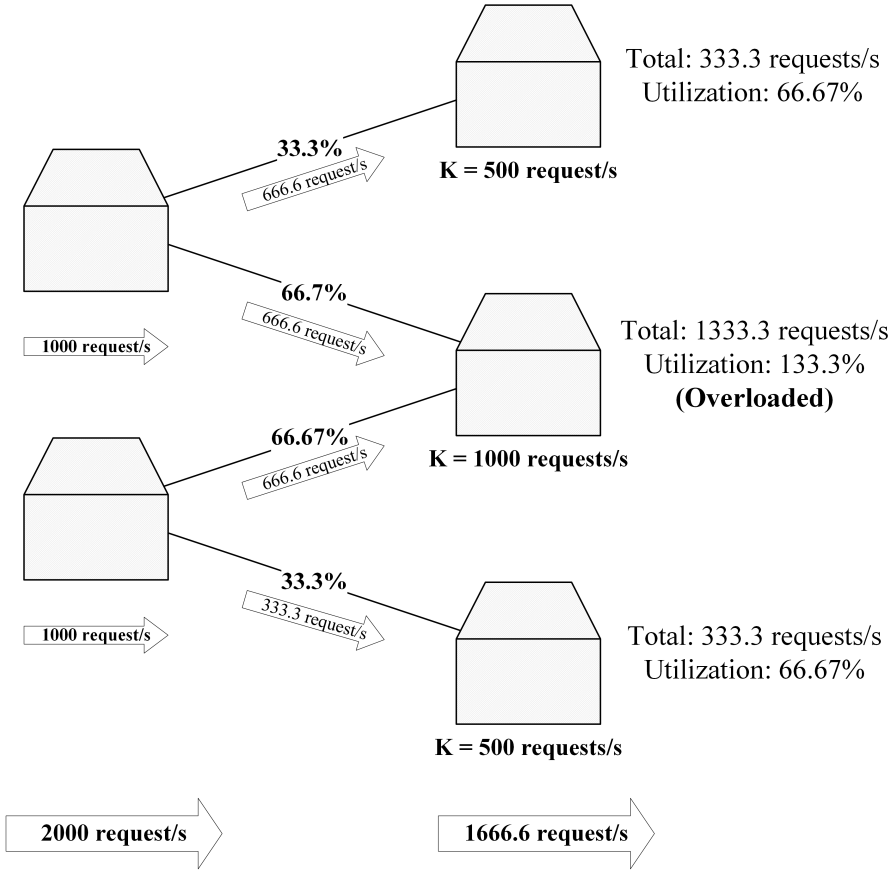

Figure 4.7: Unsympathetic selection problem

Accounting at the Seed/Directory can reduce the number of times a particular node has been shared with incoming peers. It tries to ensure that over a period of time, every registered relay is shared fairly with incoming peers. However, clustering effects still occurs due to other reasons such as race conditions, whereby a particular node will always return their neighbour list faster than others. This results in many nodes having similar neighbours. To ensure there is sufficient diversity in neighbour lists, we propose dropping neighbours periodically and adding other hosts. At a given interval, the lowest performing node by throughput is dropped and another node is chosen from the neighbour list. To ensure that optimal performance will be maintained or achieved, throughput history is kept. If

throughput falls, a new or the previous node can be selected[4].

To maintain integrity of the network, the peer list of each nodes need to be current. At each update (1) peer list and (2) peer status are transmitted. This continuous chatter adds to the overhead of the overlay network as shown below.

$O(t)$ represents all the overhead traffic generated across all nodes in the network. $o_{req}(t)$ and $o_{reply}(t)$ represents requests and response traffic. k is a binary that turns on or off active updates. n_i and n_j refers to the size of the neighbour/peer list of i and j respectively.

$$O(t) = \sum_{i=1}^{N} (o_{req}(t) + o_{reply}(t)) \tag{4.1}$$

$$o(t)_{req} = \frac{k \times \sum_{i=1}^{n_i} d_{req}}{T} \tag{4.2}$$

$$o(t)_{reply} = \frac{\sum_{i=1}^{n_i} \sum_{j=1}^{n_j} d_{reply}}{T} \tag{4.3}$$

Hence, we can sum both Equations 4.2 and 4.3 and iterate it across all nodes to describe overall traffic.

$$O(t) = \sum_{i=1}^{N} \frac{(k \times \sum_{i=1}^{n_i} d_{req}) + (\sum_{i=1}^{n_i} \sum_{j=1}^{n_j} d_{reply})}{T} \tag{4.4}$$

The equations are further simplified.

$$O(t) = \frac{N \times ((k \times n_i \times d_{req}) + (n_i \times n_j \times d_{reply}))}{T} \tag{4.5}$$

$$O(t) = \frac{N n_i}{T}(k d_{req} + n_j d_{reply}), k \in 0, 1 \tag{4.6}$$

Active updates requires a control message to be sent to the node before the node responds with peer information. Passive update does not require a control message, but each node sends its update at specified intervals. Although passive updates have smaller overhead, plenty of security measures requires the former

[4]In our prototype, we keep the neighbour list consistent (without re-selecting or dropping relays) to evaluate various techniques and optimizations

active updates.

In our scenario, the peer list remains the same for all nodes. A different peer list size can be used for node variants. Given that peer list in our design remains the same throughout, therefore $n_i = n_j$. From Equation 4.6 inter-overlay overhead is linearly dependent on the peer size n_i and inversely dependent on the update interval T. Typical values of $d_{reply} \gg d_{req}$, hence $o_{reply}(t)$ is a much larger contributor to $O(t)$.

To ensure that these control messages do not penalize network performance too heavily, we keep update frequency low (4 updates/min) while adjusting peer size n_i. For our simulations, we typically keep $\frac{n_i}{N}$ between 0.008 and 0.016 to limit these overhead traffic.

4.4 Auto-Sensing (K-iterative sampling)

4.4.1 Introduction

Traffic is generated mainly due to relaying data through the network[5]. To allow us to auto manage the number of nodes for a given traffic load, we require an efficient mechanism to estimate global traffic. This estimation can then be used for a feedback control loop shown in Figure 4.8. An initial set of parameters are issued by the controller. As the network operates, performance metrics such as throughput and success rate are gathered. These statistics are used to determine if any changes should be issued by controllers. This approach is an effective use of feedback and provides a simple and computationally inexpensive mechanism to address non-linear changes when parameters are changed.

Determining traffic matrices for the entire network could be very costly as the network scales upwards. Each node has to have an estimate of the overall network load to adjust its operating parameters effectively. We cannot perform centralized traffic measurements because (1) it creates a single point of failure and (2) it would take too long for such estimates to be useful. We require a mechanism that is distributed and relatively lightweight in terms of memory and network traffic to

[5]Our study only considers unidirectional traffic flows and the number of peers within a given sampling window remains constant.

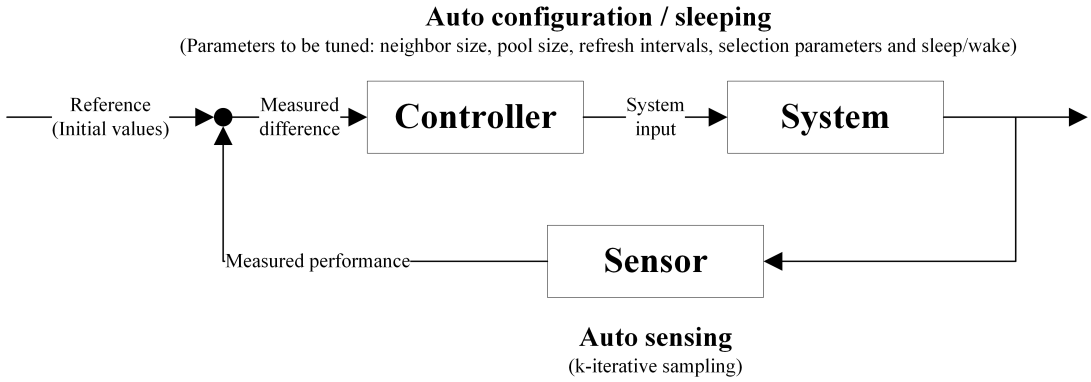

Figure 4.8: Feedback control loop

minimize operating overheads.

We propose using the following iterative query approach to aggregate traffic matrices. For 2 given nodes, we describe traffic between them as X_{ij}.

$$X_{ij}(t) = P_j^e \times B_{ij}(t) \tag{4.7}$$

where P_j^e is the probability of j node existing in i peer list. $B_{ij}(t)$ is the data throughput between i and j. Every peer in the network gets a list L_i from the seed/tracker which is a subset of nodes in the network.

The probability of getting a specific peer j is

$$P(x = j) = \frac{1}{N} \times |L_i| \tag{4.8}$$

When two nodes begin transferring data, the flow throughput can be described using the classical TCP performance model [85] TCP_{bw}.

$$TCP_{bw} = \frac{C \times MSS}{RTT\sqrt{p}} \tag{4.9}$$

whereby C is the number of TCP ACK packets, MSS is the maximum segment size, RTT is the round trip time and p is the packet drop probability. In Equation (4.9), C and MSS are constants regardless of host chosen. However, RTT and p are typically dependent on network distance and congestion[6]. Hence, we assume

[6]In our study, we consider network distance as the dominant influence in bandwidth. Congestion comes into effect only when network load reaches its maximum capacity.

the following.

$$TCP_{BW} \approx \frac{1}{(d_{ij})^s} \tag{4.10}$$

If all traffic from a node is divided among its peers, the uploading capacity $U_i(t)$ of node i is allocated with respect to $\frac{1}{(d_{ij})^s}$[7].

Therefore we can describe B_{ij} in the following.

$$B_{ij}(t) \approx \frac{1}{(d_{ij})^s} \Rightarrow B_{ij}(t) = \frac{\frac{1}{(d_ij)^s}}{\sum_{h_{i=0}}^{i=j} \frac{1}{(d_ij)^s}} U_i(t) \tag{4.11}$$

Hence

$$X_{ij}(t) = \frac{|L_i|}{N} \frac{\frac{1}{(d_ij)^s}}{\sum_{h_{i=0}}^{i=j} \frac{1}{(d_ij)^s}} U_i(t) \tag{4.12}$$

To estimate global traffic of our network, we can aggregate traffic matrices defined by Xu et al [123]. We make use of (similar) notations.

- Aggregation level k
- Peer cluster h_i^k is an i-th set of peers
- Peer group H^k consists of all the peer clusters at the aggregation level.

Using the above definition we can then view each individual peer as a single peer cluster h_i^0 as part of a group H^0. The level k is then recursively aggregated to derive new $k+1$ clusters. The traffic between two groups can be iteratively calculated by aggregating its respective peers. This can be expressed in a general form in Equation 4.13.

$$X_{ij}^k(t) = \sum_{h^{k-1}}^{h^k} \sum_{min}^{max} X^{k-1}(t), \forall k > 0 \tag{4.13}$$

This is then used to estimate the throughput of our network.

[7]This correspond to the minimal utilization / maxmium 'throughput' model used earlier

4.4.2 Simulations

A Java simulation is written to simulate the performance of these 1000 nodes stochastically. Each node has a token bucket which is used to shape traffic transmitted between them. Simulations ran for 200 seconds each. Figure 4.9 shows that the distribution of network capacity for the following simulations is approximately uniform. Tokens vary from 1×64 to 10×64 across 1000 nodes. These nodes are then divided across 4 layers equally.

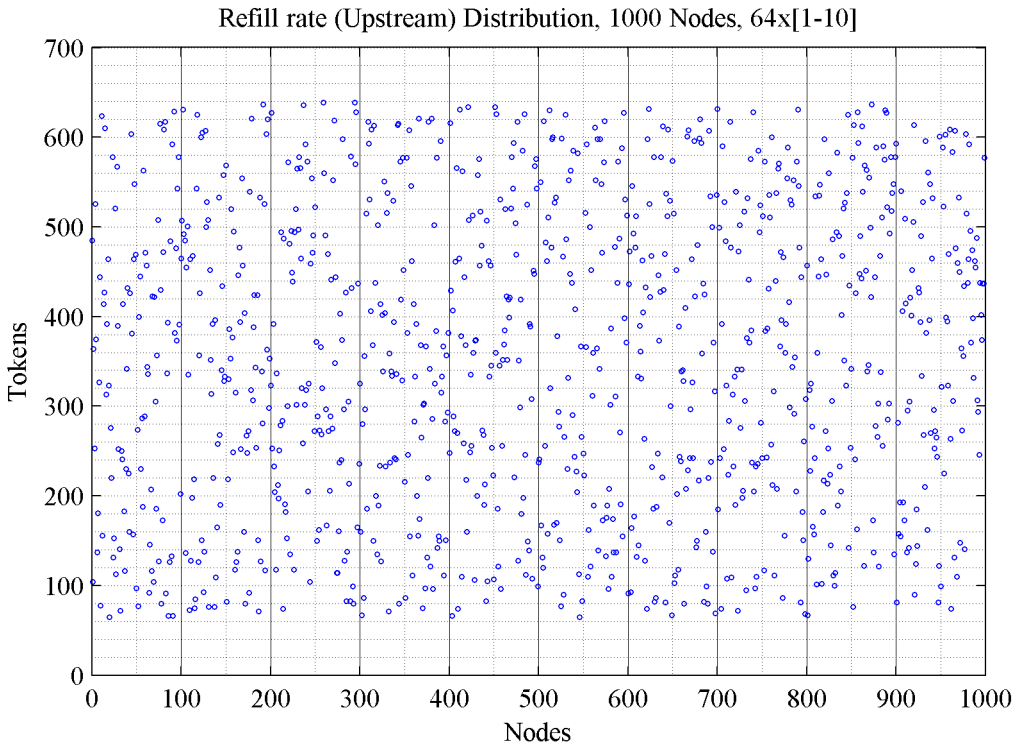

Figure 4.9: Distribution of refill rates

The iterative estimate is then used on the following scenarios:

1. **Stochastic Fair Queuing/Distribution (SFQ)** - Figure 4.10 shows the performance of SFQD allocation. Figures 4.11, 4.12 and 4.13 show the deviation $\delta u_i(t)$ of estimated traffic of each node from the actual global traffic.

$$\delta u_i(t) = \frac{\sum_{j=1}^{p_i} u_{(j,k)}(t)}{p_i} - \frac{\sum_{i=0}^{n_z} u_j(t)}{n_z} \tag{4.14}$$

where p_i is peer list, $u_{(j,k)}$ is k-value of node j. n_z refers to the nodes in a particular zone. In the following deviation results, $P_i = 8$. In the following scenarios, we assume u_i has already summed all flows from a node i. Hence, $k = 0$ refers to $\frac{1}{P_i} \times \sum_{j=1}^{P_i} u_j(t)$. The maximum error L_s is the largest error encountered at any instance of time. This is typically the maximum positive deviation L_{max} minus maximum negative deviation L_{min}. The respective L_s values are $L_s, k_0 = 0.7171$, $L_s, k = 1 = 0.5592$ and $L_s, k = 2 = 0.2790$.

As K-estimates are built over time, initial estimates are inaccurate. Figure 4.14 includes the initial estimates. We define an initialization window (e.g. t=5) which eliminates the use of partially initialized values.

We show that every iterative $(k+1)$ estimates yield a smaller deviation from the global traffic volume. At k=2, the deviations are mostly below 5%.

2. **Proportional allocation (Exponential)** - Exponential PA simulations have shown to provide a large variation in node utilization. Figure 4.15 shows the node utilization for this simulation. The variations are shown to be larger than SFQ. Results in Figure 4.17 ($L_s, k_1 = 0.8008$) and Figure 4.18 ($L_s, k_2 = 0.4013$) show that even with large variations, $k = 2$ estimates can be fairly accurate. With larger variations, k-estimates are poorer then those with less.

3. **Proportional allocation (Linear)** - Linear PA is used to simulate a network with less variation as compared to exponential. The results show that if traffic flow can be minimized, k-estimates will improve accordingly.

   Figure 4.19 shows the load experienced. Figure 4.20 shows the deviation between actual load and $k = 0$ estimates at any given time. The maximum error is $L_s, k_0 = 0.6182$. Figure 4.21 shows the deviation $L_s, k_1 = 0.5892$. Figure 4.22 shows the deviation $L_s, k_2 = 0.2891$.

4. **Waterflow (Proposed later in Chapter 5)** - Water flow attempts to minimize variations in the traffic flow. This allows better k-estimates then the aforementioned.

   Figure 4.23 shows the load experienced. Figure 4.24 shows the deviation

between actual load and $k = 0$ estimates at any given time. The maximum error is $L_s, k_0 = 0.3149$. Figure 4.25 shows the deviation $L_s, k_1 = 0.5893$. Figure 4.26 shows the deviation $L_s, k_2 = 0.2610$.

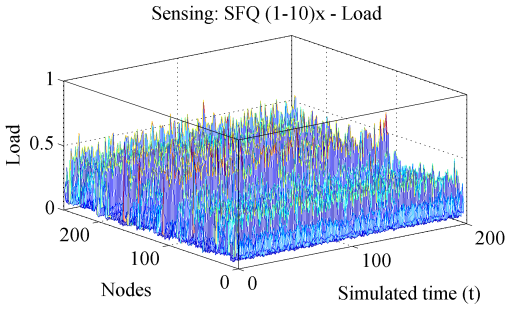

Figure 4.10: Sensing: SFQ, Load, $\sigma = 0.0908$, $L_{min} = 0.0096$, $L_{max} = 0.7416$

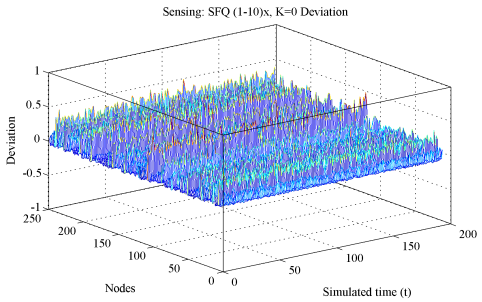

Figure 4.11: Sensing: SFQ, $k = 0$ estimates, $\sigma = 0.0905$, $L_{min} = -0.1120$, $L_{max} = 0.6051$

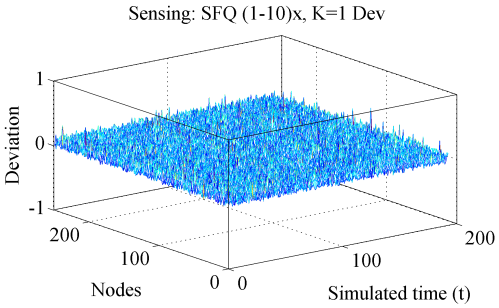

Figure 4.12: Sensing: SFQ, $k = 1$ estimates, $\sigma = 0.0544$, $L_{min} = -0.0995$, $L_{max} = 0.4597$

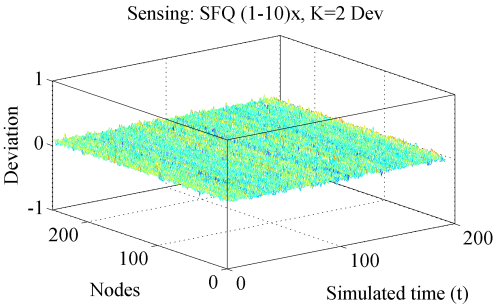

Figure 4.13: Sensing: SFQ, $k = 2$ estimates, $\sigma = 0.0276$, $L_{min} = -0.0785$, $L_{max} = 0.2005$

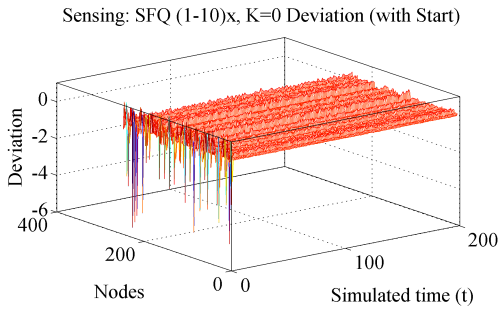

Figure 4.14: Sensing: SFQ,
$k = 0$ est (w/ Startup), $\sigma = 0.1257$,
$L_{min} = -5.4922$, $L_{max} = 1.2355$

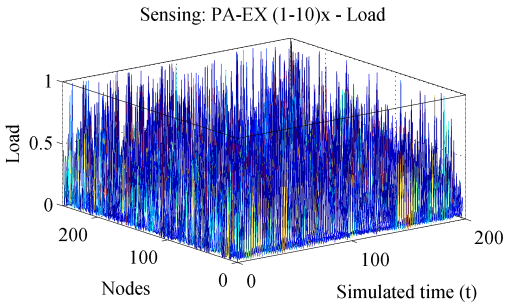

Figure 4.15: Sensing: PA-EX,
Load, $\sigma = 0.1348$,
$L_{min} = 0$, $L_{max} = 1$

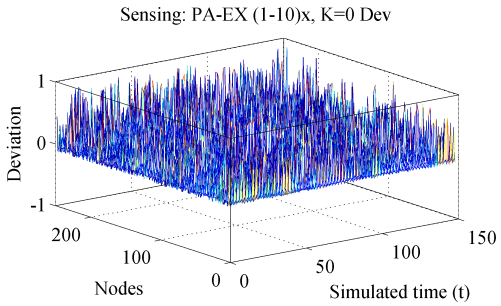

Figure 4.16: Sensing: PA-EX,
$k = 0$ estimates, $\sigma = 0.1347$,
$L_{min} = -0.1324$, $L_{max} = 0.9044$

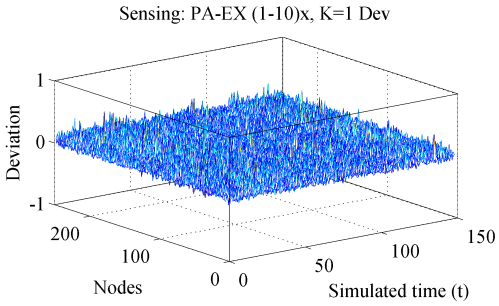

Figure 4.17: Sensing: PA-EX,
$k = 1$ estimates, $\sigma = 0.0723$,
$L_{min} = -0.1147$, $L_{max} = 0.6861$

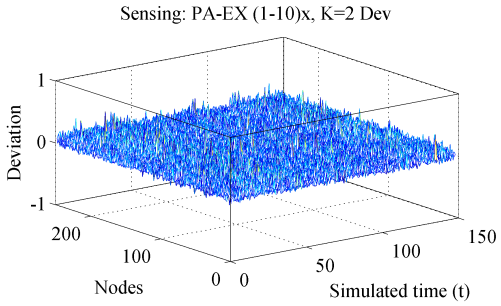

Figure 4.18: Sensing: PA-EX,
$k = 2$ estimates, $\sigma = 0.0451$,
$L_{min} = -0.1053$, $L_{max} = 0.2960$

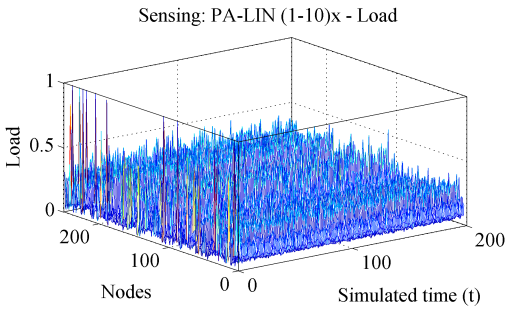

Figure 4.19: Sensing: PA-LIN, Load, $\sigma = 0.0749$, $L_{min} = 0$, $L_{max} = 1$

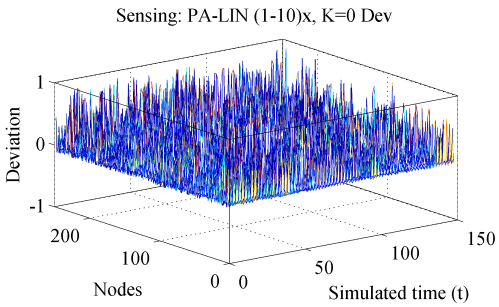

Figure 4.20: Sensing: PA-LIN, $k = 0$ estimates, $\sigma = 0.0484$, $L_{min} = -0.1044$, $L_{max} = 0.5138$

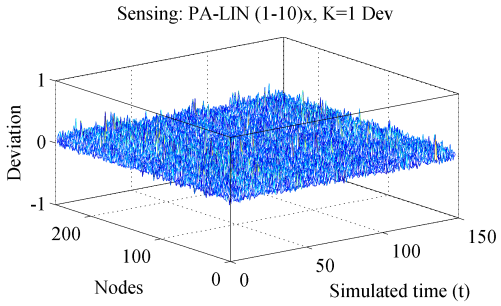

Figure 4.21: Sensing: PA-LIN, $k = 1$ estimates, $\sigma = 0.0517$, $L_{min} = -0.0946$, $L_{max} = 0.4946$

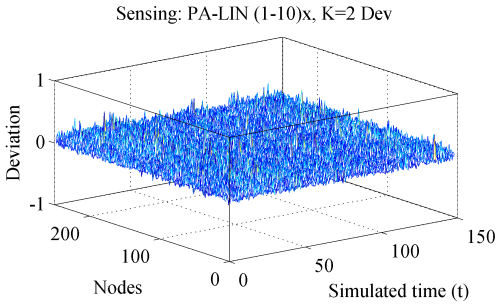

Figure 4.22: Sensing: PA-LIN, $k = 2$ estimates, $\sigma = 0.0275$, $L_{min} = -0.0517$, $L_{max} = 0.2374$

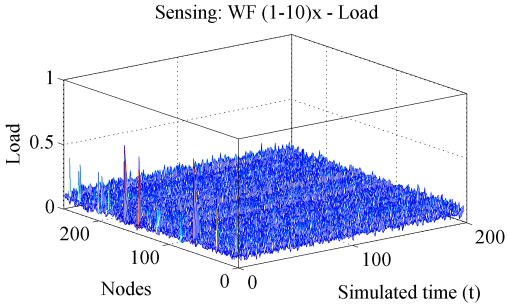

Figure 4.23: Sensing: WF, Load, $\sigma = 0.0300$, $L_{min} = 0$, $L_{max} = 0.7273$

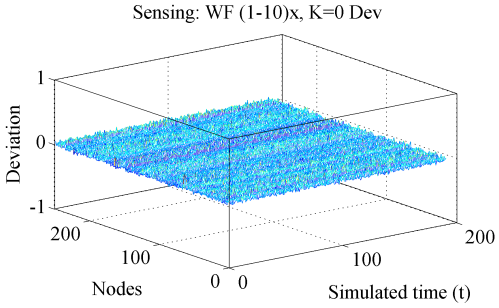

Figure 4.24: Sensing: WF, $k = 0$ estimates, $\sigma = 0.0289$, $L_{min} = -0.0844$, $L_{max} = 0.2305$

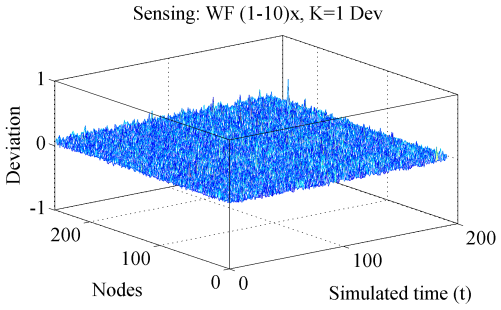

Figure 4.25: Sensing: WF, $k = 1$ estimates, $\sigma = 0.0478$, $L_{min} = -0.0676$, $L_{max} = 0.5217$

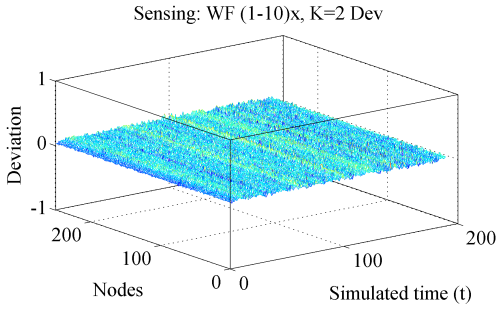

Figure 4.26: Sensing: WF, $k = 2$ estimates, $\sigma = 0.0256$, $L_{min} = -0.0318$, $L_{max} = 0.2292$

With these K-estimates, we are able to use them to sleep and wake nodes to achieve a pre-defined service level. Table 4.1 is a summary of results. As the number of iterations increase, standard deviation of error across the network is reduced.

Table 4.1: Sensing: Results summary

Strategies	**Load**, L_s	k_0, L_s	k_1, L_s	k_2, L_s	**Load**, σ	k_0, σ	k_1, σ	k_2, σ
SFQ	0.7320	0.7171	0.5592	0.2790	0.0908	0.0905	0.0544	0.0276
PA-EX	1	1.0368	0.8008	0.4013	0.1348	0.1347	0.0723	0.0451
PA-LIN	1	0.6182	0.5892	0.2891	0.0749	0.0484	0.0517	0.0275
WF	0.7273	0.3149	0.5893	0.2610	0.0300	0.0289	0.0478	0.0256

Load $L_s = L_{max} - L_{min}$.

We observe that at $k = 2$ (2 iterations), the spread and standard deviation of error is reduced.

4.5 Auto-Sleeping

With k-iterative sampling, we are able to estimate the global traffic. This can be used to determine an optimal number of nodes for a specific traffic load. This allows for a range of objective functions to be used. E.g. performance, efficiency or resiliency. A performance threshold can then be defined to instruct the nodes to sleep[8] / wake. In the following we demonstrate how we can achieve a pre-set network utilization by varying the number of active nodes.

4.5.1 Sleeping and waking nodes

Sleep

When a node determines that current load is beneath its threshold, it will send sleep commands to its peers. For this to occur reliably, error/deviations must be less than the effect of nodes going to sleep.

[8]Removing nodes from an active pool will help in concealing their existence to probing attacks. This prevents attackers from discovering the full performance potential of our networks.

The effect of a node going to sleep is defined as follows. $Q_N(t)$ = Change in network traffic. $X_i(t)$ = Traffic flow. $|H^k|$ = quantity of nodes in population k.

$$\Delta Q_N(t) = \frac{\sum_{j=1}^{H^k} X_i(t)}{|H^k| - |H^s|} - \frac{\sum_{i=1}^{H^k} X_i(t)}{|H^k|} \tag{4.15}$$

For a given deviation σ to be appreciable, change in $Q_N(t)$ must be larger.

$$\Delta Q_N(t) \geq \sigma_N(t) \tag{4.16}$$

$$\sigma_{h^k}(t) = \frac{\sum_{j=1}^{h^k} X_j^k(t)}{|h^k|} - \frac{\sum_{j=1}^{H^k} X_j^k(t)}{|H^k|} \tag{4.17}$$

Substitution yields.

$$\frac{\sum_{j=1}^{H^k} X_j(t)}{|H^k| - |H^s|} - \frac{\sum_{j=1}^{H^k} X_j(t)}{|H^k|} \geq \frac{\sum_{j=1}^{h^k} X_j^k(t)}{|h^k|} - \frac{\sum_{j=1}^{H^k} X_j^k(t)}{|H^k|} \tag{4.18}$$

$$\frac{\sum_{j=1}^{H^k} X_j(t)}{|H^k| - |H^s|} \geq \frac{\sum_{j=1}^{h^k} X_j^k(t)}{|h^k|} \tag{4.19}$$

As $h^k \to H^k$

$$\frac{\sum_{j=1}^{H^k} X_j(t)}{|H^k| - |H^s|} \geq \frac{\sum_{j=1}^{h^k} X_j^k(t)}{|H^k|} \tag{4.20}$$

$$\frac{1}{|H^k| - |H^s|} \geq \frac{1}{|H^k|}, \forall |H_s| > 0 \tag{4.21}$$

The above shows that to observe the effect of sleeping nodes, the quantity of nodes chosen to sleep must result in an increase in utilization that is above the error/deviation of the estimation of global traffic load. This must also be inversely true for waking up nodes. Different network configurations will yield varying performance for $\frac{\sum_{j=1}^{h^k} X_j^k(t)}{|h^k|}$. (These configurations were shown previously.)

In the following simulations, we specify only 1 node can sleep in 1 peer cluster.

$$h^s = \frac{H^k}{h_p} \tag{4.22}$$

The selection of node to sleep can be either (1) randomly chosen or (2) the poorest performing node[9]. In the following results, (2) is used. However, having every peer group sending a node to sleep might cause a large number of nodes to drop out altogether. To prevent that from happening, we introduce a random sleep countdown value. This value is equally distributed to minimize the chances of a large number of nodes going to sleep simultaneously. Hence the number of nodes going to sleep can be described as follows.

$$|h^s| = \frac{H^K}{h_p \times \mathcal{U}(a,b)} \tag{4.23}$$

In the following experiments, the sleep countdown value is between $a = 0$ and $b = 5$. $\mathcal{U}$ = uniformly distributed random function. When a node goes to sleep, it goes into the sleep list.

Wake

When the network load increases, more nodes are required to maintain a pre-set performance. When this occurs, nodes start to wake neighbouring node.

It first starts by turning off any of its peer's countdown. This prevents more nodes from going to sleep. It then wakes up a node from its sleep node list. In our simulations and subsequent prototype, the node chosen to wake follows FIFO. FIFO favours older sleeping nodes. This reduces the occurrence of a recently slept node from waking up too quickly. Nodes that are woken too quickly may not have completed updating its peers on its new sleep status. Choosing older nodes also allows us to verify their connectivity. If they are not contactable, they are removed from the sleep list. This process removes dead nodes from the sleep list.

4.5.2 Simulations

To demonstrate sleep/wake mechanism, we have the following 2 sets of simulations. (1) low-frequency and (2) high-frequency network changes. Similar to previous simulations, these are written in Java and make use of token bucket algorithms.

[9]The poorest performing node is one which reports the lowest throughput during periodic updates.

Utilization is calculated by the expended tokens over the maximum tokens. Each simulation consists of 1000 nodes. Each layer (1-4) have 250 nodes. The number of client requests can vary from 4000 to 16000 per second.

3 epochs

In the first simulation, 3 epochs or varying loads are defined. Figure 4.27 and 4.28 (plane view) shows the network without sleep or waking any nodes.

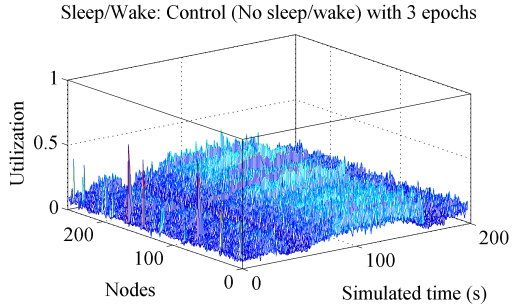

Figure 4.27: Sleep/Wake: 4000-16000 clients, 3 epochs, (Control) no sleep/wake

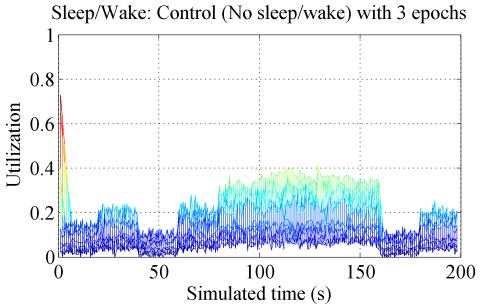

Figure 4.28: Sleep/Wake: 4000-16000 clients, 3 epochs, (Control) no sleep/wake

Figure 4.28 shows the same graph projected on simulated time and utilization axes. We can observe the varying load experienced by the network over the course of the simulation. Figure 4.29 shows the same network but with sleep/wake capabilities. As the network starts, the k-estimate shows that the network utilization is low. The network starts to sleep nodes. This sleep continues until the increase in traffic causes the nodes to stop sleeping and start waking instead.

As the network operates, nodes are directed to sleep. At approximately t=40, the rate of increase slows down as traffic load increases. When the load decreases, more nodes go to sleep. At the end of simulation, only the required number of nodes are left to be operational. Figures 4.30 and Figure 4.31 are plane projections of Figure 4.29. Figure 4.30 shows network utilization vs time. Figure 4.31 shows the nodes vs utilization to illustrate the nodes left operational at the end indicated by their almost 100% load.

The Figures 4.32, 4.33 and 4.34 show the respective Zone 2 results. We can observe that utilization with sleep/wake are similar between zones. Figure 4.35

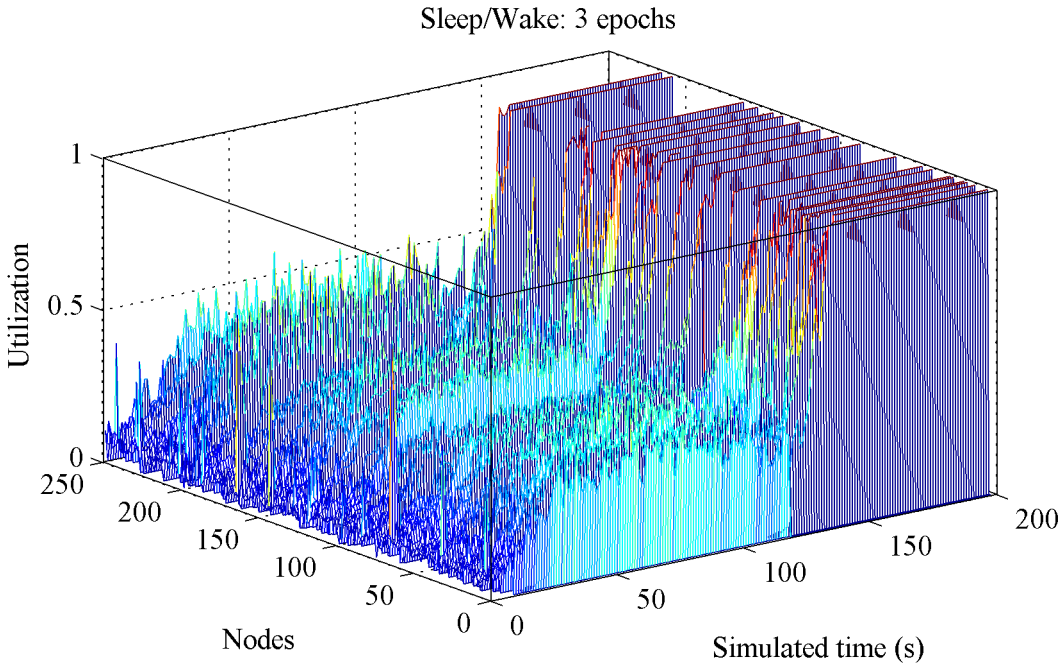

Figure 4.29: Sleep/Wake: 4000-16000 clients, 3 epochs, (Layer 1)

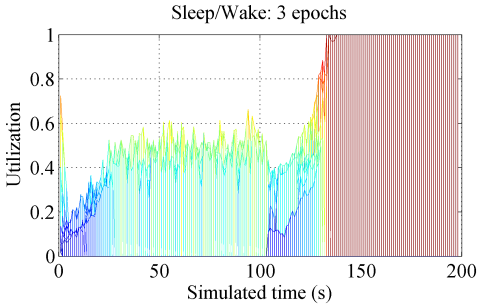

Figure 4.30: Sleep/Wake: 4000-16000 clients, 3 epochs, Utilization vs Time (Layer 1)

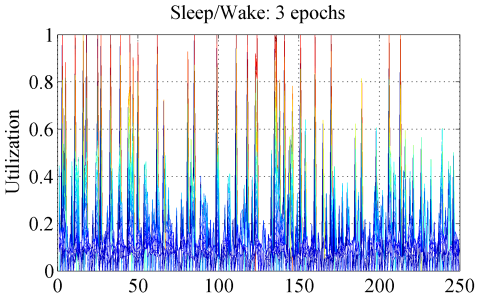

Figure 4.31: Sleep/Wake: 4000-16000 clients, 3 epochs, Utilization vs Node (Layer 1)

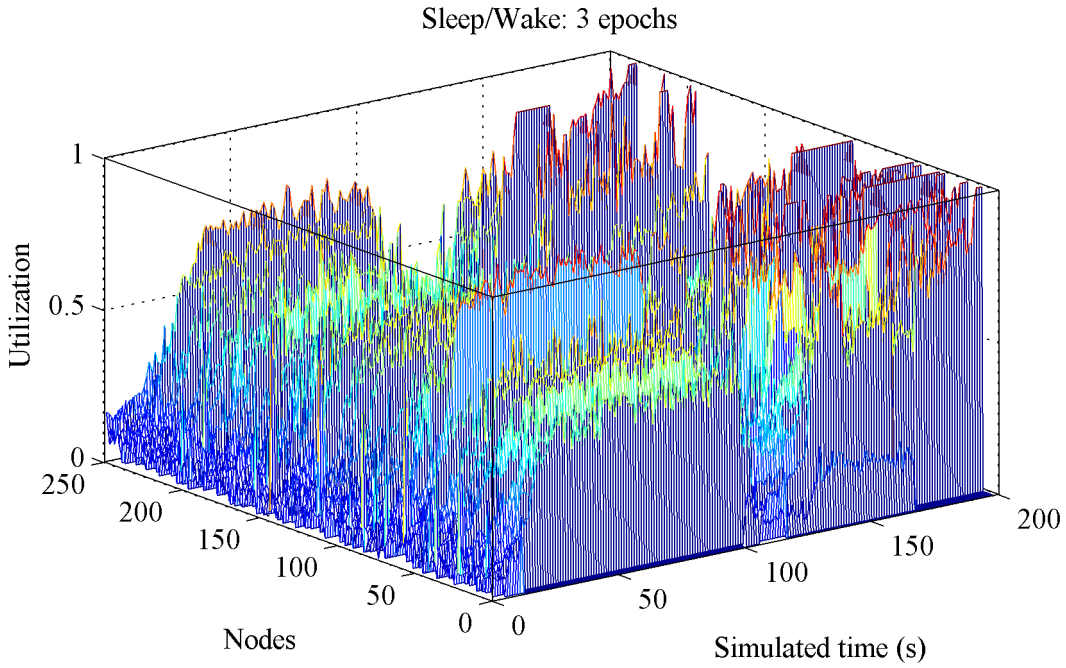

Figure 4.32: Sleep/Wake: 4000-16000 clients, 3 epochs, (Layer 2)

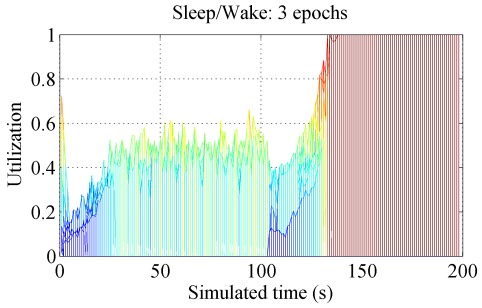

Figure 4.33: Sleep/Wake: 4000-16000 clients, 3 epochs, Utilization vs Time (Layer 2)

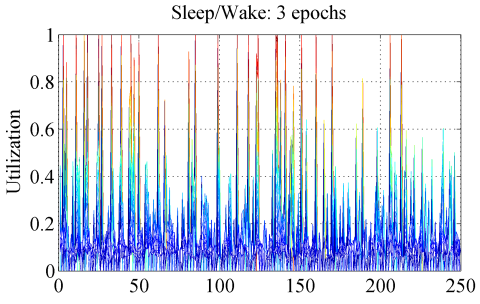

Figure 4.34: Sleep/Wake: 4000-16000 clients, 3 epochs, Utilization vs Node (Layer 2)

shows the average zone traffic. The drop at approximately $t = 140$ is caused by a substantial decrease in external requests. We can observe that they are fairly similar and the network itself is maintaining our specified service level.

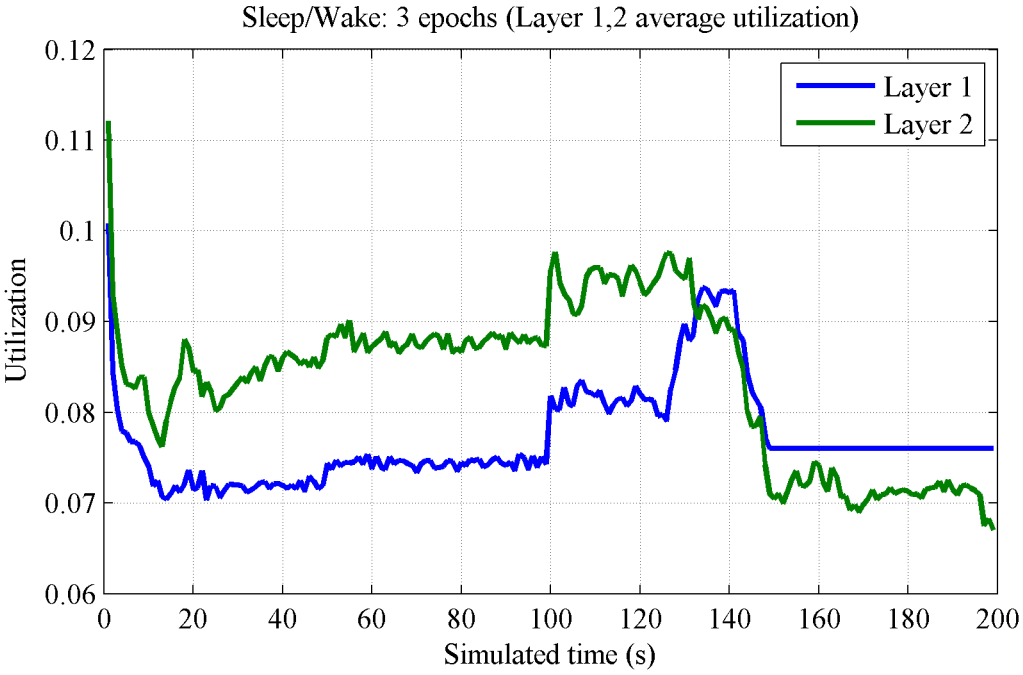

Figure 4.35: Sleep/Wake: 4000-16000 clients, 3 epochs, Layer 1,2 average utilization

9 epochs

In the following set of experiments we increase the number of epochs to 9. Figures 4.36 and 4.37 show the network without sleep/wake. This shows how the traffic load varies against time. We can observe that in these runs, traffic load continues to increase throughout the simulation.

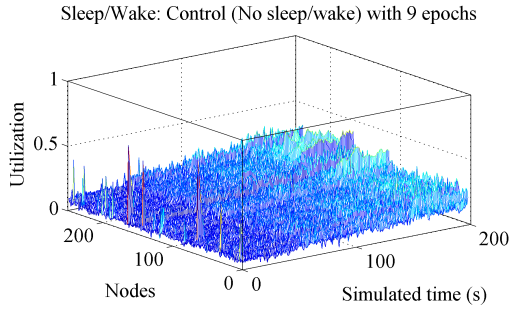

Figure 4.36: Sleep/Wake: 4000-16000 clients, 9 epochs, (Control) no sleep/wake

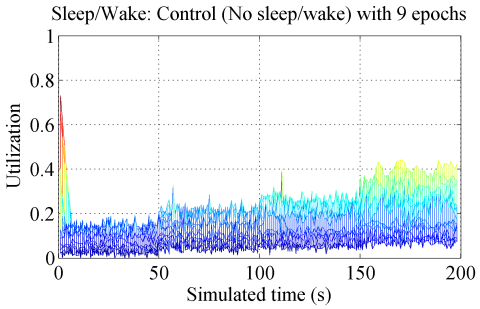

Figure 4.37: Sleep/Wake: 4000-16000 clients, 9 epochs, (Control) no sleep/wake

Figures 4.38, 4.39 and 4.40 shows sleep/wake capabilities. Figure 4.38 shows the utilization of the network over time. Figures 4.39 and 4.40 are plane projections. We can observe that in the beginning the network estimates that traffic load is low and starts to sleep nodes. However, incoming traffic is increasing, hence at $t = 45s$ s spike in traffic occurs when many nodes are asleep. Subsequently, the network determines the minimum number of nodes required.

From the above results, we show that we can sleep/wake nodes to achieve a target service level as shown in Figure 4.35 and 4.41. In these simulations, the layer average utilization remains within the threshold while nodes go into sleep and wake states.

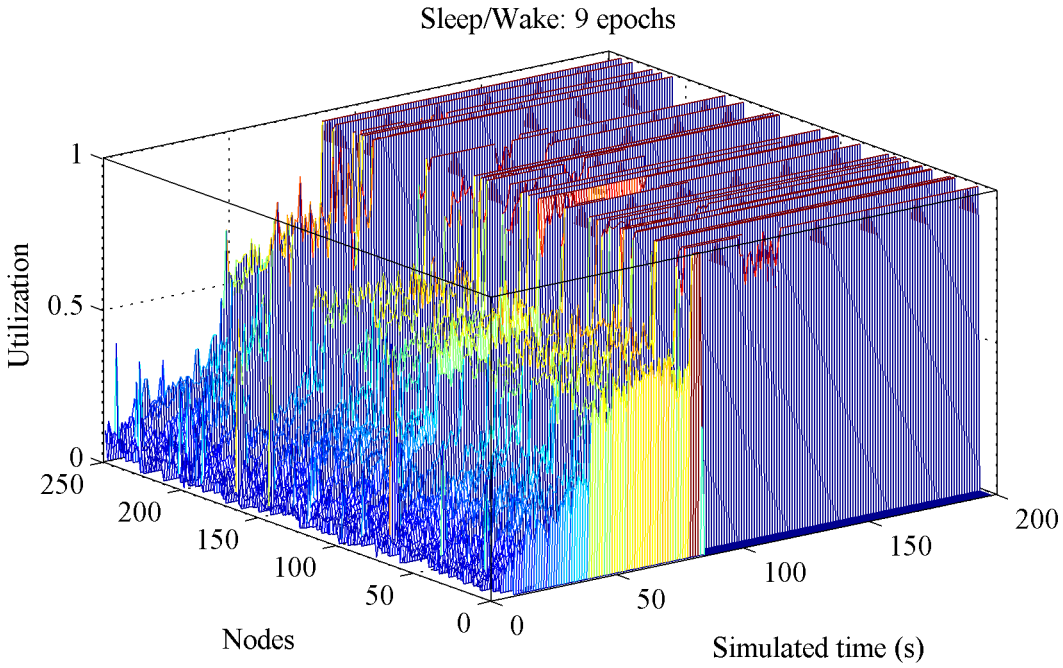

Figure 4.38: Sleep/Wake: 4000-16000 clients, 3 epochs, (Layer 2)

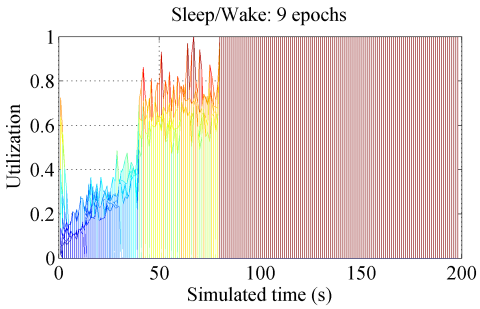

Figure 4.39: Sleep/Wake: 4000-16000 clients, 9 epochs, Utilization vs Time (Layer 2)

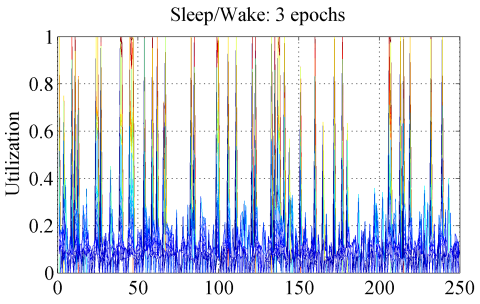

Figure 4.40: Sleep/Wake: 4000-16000 clients, 9 epochs, Utilization vs Node (Layer 2)

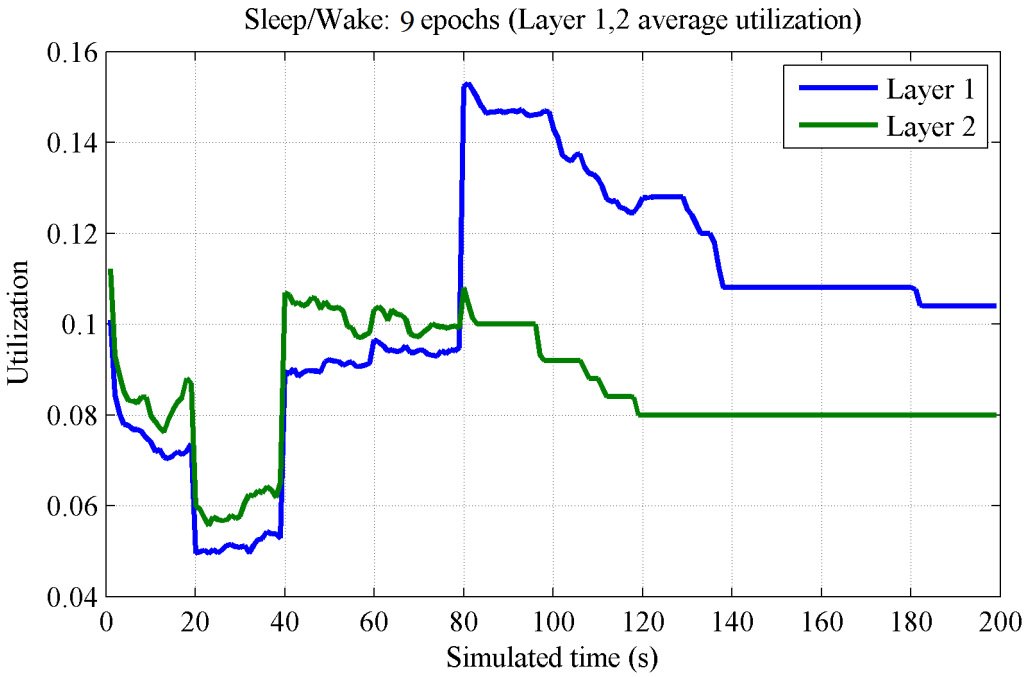

Figure 4.41: Zone 1,2 Avg, 4000-16000 clients, 9 epochs, sleep/wake

4.6 Auto-Resistance

Distributed networks such as the above pose challenges in detecting intermittent or low-rate attacks. This is primarily due to (1) lack of oversight authority and (2) load balancing effects. Large scale deployment of sensors and scrubbers in large unstructured networks poses efficiency and effectiveness challenges.

Reputation-based approaches such as EigenTrust [51, 98] does not allocate node roles in the network. Using CUSUM [133] and space similarity provides a decentralized approach to DDoS detection but requires large amount of computation to have good results. This is undesirable as it further strains the network. Back propagation rate limiting such as Pushback did not consider overlay networks.

We propose a mechanism to self-optimize the allocation of detectors and traffic scrubbers. Using an inexpensive trigger, we recruit responsible upstream nodes to detect and scrub traffic relayed through them. This provides on-demand computational and network capability to manage the on-going attack.

4.6.1 Network funnelling

Using fast-flux DNS, only IP addresses of entry nodes are publicized. They are revealed to external participants or clients through the Fast Flux DNS. As illustrated previously, malicious traffic would enter the network through entry nodes. This is illustrated in Figure 4.42.

For a given server, the maximum service rate is also known. We allocate exit nodes in respect to the maximum capacity of the servers. However, the volume of traffic experienced by entry nodes are typically higher, as they include higher proportions of illegitimate flows.

Therefore, the number of entry nodes exceeds that of exit nodes. The number of relay nodes is upper and lower bounded by $N_{\text{entry}} \geq N_{\text{relay}} \geq N_{\text{exit}}$.

4.6.2 Resisting abnormal traffic flows

When the network experiences normal traffic, only the deepest nodes are performing network sampling and scrubbing. The closer the detector is to their victim,

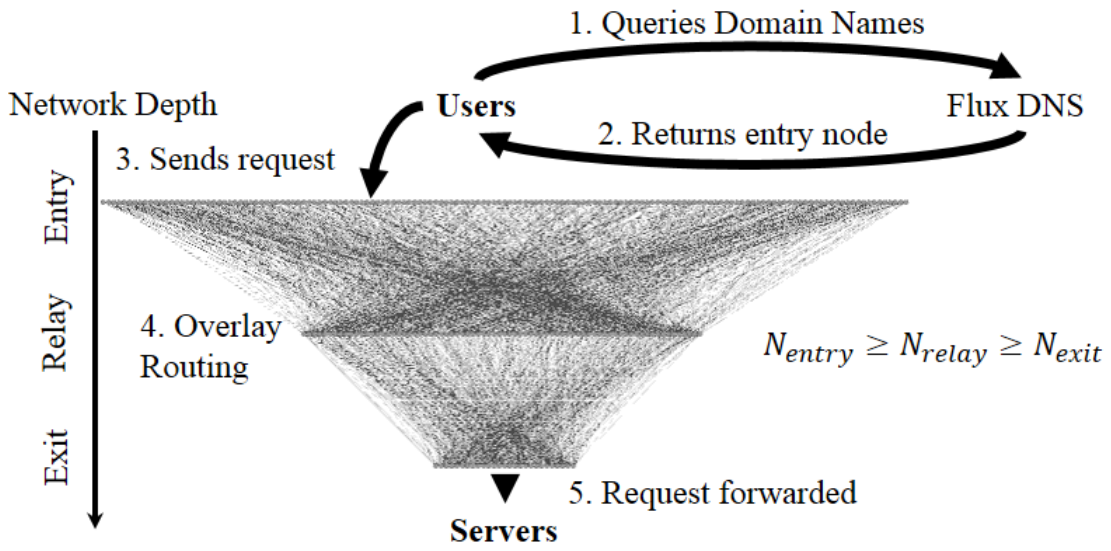

Figure 4.42: Resistance: Example overlay network (256 nodes)

the higher is the level of aggregation. This allows greater sensitivity[10]. As the number of nodes at the deepest end (exit nodes) is the smallest, the amount of resources used is minimal. However, this also limits their ability to filter and mitigate abnormal traffic flows[11].

When the amount of traffic increases, these nodes force the responsible upstream nodes to perform similar sensing and scrubbing. The thresholds used are a function of the number of peers, network distance and local capacity. This ultimately increases the nodes used to analyse and scrub traffic. The system as a whole can determine the optimal number of detectors and scrubbers at any given traffic flows.

The above effect increases detector and scrubbing intensity in response to the scale of the incoming abnormal traffic as shown in Equation (4.24). As the abnormal traffic reduces, detectors and scrubbers will reduce accordingly.

$$\sum_{z=1}^{z=k}\left(\sum_{y=1}^{y=N} n_{(z,y)}\right) \propto \sum_{z=1}^{z=k}\left(\sum_{y=1}^{y=N} X_{(z,y)}(t)\right) \tag{4.24}$$

[10]As we track source-destination pairs, an deliberate attack from a particular source does not become evident until significant aggregated traffic is observed.

[11]Nodes sample their respective workloads to determine if the current flow is abnormal. However, they do not perform analysis or scrubbing until later.

z, current layers/depth or hops.

k, maximum number of layers/hops.

y, nodes in a particular zone (E.g. entry, relay, exit).

N, maximum number of nodes in a given layer.

n, quantity of nodes that are actively detecting and scrubbing.

X, traffic observed at a given node (E.g. packets/sec).

4.6.3 Simulations

We set-up a basic overlay network of up to 3 layers. Each layer represents entry (4,915), relay (2,456), exit (820) nodes and 65,536 - 262,144 client performing requests every second. The simulation is written in Java and makes use of a token bucket algorithm to measure utilization.

An attack is simulated by increasing the number of requests/clients at $t = 500s$. Figure 4.43 shows incoming traffic at entry nodes (Layer 1). Figure 4.44 shows overlay traffic without sensing or filtering at exit nodes (Layer 3). This shows that without any intervention, most of the incoming traffic passes through to the exit layer.

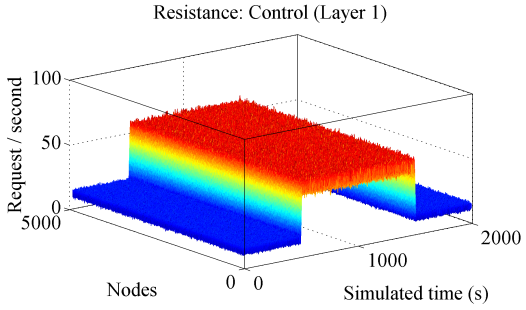

Figure 4.43: Resistance : Request rate at Layer 1 Control (no scrubbing)

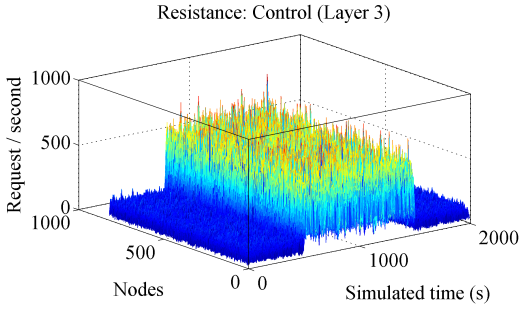

Figure 4.44: Resistance : Request rate at Layer 3 Control (no scrubbing)

It is then shown in Figure 4.45, when the traffic flow increases by a factor of 4 to simulate a flash crowd, a large resistive response shown in Figure 4.46 from the same network is observed. The aggressiveness of this increased sampling and scrubbing corresponds to the rate of increase in traffic. The blue graph indicates filtering intensity of the entry (Layer 1) nodes and green refers to the relay (Layer

2) nodes. Layer 2 receives plenty of triggers from Layer 3 as shown by the high filtering intensity. As Layer 2 is larger, there is less need to trigger Layer 1's filter. Hence, Layer 1's filter intensity is lower. This demonstrates how triggering upstream nodes can minimize impact at the exit layers.

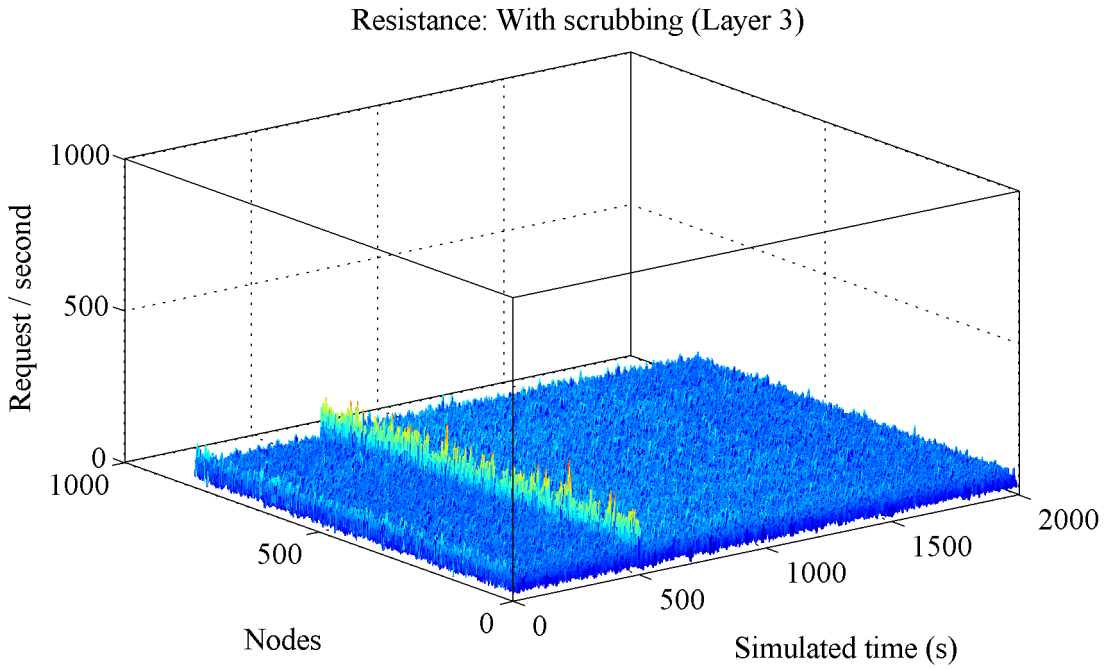

Figure 4.45: Resistance : Request rate at Layer 3, with scrubbing

Various detection and filtering mechanisms can be used in this system. In our approach, we track the source-destination pair and apply a corresponding threshold. We assume that legitimate users will not exceed a pre-set threshold. When this threshold is exceeded, the triggering mechanism comes into effect. More sophisticated traffic analyses can be used in place to improve detection success. When malicious traffic is detected, it is simply dropped and not forwarded through our networks.

In summary, we described a trigger mechanism to auto-determine optimal quantity of traffic scrubbers. We have shown that as traffic increases beyond a threshold, we propagate triggers to upstream nodes. This does not require a centralized detector and scrubber but is still able to provide a balance between sensitivity (to bad flows) and capability (to scrub).

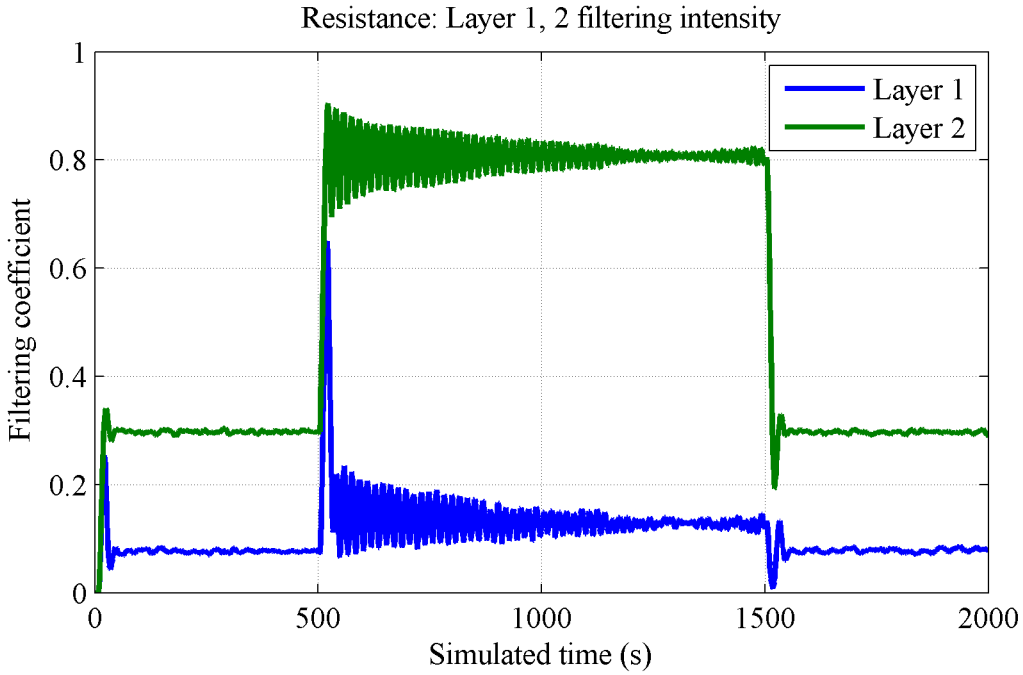

Figure 4.46: Resistance : Filtering intensity at Layer 1 and 2

4.7 Summary

In this chapter, we have discussed various self-configuration, healing, optimization and protection features. Fast flux session binding provides an iterative approach to build and heal networks during outrages while directing clients to gateways. This mechanism also masks location of end point servers to avoid those becoming targets of DDoS attacks. Iterative sampling allows us to estimate global traffic efficiently.

Multiple selection algorithms with varying performance are used to show the feasibility of the proposed iterative sampling technique. This is shown in Table 4.2. We can observe that the deviation between estimated global traffic is minimized with increasing iterative k-estimates.

This forms a control loop to optimize various network parameters. This is then extended to 'sleeping' and 'waking' nodes depending on predefined thresholds. We illustrated how sleep/wake is managed with 2 sets of simulations. The table of figures shown is in Table 4.3. We can observe from the figures, our network was able to vary the number of active nodes to maintain quality of service.

Table 4.2: Sensing: Table of figures and results

Route selection	**Load**	k_0 (Dev)	k_1 (Dev)	k_2 (Dev)
SFQ	Fig 4.10, 0.7320	Fig 4.11, 0.7171	Fig 4.12, 0.5592	Fig 4.13, 0.2790
PA-EX	Fig 4.15, 1	Fig 4.16, 1.0368	Fig 4.17, 0.8008	Fig 4.18, 0.4013
PA-LIN	Fig 4.19, 1	Fig 4.20, 0.6182	Fig 4.21, 0.5892	Fig 4.22, 0.2891
WF	Fig 4.23 0.7273	Fig 4.24, 0.3149	Fig 4.25, 0.5893	Fig 4.26, 0.2610

Table 4.3: Sleep/Wake: Table of figures

	3 epochs	**9 epochs**
no sleep	Fig 4.27	Fig 4.36
no sleep (plane view - util vs time)	Fig 4.28	Fig 4.37
with sleep	Fig 4.29 (zone 1), Fig 4.32 (zone 2)	Fig 4.38
with sleep (plane view - util vs time)	Fig 4.30 (zone 1), Fig 4.33 (zone 2)	Fig 4.39
with sleep (plane view - util vs nodes)	Fig 4.31 (zone 1), Fig 4.34 (zone 2)	Fig 4.40
with sleep, average zone util	Fig 4.35	Fig 4.41

Finally, we demonstrate self-protection by proposing an upstream trigger for nodes to scrub traffic when excessive requests are made. Figure 4.43 and 4.44 are control simulations. Figure 4.45 shows the effects of our scrubbing technique.

Chapter 5

Swarm Networking

5.1 Overview

In the previous Chapter 4, we described various components that makes up the swarm network. They address interface and control mechanisms that is used to build and maintain network structure. The following chapter describes how traffic is managed in a distributed manner during network operation. Figure 5.1 shows shaded components involved in distributed traffic management.

Section 5.2 describes a traffic model of our network. Section 5.3 describes how allocation and queuing are performed by Linux systems. Section 5.4 describes how variations in the network affects the fair usage of nodes. Tokens are then extended to measure the utilization and used for route selection in Section 5.5. We then describe our proposed Water flow algorithm in Section 5.6.

5.2 Network and Traffic models

In this section, we describe our representation of traffic flows in a network. We show that a user equilibrium state driven by non-cooperative nature of individual nodes can approximate an optimal solution to the routing problem. This user equilibrium state is first described as a Wardrop equilibria [119]. This forms the basis of implementing a distributed mechanism such as a swarm algorithm that mimics the selfish nature described in the following sections.

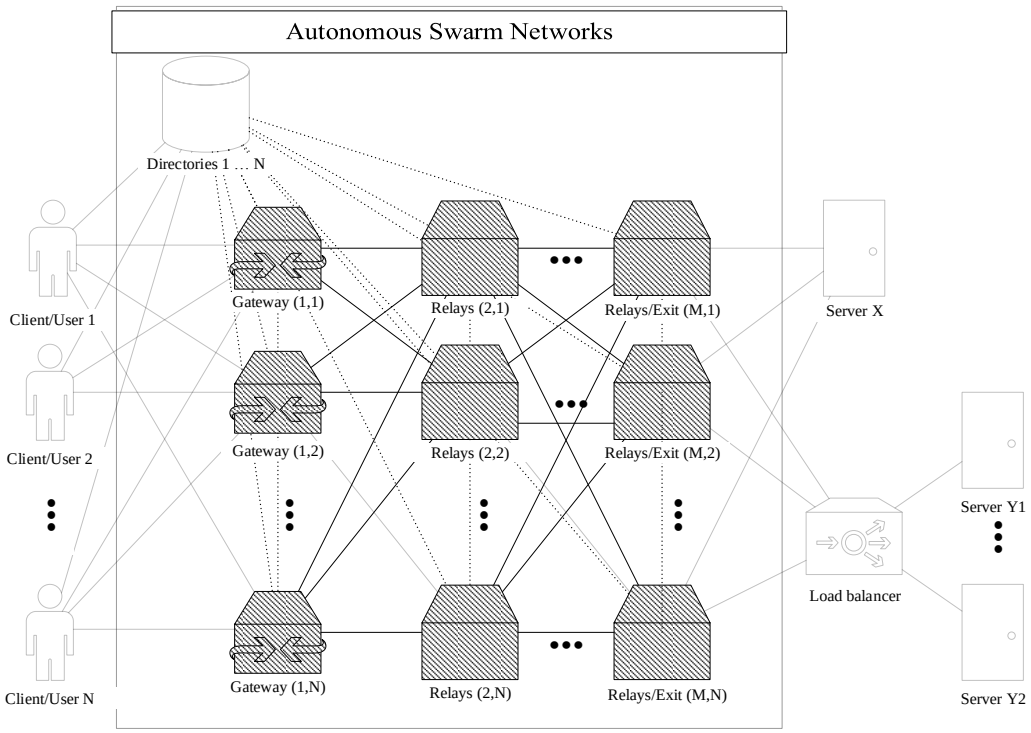

Figure 5.1: Components involved in distributed traffic management

5.2.1 Network structure and dynamics

To begin, we use a simple graph in Figure 5.2 to illustrate how nodes (hosts) can be connected in a network.

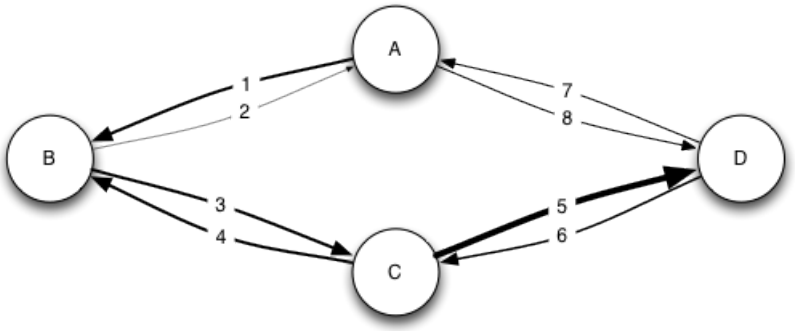

Figure 5.2: Nodes and Links

Figure 5.2 shows 4 nodes, connected to one another directly or indirectly through a set of 8 links. A minimum of 4 nodes is required to illustrate the alternative routes to adjacent nodes that are bidirectionally linked to one another. We can imagine each node representing hosts and the links represent uplinks or downlinks between different nodes. The thickness of the links illustrates the asymmetric property of uplinks and downlinks[1]. The arrow indicates the direction of the links[2].

Let L be the set of directed links and let R be the set of possible routes. We describe the relationship between links and routes with a matrix. The matrix is defined as follows. Set $A_{lr} = 1$ if link l lies on route r, and set $A_{lr} = 0$ otherwise. This defines a matrix $A = A_{lr}, l \in L, r \in R$ called the link-route incidence matrix. Each column of the matrix corresponds to one of the routes r, and each row to one of the links l of the network. The column for route r is composed of 0s and 1s[3]. Thus we can construct the following incidence matrix[4] shown in Figure 5.3.

[1]Asymmetric uplinks and downlinks are commonplace in the Internet. Servers usually have larger uplink bandwidths to serve content, while clients have larger downlink bandwidths allocated to receive it

[2]Links/Cables are bi-directional, but the above illustrates the different bandwidth allocated to them

[3]1s in the row for link j indicates which route passes through that link.

[4]Note that for the following incidence matrix, suffix with 1s are routes in an anti-clockwise direction, and 2s are in a clockwise direction

Routes/Links	AB1	AB2	AC1	AC2	AD1	AD2	BA1	BA2	BC1	BC2	BD1	BD2
1	1	0	1	0	1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	0	1	0	1	0
3	0	0	1	0	1	0	0	1	0	1	0	1
4	0	1	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	1	0	0	1	0	0	0	1
6	0	1	0	1	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0	0
8	0	1	0	1	0	1	0	0	1	0	1	0
Routes/Links	CA1	CA2	CB1	CB2	CD1	CD2	DA1	DA2	DB1	DB2	DC1	DC2
1	0	0	0	0	0	0	0	0	1	0	1	0
2	1	0	0	0	0	1	0	1	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	1	0
4	1	0	0	1	0	1	0	1	0	1	0	0
5	0	1	1	0	1	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	1	0	1	0	1
7	0	1	0	0	0	0	1	0	1	0	1	0
8	0	0	0	0	0	1	0	0	0	0	0	0

Figure 5.3: Link-route incidence matrix, A

Given the above example, AB1 refers to one of two possible routes linking Node A to Node B. AB1 refers to the route using Link 1. With the above, we attempt to quantify the volume of traffic along a particular route or link. Let $x_r(t)$ be the flow on route r, defined as the number of bits that can travel along that route. We can list the flows along all the routes as a sequence of numbers $x(t) = (x_r(t), r \in R)$, and we can interpret this sequence as a vector. We can then quantify the total flow through a particular link. In general, since $A_{lr} = 1$ when a route r passes through the link l and $A_{lr} = 0$ when it does not, the total flow through link l coming from all routes can be described as (5.1).

$$y_l(t) = \sum_{r \in R} A_{lr} x_r(t), l \in L. \tag{5.1}$$

As we have previously described, the set of numbers $y_l, l \in L$ can be viewed as a vector. Hence the above equation can be represented in the following matrix form in (5.2).

$$y(t) = Ax(t) \tag{5.2}$$

We can expect that the level of congestion at a particular link to correlate with the total flow through that link. This level of congestion also influences the time

required to traverse through the link[5]. The time taken for data to traverse a particular link can be described as delay. Network delay is described as a non-linear strictly increasing function. When a link is underutilized (little or no flow), the delay experienced by a packet moving between nodes is just the processing, transmission and propagation delay. As the link becomes heavily utilized, congestion delay will become larger. We describe this relationship with $D\big(y(t)\big)$, where at larger flows $y(t)$, it is larger due to congestion effects.

Let $D_l\big(y_l(t)\big)$ be the delay[6] along a particular link l when the flow through the link is $y_l(t)$. The delay varies between different links. We define D_l with the subscript l to highlight the difference in delays.

5.2.2 Route selection

Any given two nodes in a network will have a variety of possible routes capable of linking them. Example from Figure 5.2 shows that flows originating from source A to source B can make use of either links AB1 or AB2. We now need another matrix to describe the relationship between source-destination pairs and their routes.

Let s denote a source-destination pair, and let S denote the set of all source-destination pair. Let r denote a route, and let R denote the set of all routes. Let $B_{sr} = 1$ if s can be served by route r, and let $B_{sr} = 0$ otherwise. Hence we have the matrix $B = (B_{sr}, s \in S, r \in R)$;. The matrix of routes and source destination is shown in Figure 5.4.

We can quantify the total flow between a source-destination pair s with $x(t)$ and matrix B. We define f_{sr} as the total flow of traffic added up over all of the routes serving the source-destination pair sr. Hence we can conclude the following relationship in (5.3)

$$f_{sr}(t) = \sum_{r \in R} B_{(sr)} x_r(t), s \in S \tag{5.3}$$

Thus the vector $f(t) = f_s(t), s \in S$ of the source-destination flows can be

[5]Intuitively, when a link becomes saturated, it takes a longer period of time to traverse this. This is mainly due to congestion control mechanisms that reduces the rate of flow between source-destination pairs).

[6]$x(t)$ refers to the total flow, y refers to the flow of a link

SD pairs/Routes	AB1	AB2	AC1	AC2	AD1	AD2	BA1	BA2	BC1	BC2	BD1	BD2
ab	1	1	0	0	0	0	0	0	0	0	0	0
ac	0	0	1	1	0	0	0	0	0	0	0	0
ad	0	0	0	0	1	1	0	0	0	0	0	0
ba	0	0	0	0	0	0	1	1	0	0	0	0
bc	0	0	0	0	0	0	0	0	1	1	0	0
bd	0	0	0	0	0	0	0	0	0	0	1	1
ca	0	0	0	0	0	0	0	0	0	0	0	0
cb	0	0	0	0	0	0	0	0	0	0	0	0
cd	0	0	0	0	0	0	0	0	0	0	0	0
da	0	0	0	0	0	0	0	0	0	0	0	0
db	0	0	0	0	0	0	0	0	0	0	0	0
dc	0	0	0	0	0	0	0	0	0	0	0	0
SD pairs/Routes	CA1	CA2	CB1	CB2	CD1	CD2	DA1	DA2	DB1	DB2	DC1	DC2
ab	0	0	0	0	0	0	0	0	0	0	0	0
ac	0	0	0	0	0	0	0	0	0	0	0	0
ad	0	0	0	0	0	0	0	0	0	0	0	0
ba	0	0	0	0	0	0	0	0	0	0	0	0
bc	0	0	0	0	0	0	0	0	0	0	0	0
bd	0	0	0	0	0	0	0	0	0	0	0	0
ca	1	1	0	0	0	0	0	0	0	0	0	0
cb	0	0	1	1	0	0	0	0	0	0	0	0
cd	0	0	0	0	1	1	0	0	0	0	0	0
da	0	0	0	0	0	0	1	1	0	0	0	0
db	0	0	0	0	0	0	0	0	1	1	0	0
dc	0	0	0	0	0	0	0	0	0	0	1	1

Figure 5.4: Matrix of Routes and Source-Destination pairs, B

expressed in the following matrix form.

$$f(t) = Bx(t) \tag{5.4}$$

5.2.3 Wardrop Equilibria

With the above representation of the network configuration and traffic flows, we have to show that we are able to determine an effective solution to the routing problem. We propose finding the user-equilibrium state first proposed by Wardrop [119].

We address how traffic flows between various sources and destinations distribute themselves over the links of the network. Each node will attempt to use whichever route is quickest. However, this choice in itself may make other routes quicker or slower and hence affecting the choices of other hosts. However, when nodes are unable to find alternate quicker routes, the nodes will have no incentive to change routes[7].

[7]inferred from Wardrop's 2nd principle.

Let us describe the time taken for data to flow through route r. The columns labelled r of the matrix indicates which links l are on route r. By summing up the delays on each of these links, we can determine the time taken to travel along route r with the following expression (5.5).

$$t = \sum_{l \in L} D_l(y_l(t)) A_{lr} \tag{5.5}$$

As described earlier, a node will not choose an alternate path, if it is not quicker. Hence, for a node to be contented with it's current choice, the following inequality must be satisfied.

$$\sum_{l \in L} D_l\big(y_l(t)\big) A_{lr} \leq \sum_{l \in L} D_l\big(y_l(t)\big) A_{lr'} \tag{5.6}$$

Where r' route refers to any other route that is not r. We then can define a Wardrop equilibrium to be a vector $e = (e_r, r \in R)$ of non-negative numbers serving all routes of the same source-destination pair [12].

$$e_r > 0 \Rightarrow \sum_{l \in L} D_l\big(y_l(t)\big) A_{lr} \leq \sum_{l \in L} D_l\big(y_l(t)\big) A_{lr'}, \text{ where } y(t) = Ax(t) \tag{5.7}$$

Equation (5.7) describes the self-interest nature of nodes. This is one of the characteristics of a Wardrop equilibrium. Wardrop states that if a route r is actively used, then it achieves the minimum delay over all routes serving its source-destination pair $s(r)$. Hence we need to find vector e that satisfies all the above implications.

Kelly [54] and Beckmann et al [11] used the following optimization problem (5.8) to show that if (x, y) has a solution, vector x will correspond to a Wardrop equilibrium.

$$\begin{gathered} \text{Minimize } \textstyle\sum_{j \in J} \int_0^{y_j(t)} D_j(t)(u)du \text{ over } x(t) \geq 0, y(t), \\ \text{subjected to } Bx(t) = f(t), Ax(t) = y(t). \end{gathered} \tag{5.8}$$

This optimization problem has the following premises.

1. The flows along each route are non-negative. Hence $x(t) \geq 0$

2. $Bx(t) = f(t)$ and $Ax(t) = y(t)$ are the vectors defined earlier.

3. The constraints $Bx(t) = f(t)$ and $Ax(t) = y(t)$ enforces the rules that allows the source-destination flow $f(t)$ and the link flows $y(t)$ to be calculated from route flows $x(t)$ using matrices A and B.

Kelly then consider that at any point, the source-destination flow $f(t)$ is constant, but is distributed over a number of routes. Hence given a choice of routes, the route flows $x(t)$ and $y(t)$ are required. A solution to the optimization problem (Eqn 5.7) is defined by the following function.

$$L(x(t), y(t), \lambda, \mu) = \sum_0^{j \in J} \int_0^{y_j(t)} D_j(u)du + \lambda \cdot (f(t) - Bx(t)) - \mu \cdot (y(t) - Ax(t)),$$
$$\text{Where } \lambda = (\lambda_s, s \in S), \mu = (\mu_j, j \in J) \text{ are vectors of Lagrange multipliers} \tag{5.9}$$

Kelly chose appropriate Lagrange multipliers to minimize the function L over $x(t)$ and $y(t)$ to find a solution under the constraints $Bx(t) = f(t)$ and $Ax(t) = y(t)$ in the minimization of L. To find the minimal, they differentiated the following.

$$\begin{array}{ll} \text{First:} & \frac{\partial L}{\partial y_j(t)} = D_j(y_{j(t)}) - \mu_j \\ \text{Second:} & \frac{\partial L}{\partial x_r(t)} = -\lambda_{s(r)} + \sum_{j \in J} \mu_j A_{jr} \end{array} \tag{5.10}$$

The form of matrix B causes the derivative with respect to $x_r(t)$ to pick out exactly one component of λ, namely $\lambda_{s(r)}$, and the form of the matrix A causes the derivative to pick out just components of μ that correspond to links on route r.

This allows a minimum of L (under $x(t) \geq 0$ and for all $y(t)$) to occur in the following.

$$\mu_l = D_l(y_l(t)) \text{ and}$$
$$\lambda_{s(r)} \begin{cases} = \sum_{l \in L} \mu_l A_{lr} & \text{if } x_r > 0 \\ \leq \sum_{l \in L} \mu_l A_{lr} & \text{if } x_r = 0. \end{cases} \tag{5.11}$$

The equality condition for $\lambda_{s(r)}$ can be interpreted as follows.

If $x_r(t) > 0$ then small variations up or down in $x_r(t)$ should not decrease the function $L(x, y; \lambda, \mu)$, and hence we deduce that the partial derivative with respect

to $x_r(t)$ must be 0. But if $x_r(t) = 0$ then we can only vary $x_r(t)$ upwards[8], and so we can deduce that the partial derivative with respect to $x_r(t)$ is non-negative, and from this we deduce the inequality condition for $\lambda_{s(r)}$.

Minimizing the function L correspond to allowing the constraints $Bx(t) = f$, $Ax(t) = y(t)$ to be violated but at a cost: now one charges a price λ_s for any shortfall of the sum $\sum_{j \in J} A_{jr} x_r(t)$ below $y_j(t)$. From general results on convex optimization it is known there exist Lagrange multipliers (λ, μ) and a vector (x, y) such that (x, y) minimizes $L(x, y; \lambda, \mu)$, satisfies the constraints $Bx(t) = f(t)$, $Ax(t) = y(t)$, and solves the original optimization problem [13].

Kelly's solution for the Lagrange multipliers shows μ_j is the delay on the link j and λ_s is the minimum delay over all routes serving the node pair SR. The various conditions established for the multipliers thus show that an optimum of the function L, known as the objective function, corresponds to a Wardrop equilibrium.

5.2.4 Optimization through selfish behavior

The above discussions shows that if the traffic in a network distributes itself according to the self-interest of each nodes, the equilibrium flows in a manner that solves the optimization problem. The pattern of traffic resulting from individual selfish decisions behaves in the same way as if there is a central intelligence to direct and optimize a certain objective function.

An obvious example of distributed flow control on the Internet is TCP's congestion control. TCP's congestion control mechanism is implemented on the end-nodes of the network. Congestion is detected when packets are lost during the transmission. When packet loss is detected, the sender reduces its sender rate until the loss disappears. When there is no loss, the sender will resume and increase its sending rate. The use of a feedback loop allows the TCP to provide efficient flow control to different packet flows throughout the Internet.

Our approach optimizes the intermediary nodes (between entry and exit nodes of the swarm), and provides alternate routes for packet sending to achieve our desired service level. In our design, we have to also address the transient and ad-hoc nature of our transport network with an increased robustness and adaptability

[8]When the flow is 0, it can only increase, hence it can only vary upwards

that is beyond the abilities of a direct feedback based congestion control system.

The following section describes how we realize the concepts discussed above.

5.3 Allocation and Queues

Packet queues are used in many network stacks. This is a crucial component that requires optimization. The following Figure 5.5 shows commonly used Linux Network Stack [99].

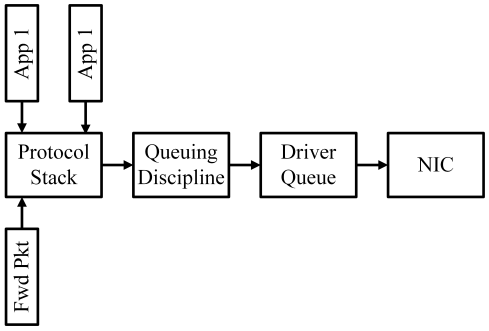

Figure 5.5: Allocation and Queues: Simplified Linux Network Stack

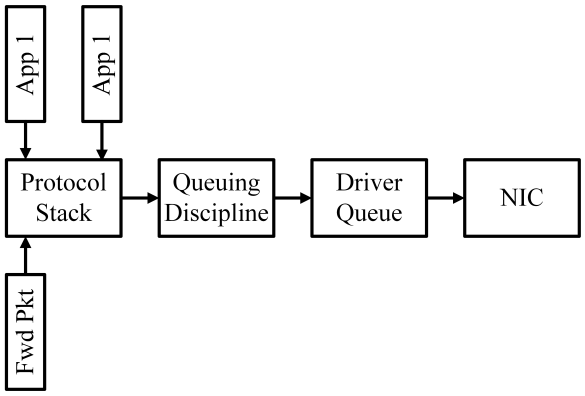

Figure 5.6: Allocation and Queues: Simplified Linux Network Stack

The driver queue is typically implemented as a first-in, first-out (FIFO) ring buffer. It contains descriptors/pointers to Socket Kernel Buffers (SKBs) that hold the packet data used throughout the kernel. The IP stack queues complete IP packets into the packet queue. These packets can be generated locally or received from another NIC when it's functioning as a router. The packets are then sent across to the NIC hardware for transmission.

Traffic classification and shaping takes place at the egress of a router. Ingress traffic flows are hard to manage as they have already arrived. To manage traffic control, various types of packet schedulers and shapers are used. Linux uses QDisc to build various controllers. QDisc defines a number of controllers such as the following.

First in, First out

`pfifo_fast` is one of the default Linux QDisc for all interfaces. It is based on conventional FIFO with limited prioritize capabilities. Higher priority bands are emptied before lower ones are processed.

Stochastic Fair Queuing, SFQ

Stochastic Fair Queuing is a probabilistic variant of Fair Queuing. Fair Queuing assigns each flow a queue, while SFQ uses a hash algorithm to assign traffic flows across a limited number of queues shown in Figure 5.6. Stochastic fairness queuing can be easily compared to strict fair queuing. In strict fair queuing, queues are serviced in a round-robin fashion, while the other uses a hash function to allocate packets to queues. To prevent prolonged collision between two packet queues, the hash function is changed periodically. Studies have shown that stochastic fair queuing can achieve a performance that approximates fair queuing and exceeds simple FCFS queuing [76]. However, note that if the packets are the same size, this degenerates to a simple round-robin service.

Token Bucket Filter, TBF

This QDisc uses tokens and buckets to shape egress traffic from an interface. Tokens are roughly correspond to bytes. Typically an additional constraint is added that each packet consumes some tokens, no matter how small it is. This models a zero-sized packet that occupies the link for sometime. The number of tokens in a bucket correspond to the amount of data that can be transmitted at any given time. If insufficient tokens are available, the packets are throttled. It is important to note that the extent of how "perfect" the traffic is shaped is limited by the kernel's ability to throttle at minimum 1 'jiffy'. For perfect shaping, only a single packet can be sent per jiffy. This is illustrated in Figure 5.7.

jiffy $= \frac{1}{f}$, f corresponds to the frequency of processing the controller. Hence if f = 1000 and (average) packet size = 1000 correspond to a 1 Mbps stream. The above QDisc represents queuing of packets within a host. We then extend these similar strategies to a closed network of nodes[9].

[9]Note that classful QDisc are not considered here.

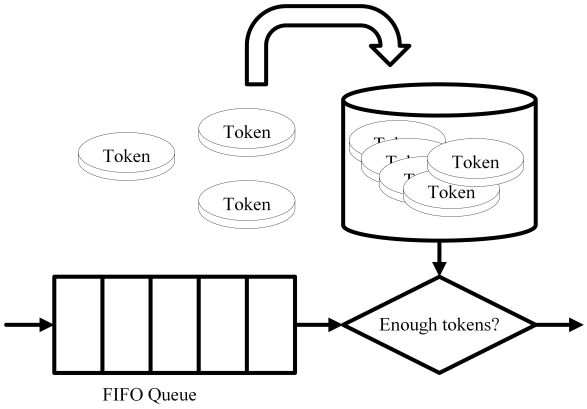

Figure 5.7: Token bucket filter, TBF

5.4 Network variability and fair usage

For any given network of nodes, strategies for fair usage depends heavily on the information available. In a uniform environment, all nodes perform identically and link capacities are equal. In such a scenario, instantaneous sampling/information about the network is not necessary for fair allocation.

Hence strategies such as round robin or random/uniform probability provides relatively fair usage of each node in the network. (Follows in a similar fashion as SFQ).

The following simulations use uniform probability as opposed to round robin. Previous studies have shown Stochastic Fair Queuing as a practical alternative to round-robin or other fair allocation schemes. The following simulations are concerned with the node/link capacity of the nodes. Traffic between FDNS and endpoints are not included. Simulation parameters for Figure 5.8 are described in Table 5.1.

As observed in the Figure 5.8 above, there are 4000 requests/tick[10]. During steady state, these traffic is replicated over 4 layers (of 250 nodes each) fairly equally. Each node in a particular layer expects to receive $\frac{\text{requests}}{\text{layer nodes}} = \frac{4000}{250} = 16$ data units(tokens)/tick. Utilization is then calculated as $\frac{\text{tokens used}}{\text{total tokens}} = \frac{16}{64} = 0.25$. Figure 5.8 shows the expected mean utilization of 0.25.

A homogeneous network does not benefit as much from scheduling strategies

[10]A tick is the smallest unit of time processed by our simulator.

Table 5.1: SFQ/D Uniform Capacity Parameters

Parameter	**Value**
Client (Request/s)	4000
Total nodes	1000
Nodes per layer	250
Peer size	16
Token buckets	Uniform (64 max / 64 refill)

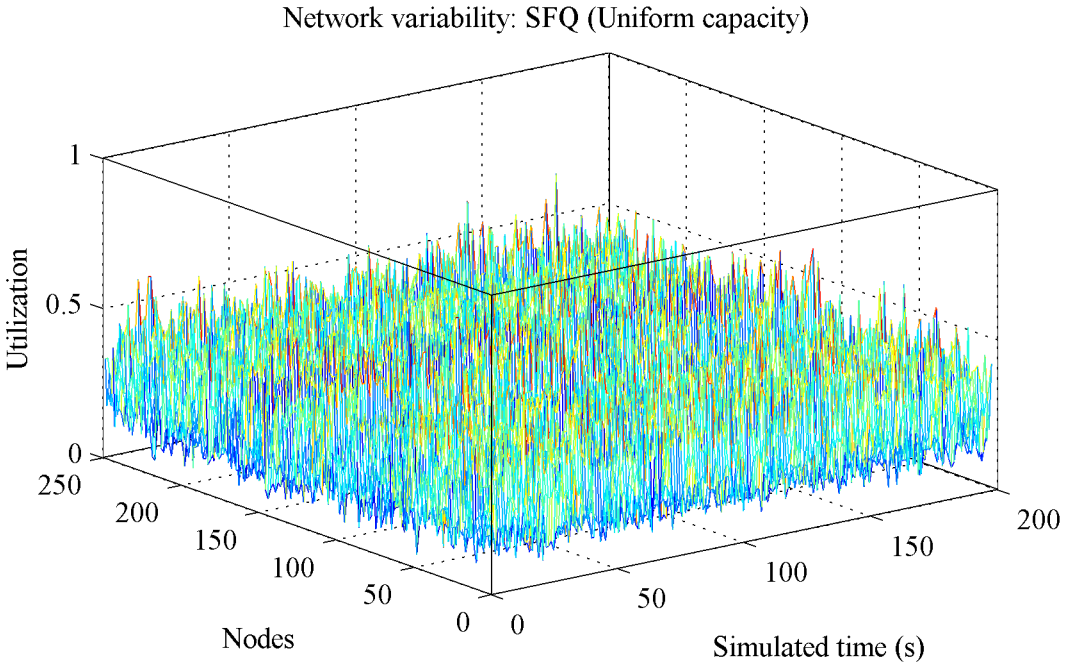

Figure 5.8: Network variability: SFQ (Uniform capacity)

described below. When all nodes have the same processing and network capability, simple round-robin scheduling or equivalents will achieve a perfectly fair allocation. Cost modelling or proportional allocation will not be needed as all nodes are under identical loads for present and future. However this is not possible in real-world networks when devices are not homogeneous. Hence, we introduce non-uniform network elements and devices. In Figure 5.9, we varied (refill) capacity between 0.8 and 1.2 (Corresponding to 51-76 units/tokens). When the capacities were varied, congestion started to occur near the middle node 150.

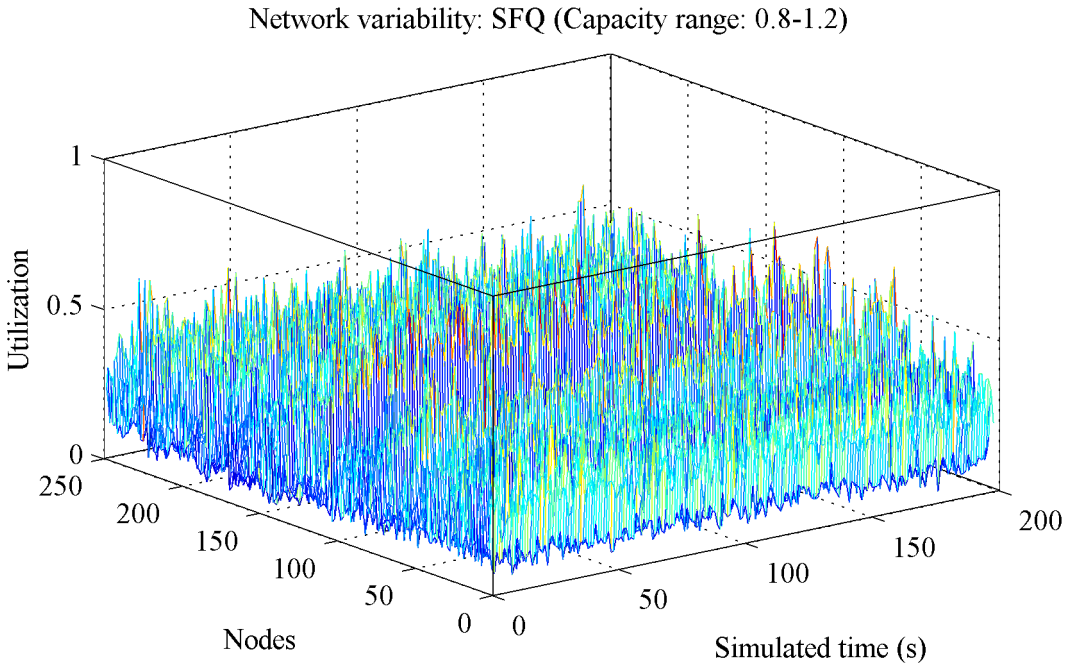

Figure 5.9: Network variability: SFQ (Capacity range: 0.8-1.2)

This problem can be addressed if we have instantaneous network state information. With knowledge of node/link capacity we can make use of the following strategies.

5.4.1 Highest Capacity (Absolute)

In this strategy, we direct the payload during one transmission interval to the node/link with the highest capacity. The parameters used are documented in

Table 5.2: Highest Tokens

Parameter	Value
Client (Request/s)	4000
Total nodes	1000
Nodes per layer	250
Peer size	16
Token buckets	Uni. Dist. (0.8x-1.2x)

Table 5.2.

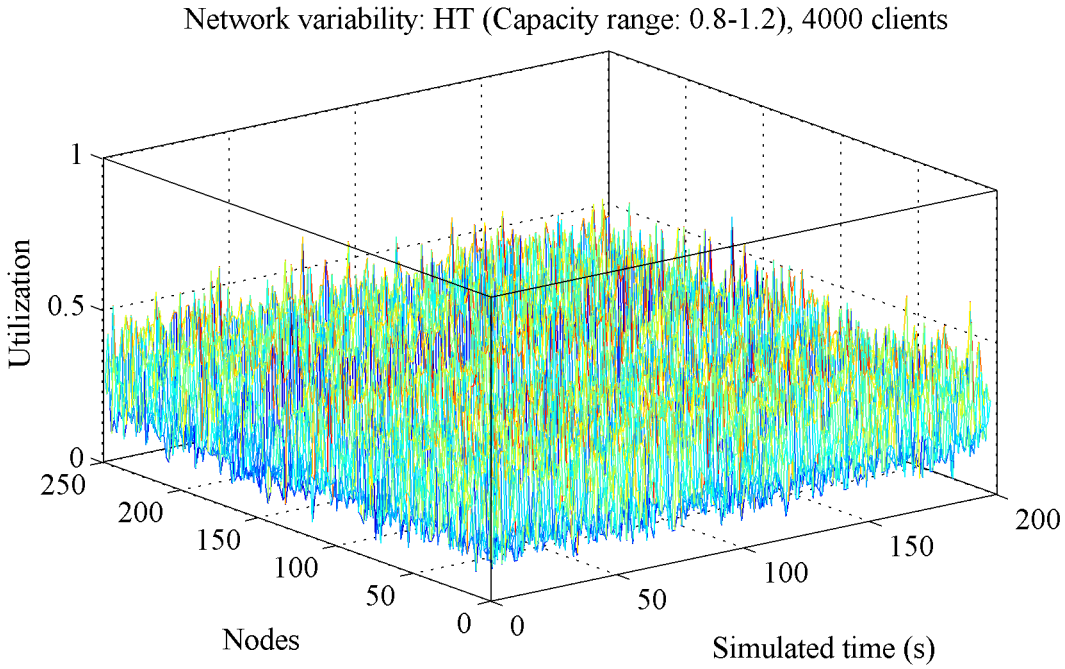

Figure 5.10: Network variability: HT (Capacity range: 0.8-1.2), 4000 clients

When we compared the results shown in Figure 5.10 to Figure 5.9, the slight congestion is reduced. However, as transmission volume tend towards the maximum capacity of the nodes, the network performs poorly. This is because the highest capacity node is overloaded quickly as all traffic is directed towards particular nodes as shown in Figure 5.11

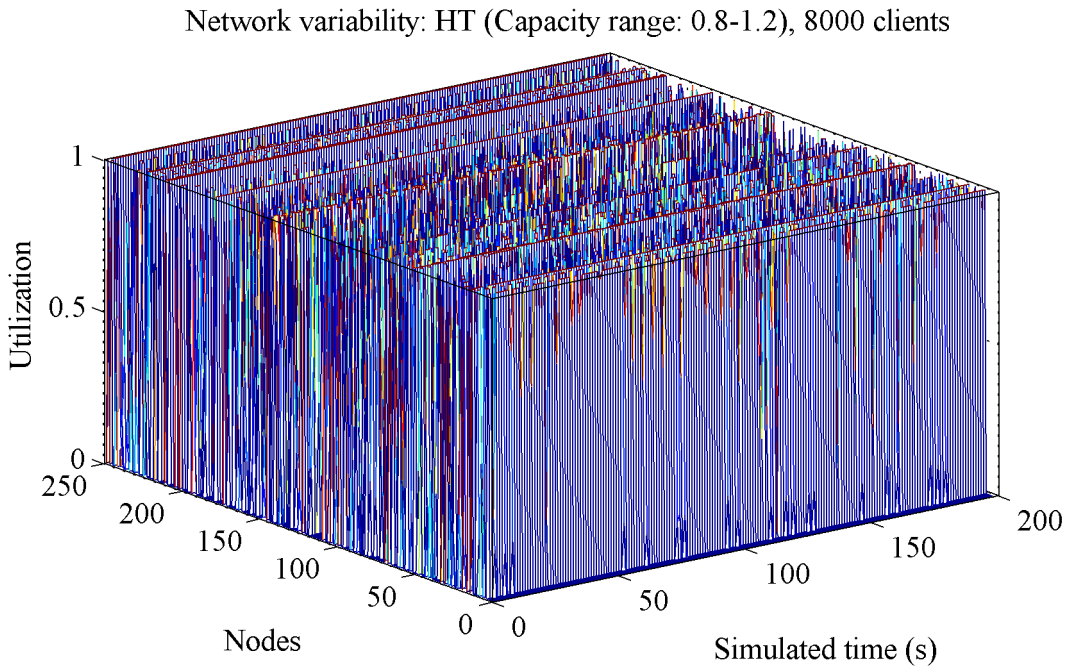

Figure 5.11: Network variability: HT (Capacity range: 0.8-1.2), 8000 clients

We observe that the amount of traffic moving through the network does not saturate every node, but only those who reported higher capacities then others.

The results shown in Figure 5.11 are further explained in Appendix A.

Considering a single source of traffic results in the following scenarios. α represents the traffic load as a percentage of maximum capacity. β represents the sustained service (refill) rate of the nodes. The summation of all increments over a time period $G_i(t)$ for node i excluding refill rate R_i can be described as follows.

$$G(t) = \sum_{i=1}^{n} (T_i(t) - T_i(t-1) - R_i) \tag{5.12}$$

$$R_j = \beta \times \max T_j \tag{5.13}$$

$$T_i(t) = T_i(t-1) + (\alpha \times \frac{1}{n} \times T_\mu) + (\beta \times \max T_i) \tag{5.14}$$

Note that $\frac{1}{n}$ can be used here because it is an expected value that is summed

up in Equation (5.15)

$$G(t) = \sum_{i=1}^{n}((\frac{\alpha}{n} \times T_{\mu}) + (\beta \times \max T_i) - R_i) \tag{5.15}$$

$$G(t) = \alpha \times T_{\mu}, \forall t \geq 1 \tag{5.16}$$

If α and β remains unchanged over time, G is time-invariant $G(t) = G$. We also do not include the token limits.

Whereby T_{μ} is the mean capacity and n is the number of peers node i has. T represents the tokens denoting capacity at instance t. Hence it is shown that the variance observed is directly correlated to α (the ratio of incoming traffic to mean capacity of the network). Furthermore, under multiple sources, this effect/congestion is compounded.

5.4.2 Highest Capacity (Proportional)

The above problem could be addressed if we distribute the load proportionally. To distribute the traffic proportionally across its connected peers, we construct a probability table. This allows us to allocate traffic volume without knowing the total communication quantum in a given transmission period. The probability of selecting host y is described in Equation 5.17.

$$P(x = y) = \frac{T_y(t)}{\sum_{i=1}^{n} T_i(t)} \tag{5.17}$$

Where $T(t)$ is the tokens representing a peer/link at time t.

We distribute the traffic according to instantaneous node/link capacity (tokens) proportionally in a uniform network environment with the following parameters in Table 5.3. Results are shown in Figures 5.12 (4000 clients), 5.13 (8000 clients)and 5.14 (16000 clients).

This significantly reduces the variance observed from before.

The following describes the traffic going through these nodes. $G'(t)$ is the summation of all increments similarly defined in Equation (5.12).

Table 5.3: Highest Tokens (Proportional, Probabilistic)

Parameter	Value
Client (Requests)	4000/8000/16000
Total nodes	1000
Nodes per Zone	250
Peer size	16
Token Buckets	Uni. Dist.

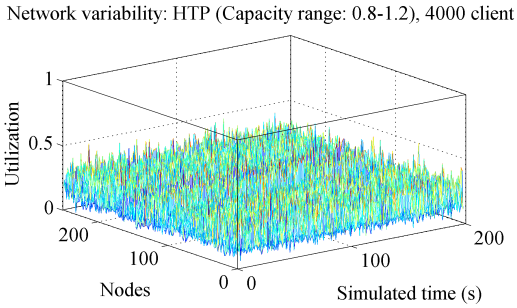

Figure 5.12: Network variability: Highest Tokens (Proportional) 4000 clients

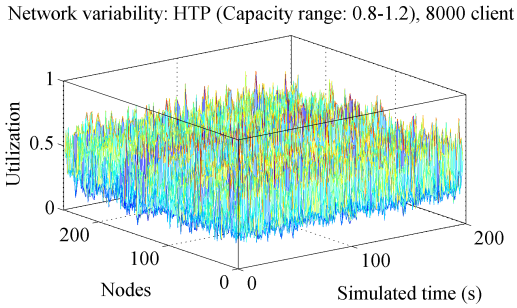

Figure 5.13: Network variability: Highest Tokens (Proportional) 8000 clients

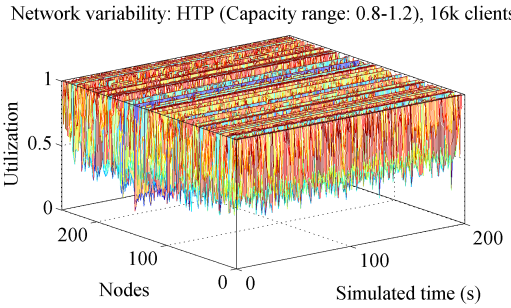

Figure 5.14: Network variability: Highest Tokens (Proportional) 16000 clients

$$G'(t) = \sum_{i=1}^{n}(T_i(t) - T_i(t-1) - R_i) \tag{5.18}$$

$$R_i = \beta \times \max C_i \tag{5.19}$$

$$T_i(t) = T_i(t-1) + (\alpha \times \frac{T_i(t)}{\sum_{j=1}^{n} T_j(t)} \times T_\mu) + (\beta \times \max T_i) \tag{5.20}$$

$$G'(t) = \sum_{i=1}^{n}((\alpha \times \frac{T_i(t)}{\sum_{j=1}^{n} T_j(t)} \times T_\mu) + (\beta \times \max C_i) - R_i) \tag{5.21}$$

$$G'(t) = \alpha \times T_i(t), \forall t \geq 1 \tag{5.22}$$

This correspond to the same Equation (5.16).

However, given both G and G' are the same, variance/delta between $T(t)$ and $T(t-1)$ are different. We now let $V(t)$ and $V'(t)$ be functions that describe the variation.

$$V(t) = \sum_{j=1}^{n} v_i(t)^k \tag{5.23}$$

$$v_i(t) = T_j(t) - T_j(t-1) - R_j \tag{5.24}$$

For absolute allocation, there will only be 1 peer that is responsible for delta v^k. However, for proportional allocation, 1 or more peers will be used (e.g. $v_1^k + v_2^k + ...$).

$$\sum_{j=1}^{n} v_j = m, m^k \geq \sum_{j=1}^{n}(v_j)^k \tag{5.25}$$

Where $V(t)$ represents absolute allocation and V' represents proportional allocation. Therefore the following can be concluded.

$$V(t) \geq V'(t) \tag{5.26}$$

When we allocate flows according to the available capacity, congestion is reduced as shown in Equation (5.26).

5.5 From tokens to utilization measure

In the previous sections, we have shown that having instantaneous network information allows more efficient allocation. However, estimation or sampling have shown to be difficult. To overcome this, we propose that individual nodes report and act upon on both their perceived utilization as well as their peers.

When we use node-side information, there are various strategies that can be employed. Two strategies will be discussed in detail.

1. Inversely correlated to utilization (Equation 5.27)

2. Linearly correlated to utilization (Equation 5.28)

These two functions are chosen to represent both addictive and multiplicative allocation of traffic flows. They are used as control benchmarks used to evaluate our cost-modelling approach shown later.

Utilization is measured as the current load over the maximum load of the node. In our simulations, a token bucket algorithm is used. The current utilization is the expended tokens over maximum size of the bucket. In our prototype, it is the current throughput (requests/second) over the maximum throughput of the node.

5.5.1 Proportional allocation (Exponential)

Similar to forwarding traffic towards the node with the highest capacity, we allocate traffic according to its reported utilization. The probability of selecting a particular host y is described in Equation (5.27).

$$P(x = y) = \frac{\frac{1}{u(t)_y}}{\sum_{i=1}^{n} \frac{1}{u(t)_i}} \tag{5.27}$$

For example, a small group of 4 nodes with utilization A = 0.1, B = 0.2, C = 0.3 and D = 0.4 respectively. These utilization indexes can be calculated from the node.

Note that $\sum_{i=1}^{4} \frac{1}{u(t)_i} = 20.833$

Table 5.4: Group of 4

Node	1	2	3	4
$u(t)_y$	0.1	0.2	0.3	0.4
$P(x = y)$	0.480	0.240	0.160	0.120

The traffic is then divided proportionally in accordance to its reported utilization similar to Highest Tokens. However, this causes 2 problems of its own. The first challenge comes from the over-aggressive nature of allocating traffic corresponding to the inverse of its utilization. Nodes with low utilization receives much more traffic and hence overload easily as shown in Figure 5.15.

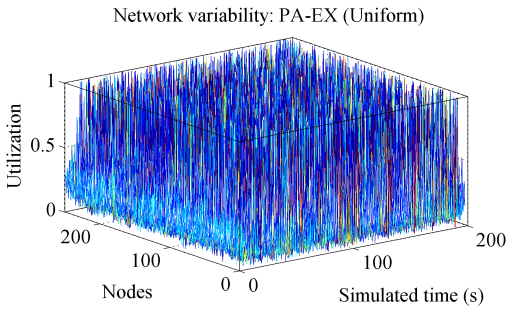

Figure 5.15: Token and Utilizations: PA-EX (Uniform capacity) 4000 clients

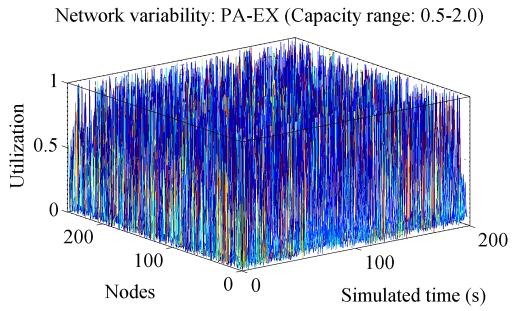

Figure 5.16: Token and Utilizations: PA-EX (Capacity range: 0.5-2.0 4000 clients)

The second challenge is the unfair allocation of traffic. There is no network information to correlate and normalize perceived utilization and its actual instantaneous capacity. This means that 2 nodes with utilization 0.5 do not have the same unused capacity. Hence, as the network varies more, effectiveness of allocation reduces as we compare Figure 5.16 with a uniform Figure 5.15 (Note the percentage of utilization between 0.2-0.4 on the latter Figure).

5.5.2 Proportional allocation (Linear)

We distribute traffic linearly to overcome over-aggressive allocation of traffic among nodes with different utilization.

The probability of selecting host y is described in Equation 5.28.

$$P(x = y) = \frac{(1 - u(t)_y)}{\sum_{i=1}^{n}(1 - u(t)_i)} \tag{5.28}$$

Where $u(t)_i$ is the utilization reported by the node i at time t.

With similar parameters as before and a uniform capacity distribution, results are shown in Figure 5.17.

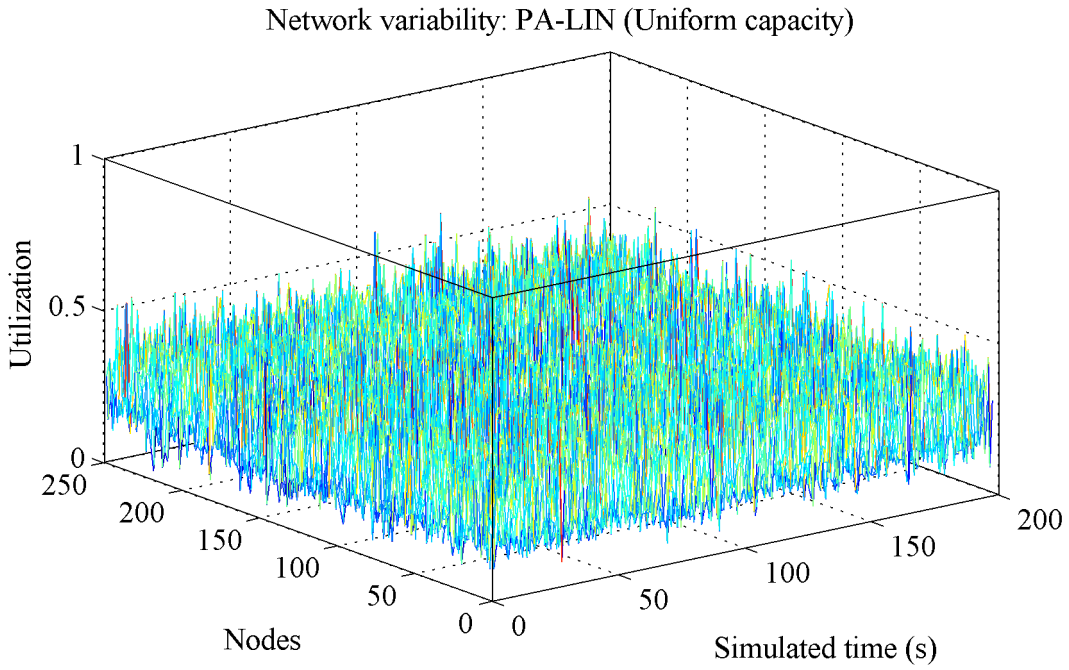

Figure 5.17: Token and Utilizations: PA-LIN (Uniform capacity), 4000 clients

The above shows that the relationship between utilization and available capacity must be studied further in order to maximize efficient allocation.

5.5.3 Additional simulations (Capacities: 0.8-1.2,0.5-2.0)

A number of simulations were performed with various network capacities of (1) 0.8-1.2x and (2) 0.5-2.0x

Capacities 0.8-1.2x

Figure 5.18 shows the spread of network capacities among the network for 0.8-1.2x. This corresponds to a refill rate of 51-76 tokens/tick. This network results in the following. Figure 5.20 shows the performance benchmark when instantaneous network information is available. Figure 5.20 shows similar congestion effects described earlier. Figure 5.21 shows better performance as compared to 'PA-EX'.

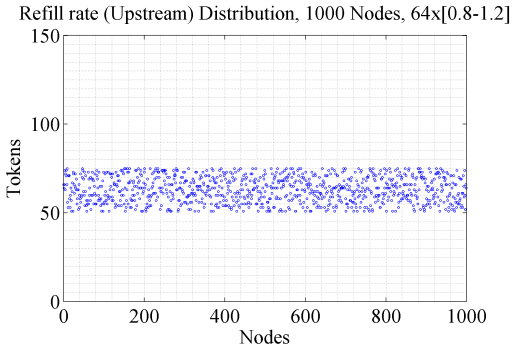

Figure 5.18: Distribution of refill rates, 51-76

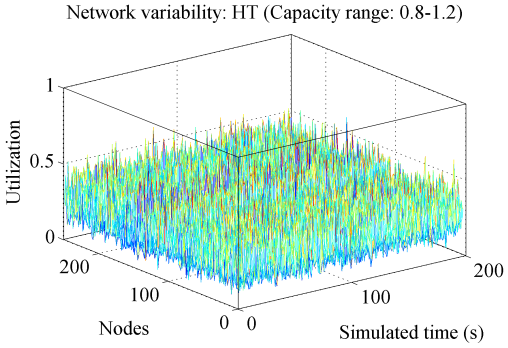

Figure 5.19: Token and Utilizations: HT (Capacity range: 0.8-1.2) 4000 clients)

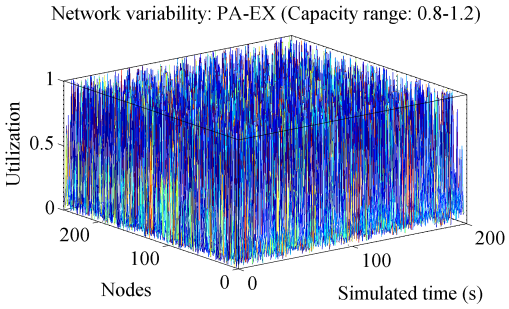

Figure 5.20: Token and Utilizations: PA-EX (Capacity range: 0.8-1.2) 4000 clients

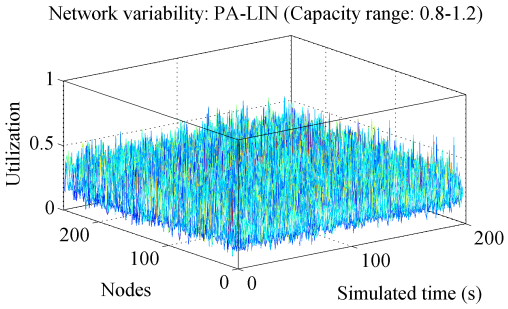

Figure 5.21: Token and Utilizations: PA-LIN (Capacity range: 0.8-1.2) 4000 clients)

Capacities 0.5-2.0x

Figure 5.22 shows the spread of network capacities among the network for 0.5-2.0x.

Figure 5.22 shows a significantly wider spread as compared to 0.8-1.2x The fill rate corresponds to 32-128 tokens/tick. The following Figures 5.25, 5.24 and 5.25 shows the corresponding results.

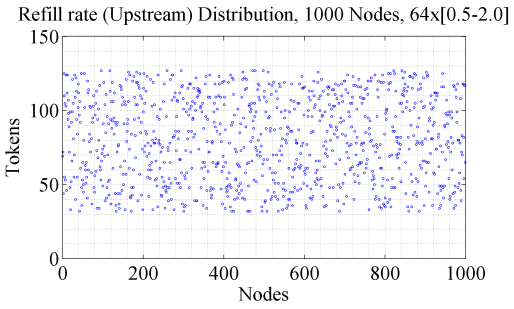

Figure 5.22: Distribution of refill rates, 32-128

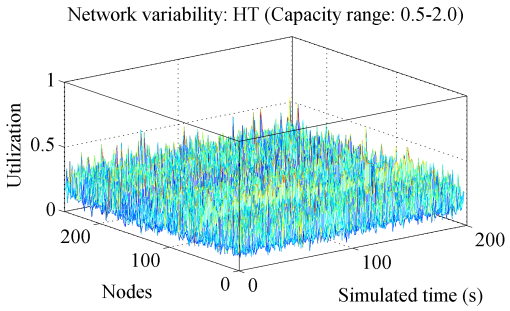

Figure 5.23: Token and Utilizations: HT (Capacity range: 0.5-2.0) 4000 clients

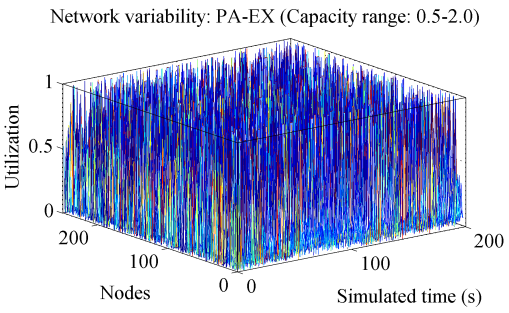

Figure 5.24: Token and Utilizations: PA-EX (Capacity range: 0.5-2.0) 4000 clients

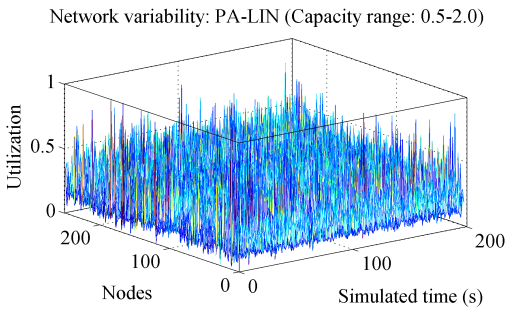

Figure 5.25: Token and Utilizations: PA-LIN (Capacity range: 0.5-2.0) 4000 clients

5.5.4 Sampling intervals

In the previous section, (1) highest tokens, (2) lowest utilization and (3) highest availability have almost instantaneous update of the network state. As the overhead chatter by the link update would be too high (Equation 4.6), the rate of updates must be reduced. This will cause further allocation inefficiencies and variances.

5.6 Water flow

In the following section, we introduce our strategy to mitigate challenges described above.

5.6.1 Swarm Intelligence

Swarm Intelligence is a collective behaviour of decentralized, self-organized systems, natural or artificial [15]. Swarm Intelligence comprises of collective behaviours of (unsophisticated) agents interacting locally with their environment for a global patterns to emerge.

A high level view of a swarm suggests that all the agents in a swarm cooperate with one another to achieve some purposeful behaviour and goal. This results in an apparent "collective intelligence" that emerges from what often is a large groups of relatively simple agents. These agents use simple rules to direct their actions. Through interactions of the entire group, the swarm achieves its objective. This is an example of how "self-organization" emerge a collection of actions of the swarm.

This swarm intelligence is the emergent collective intelligence of groups of simple autonomous agents. Autonomous agents act relatively independently from all other agents. These autonomous agents do not follow commands from a leader, or some global plan. This can be illustrated from the practice of flocking birds. A flocking bird that participates in a flock, it adjusts its movement to coordinate with the moment of its neighbours. There is no specific hierarchy that comes with a flock of birds. Each participating bird can join the flock at any position, with no restrictions [71].

In summary, a typical swarm intelligence system consists of the following behavioural components.

1. It is composed of a large number of similarly behaving individuals.

2. The interaction among the individuals is based on simple behavior rules.

3. The overall behavior of the system is a result of the interaction of the individuals with the environment.

5.6.2 Water and Soil inspiration

Fast-fluxing Domain Name Servers (FDNS) only provides the means to direct the traffic to desired nodes for relaying. The decision model used to formulate these alternate routing paths requires a form of intelligent re-organization. To achieve this goal, we apply techniques associated with swarm intelligence to the network. This allows the network to self-regulate and exhibits autonomy. The self-organizing capability of the swarm network allows it to optimize its routing solutions.

Our implementation is based on the Intelligent Water Drop algorithm [96] to allow the swarm network to determine the solution to the optimization problem. The movement of the water in nature is the underlying inspiration for the Intelligent Water Drop algorithm. It is evident in the observation of flowing water, that water always finds the path of least resistance. This provides us with a simple directive in solving a potentially complicated optimization problem.

The Intelligent Water Drop (IWD) algorithm consists of (1) Velocity and (2) Soil. These two parameters vary considerably throughout the lifetime of the IWD algorithm. The rate of change is directly correlated with the changing nodes and network conditions. As the IWD moves from the source to the destination, it affects the soil it passes through. The amount of soil at each location directly affects whether the IWD gains or loses speed. The IWD also carries soil along with it, to reflect its own impact on the network. The movement of soil between the nodes forms the memory of the network. The IWD, like its real-world counterpart, will move along the path of least resistance. This allows it to solve network optimization problems.

IWD exhibits an advantage over other network optimization solutions by virtue that it does not require access to the entire problem space to start optimization. IWD can perform partial optimization with the portion of the network within its domain. The soil parameters can change to reflect a large variation in the network environment without impeding the operation of the IWD algorithm. The swarm is designed to be a self-replicating network. Hence each node at different levels behaves in an iterative manner.

The swarm makes use of the methodology of intelligent water drop to determine the most effective route to relay messages between the client and servers. The speed

and latency of the nodes are used as features (inputs) for the IWD algorithm. The amount of delay incurred by going through any particular node can be used as a measure of the amount of soil between two nodes.

IWD is preferred over other meta-heuristic algorithms because it most resemble our traffic allocation problem as opposed to Ant Colony Optimization [100]. Our aim is to demonstrate that swarm algorithms such as IWD can be adapted to serve as a distributed flow controller and provide continuous optimization.

5.6.3 Algorithm

To address the aforementioned deficiencies due to (1) cost modelling and (2) sampling interval, we propose and extend a Water flow model and algorithm. The cost of each transmission differs between each host. Individual models will be allocated for various node/link capacities. As link updates are infrequent, maintaining a stateful model of each link allows us to track its consumption. This allows us to mitigate the large variances caused by gaps in sampling.

Our Water flow (WF) algorithm can be described as follows.

1. Initialization/Update of parameters.

2. Each node i chooses the next node j with the following probability model P_i^{WF}.

$$P_i^{WF}(x = j) = \frac{f(soil(i,j))}{\sum_{k \forall n_i} f(soil(i,k))} \tag{5.29}$$

3. As data is transmitted from node i to node j the velocity $v_{(i,j)}$ is updated as follows.

$$v_{(i,j)}(t+1) = v_{(i,j)}(t) + \frac{a_v}{b_v + (c_v \times s_{(i,j)}^{d_v})} \tag{5.30}$$

4. Soil $s_{(i,j)}$ is then updated corresponding to the velocity of the flow.

$$soil_{(i,j)}(t+1) = \big(e_v \times soil_{(i,j)}(t)\big) + \frac{a_s}{b_s + (c_s \times v_{(i,j)}^{d_v})} \tag{5.31}$$

We can also include a penalty/influence function to the above to reduce the rate of accumulation of soil due to other inputs.

$$soil_{(i,j)}(t+1) = \big(e_v \times soil_{(i,j)}(t)\big) + \frac{a_s}{b_s + (c_s \times f(v_{(i,j)}, t)^{d_v})} \tag{5.32}$$

Whereby $f(v,t) = v + v_{(i,j)} \times \rho(t)$, $\rho(t)$ is the penalty at time t

In the following simulations, we do not change the rate of accumulation, but use constants a, b, c, d and e instead.

5. This algorithm is then repeated for every transmission. (Go back to 1)

Note that a_v, b_v, c_v and d_v are tuning parameters. These parameters need to co-relate (linearly or exponentially) with the size of individual packets. To show the viability, we simulate networks with varying capacity distribution. (1) 1-2x and (2) 1-20x. Further results 1-5x, and 1-10x can be found in Appendix B.

5.6.4 Simulations

Capacity Distribution, 1-2x

Similar to simulations earlier, the following is a Java implementation whereby each node has a token bucket filter. This allows us to control the traffic flow and determine utilization of each nodes. The following are results of 1-2x capacity distribution. 1-2x correspond to a refill rate of 64-128 units (tokens). Figure 5.26 shows an allocation strategy for the highest capacity node/link. This is used as a benchmark for our approaches.

For both lowest utilization and highest capacity, the unfair capacity distribution and lack of normalization mechanism results in poor performance. This is shown in Figure 5.27 and Figure 5.28. Using our Water flow approach, we are able to achieve similar performance to Highest Capacity as shown in Figure 5.29.

Capacity Distribution, 1-20x

The following Figures 5.30, 5.31, 5.32 and 5.33 are results of 1-20x capacity distribution. 1-20x correspond to a refill rate of 64-1280 units (tokens).

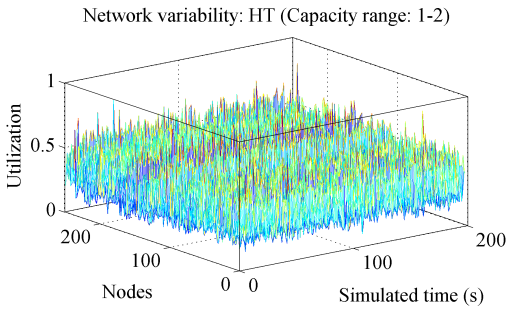

Figure 5.26: WF Compare: HT
8000 clients, 1-2x
$\mu = 0.3347$, $\sigma = 0.1007$

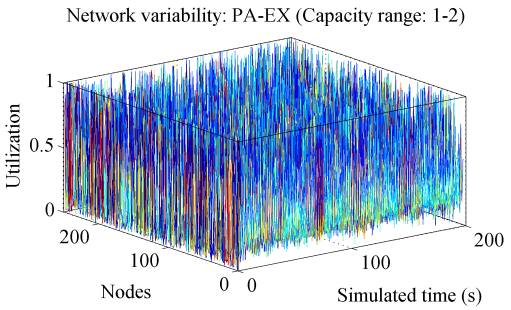

Figure 5.27: WF Compare: PA-EX
8000 clients, 1-2x
$\mu = 0.3427$, $\sigma = 0.2057$

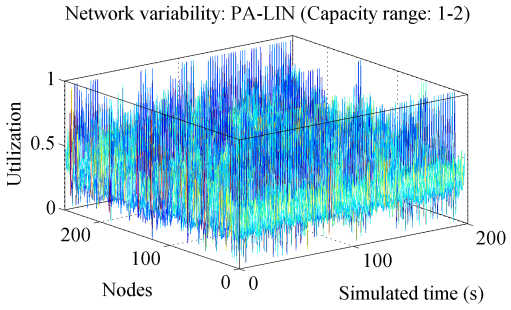

Figure 5.28: WF Compare: PA-LIN
8000 clients, 1-2x
$\mu = 0.3426$, $\sigma = 0.1182$

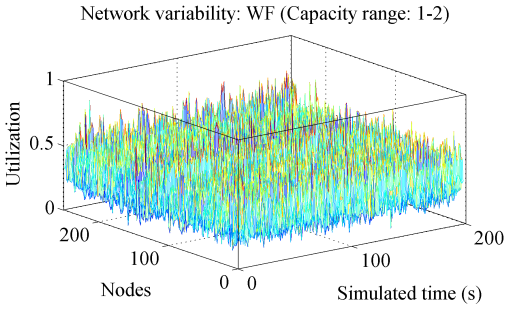

Figure 5.29: WF Compare: WF
8000 clients, 1-2x
$\mu = 0.3347$, $\sigma = 0.1049$

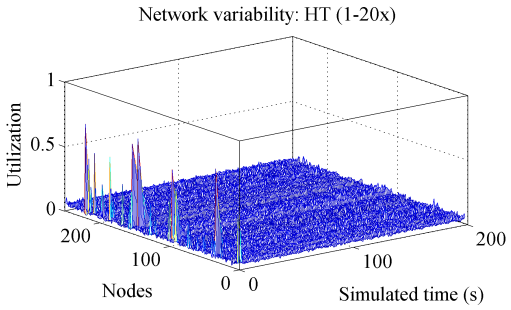

Figure 5.30: WF Compare: HT
8000 clients, 1-20x
$\mu = 0.047372$, $\sigma = 0.02044$

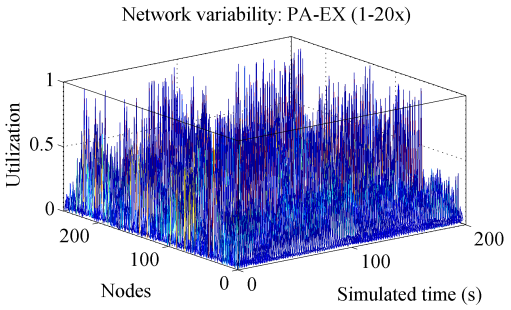

Figure 5.31: WF Compare: PA-EX
8000 clients, 1-20x
$\mu = 0.06753$, $\sigma = 0.10307$

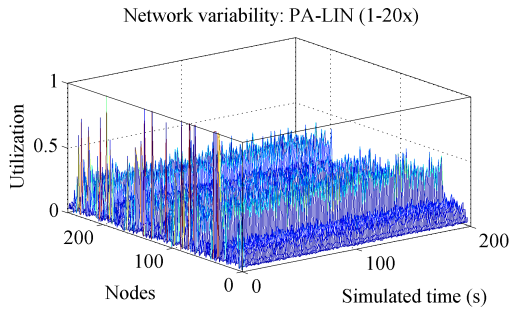

Figure 5.32: WF Compare: PA-LIN
8000 clients, 1-20x
$\mu = 0.07232$, $\sigma = 0.06900$

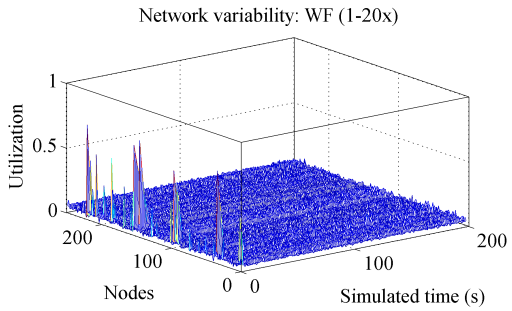

Figure 5.33: WF Compare: WF
8000 clients, 1-20x
$\mu = 0.04736$, $\sigma = 0.02038$

The above results shows that our proposed model compensates for different network capacities. They are summarized in the following Table 5.5. We show that WF is able to achieve similar μ and σ performance as the benchmark HT.

μ is the average utilization of each node, while σ is the variation of utilization between nodes.

Table 5.5: Summary of 'Waterflow' experiments

Capacities / μ	**HT**	**PA-EX**	**PA-LIN**	**WF**
1-2x	0.3347	0.3427	0.3426	0.3347
1-5x	0.16762	0.18414	0.19208	0.16754
1-10x	0.09006	0.11087	0.11610	0.09011
1-20x	0.047372	0.06753	0.07232	0.04736
Capacities / σ	**HT**	**PA-EX**	**PA-LIN**	**WF**
1-2x	0.1007	0.2057	0.1182	0.1049
1-5x	0.05288	0.18795	0.09237	0.05324
1-10x	0.02959	0.14048	0.07482	0.02995
1-20x	0.02044	0.10307	0.06900	0.02038

5.7 Summary

In this chapter, we show how swarm algorithms can be used for distributed traffic management. We describe a simple network representation and use it to show how

optimization by individual nodes can approximate a global solution. We then show various algorithms (1) highest capacity, (2) proportional allocation (exponential and linear) perform with nodes of different performances in Figures 5.18 - 5.25. We then modify and adopt a swarm algorithm - Intelligent Water Drop to perform distributed route selection. We show how we can achieve similar performance to highest capacity benchmark with our adopted WF algorithm. Results are shown in Figures 5.26 - 5.33.

Chapter 6

Prototype

6.1 Overview

In this chapter, we describe building of a prototype that exhibits features described in the previous chapters. A description on the various software packages, frameworks and methodology is presented in Section 6.2. Description of the hardware and benchmarking tools is provided in Section 6.3. Two set of results including (1) Reinforcement, Resistance (from Chapter 4) and (2) Flow Optimization (from Chapter 5) will be discussed.

6.2 Software approach

The following Figure 6.1 provides a summary of techniques and frameworks used in our prototype. Java.IO, Java.NET and Grizzly provides frameworks for both inter and intra overlay communications. Our contributions are shaded as shown in Figure 6.1

6.2.1 Event driven, HTTP Servicing

For fast-flux session binding to take place, gateways have to maintain a persistent connection to the clients. This allows the client to receive a reply from the servers within the same HTTP request/response.

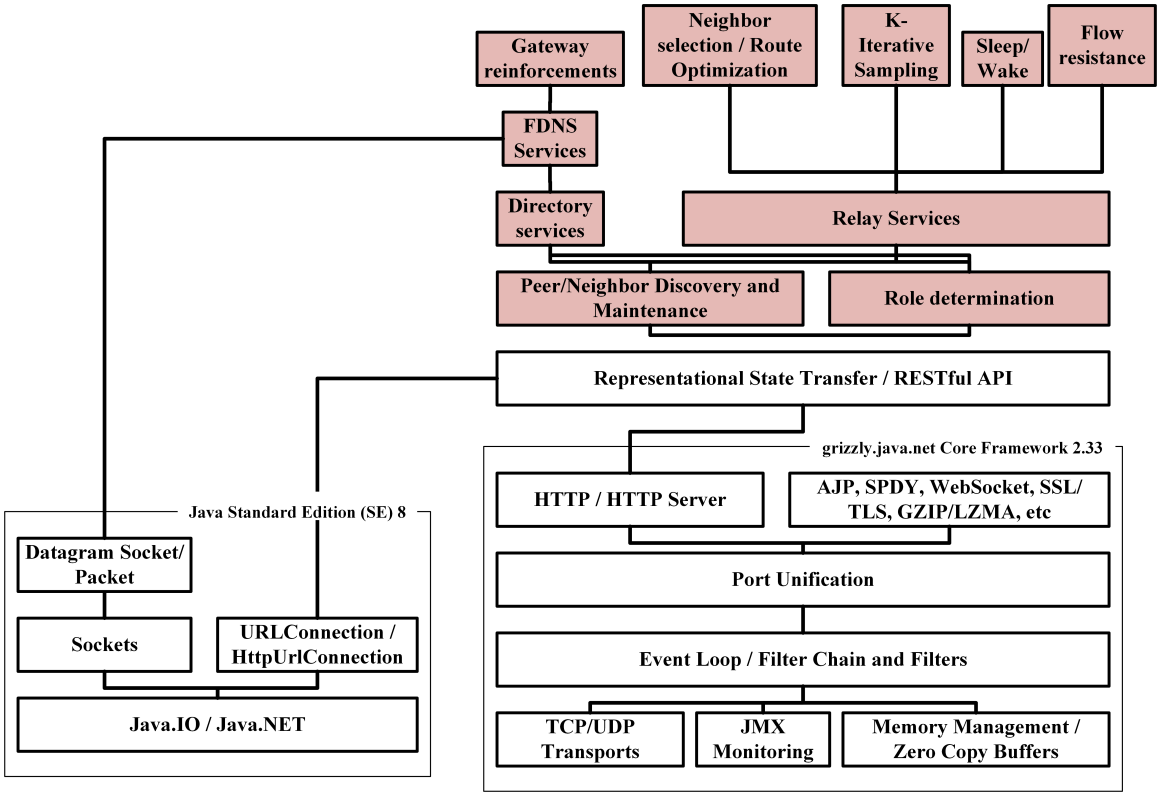

Figure 6.1: Software frameworks and packages

However, if relays are 'activated' one after another, each node will have poor performance. Threading the listeners takes up large amount of resources, while providing a modest performance gain. To address the large throughput required, we use event-driven Java NIO.

However, as a relay takes place across a number of nodes, a particular request has a tendency to become a long running transaction. This becomes problematic with an NIO-based server. Every node/server has limited number of threads, and a long running transaction would consume one thread. If this occurs frequently, this easily leads to a denial of service. This is mitigated by allowing a response to be suspended until it can be serviced. This process is illustrated in Figure 6.2.

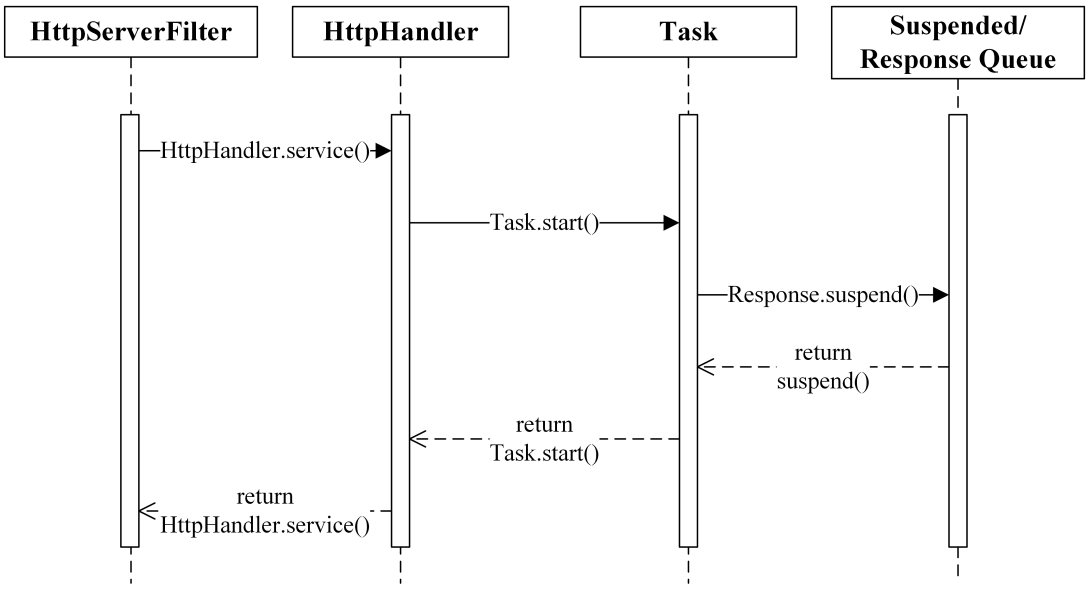

Figure 6.2: HTTP and Java NIO

6.2.2 Applying Representational State Transfer

Representation State Transfer, REST is used for all communications between nodes with the exception of domain name services. HTTP GET/POST are used to invoke appropriate Handlers. Additional information is encapsulated within header values or http content. For example, Hop state information is embedded within the HTTP header's `proxy` field. Neighbour lists are pushed through POST or received as a

consequence of a HTTP request. The following is a summary of REST commands used.

- **Shared services** (Used by all nodes regardless of roles.)
  - `peerGet` and `relayGet` - They are used to retrieve neighbor lists from seeds and peers.
  - `peerPush` and `relayPush` - Used to push neighbor lists within the network.
  - `ping` / Heartbeat - Used to determine if a node is alive or active. For relays, this is further specified to a `kGet` to address the same requirement.
- **Seeds/FDNS/Directories**
  - `relayRegister` - Used intitially by a node to register itself with a seed.
  - `Reinforce` - Optionally used by seeds to ask a particular node to perform graceful and timely reallocation. Directory servers can move nodes from other zones to gateway by modifying the FDNS table only.
- **Relays**
  - `relayRegister` - Used intitially by a node to register itself with a seed.
  - `Reinforce` - Used to gracefully remove itself from the neighbor list and restart itself.
  - `relay` (`open`, `forward` and `push`) - Used to invoke the relay of http requests.
  - `kGet0`, `kGet1` and `kBigK` - To gather performance measures. In this context, it is used to gather a normalized utilization measure from each node and their subsequent higher order utilization estimates.
  - `kSleep`, `kHold` and `kWake` - To ask another node to go to sleep, maintain its current status and wake a sleeping node.
  - `blacklist` - Used to ask a node to reduce/stop relaying requests from a particular client IP.

6.2.3 Refresh intervals and Timings

Maintaining integrity of the network requires (1) Peer discovery, (2) Neighbour state and (3) K Decisions to take place periodically. Peer discovery is pursued aggressively when the node starts. This helps it build its neighbour list quickly. Node queries for neighbours every 1-2 seconds. The query interval is reduced to once every few minutes when the list is filled or a specified time has elapsed[1]. Excess nodes are then used to replace dead nodes, or swapped with existing nodes to ensure diversity[2].

`ping` / Heartbeat is integrated with `kGet` to reduce background network chatter. This overhead is categorized in the earlier section 4.3.2. Similar to `kGet(0)`, `kGet(1)` contributes to the background traffic in a similar manner. K-Decisions takes place at a much lower frequency then above.

6.3 Setup

6.3.1 Hardware and Networking

- Computers
  - 12 Desktops/Workstations
  - 28 Notebooks
- Servers
  - 4 x 1U servers
- Networking
  - 1 x Gigabit Router
  - 1 x Gigabit Switch (16 ports)

[1] A maximum query time is used to prevent excessive neighbour list query when there are insufficient peers. When requesting for neighbour list, each node responses with the nodes it has. It does not provide unique hosts to the requester. Hence it is prone to waste when there are insufficient hosts.

[2] We do not churn the neighbour list due to data collection constraints

  - 1 x Ethernet Switch (24 ports)
  - 2 x Ethernet Switch (16 ports)

- Data collection / Storage
  - 1 x Network Attached Storage (8 Disk, Raid 6)
  - 40 x USB drives as secondary storage for clients

Performance of each node is made up both link delay and processing speed. During initial tests when nodes are geographically spread in multiple off site locations, Round Trip Times (RTT) were large enough to contribute to the total time required to service each request. However, as we move the bulk of the network into our own test range, RTT is reduced significantly and were negligible regardless of network load. However, processing capability of each node now becomes the bottleneck of each 'link'. Hence, in this prototype we emphasis on processing speed (K-value) of each node rather than link delay as a function of performance measure.

To facilitate our study, we make use of the network topology illustrated in Figure 6.3. Off site locations are initially used to verify assumptions regarding load and `RTT`. Within the demilitarized network, nodes are connected as shown in Figure 6.4.

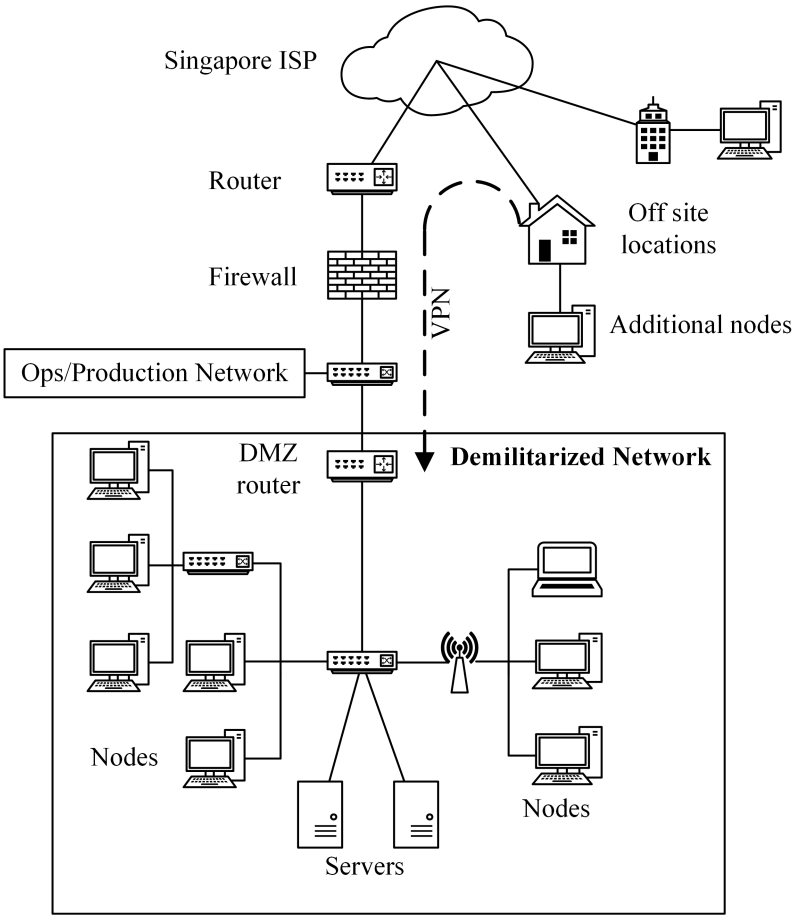

Figure 6.3: Network topology

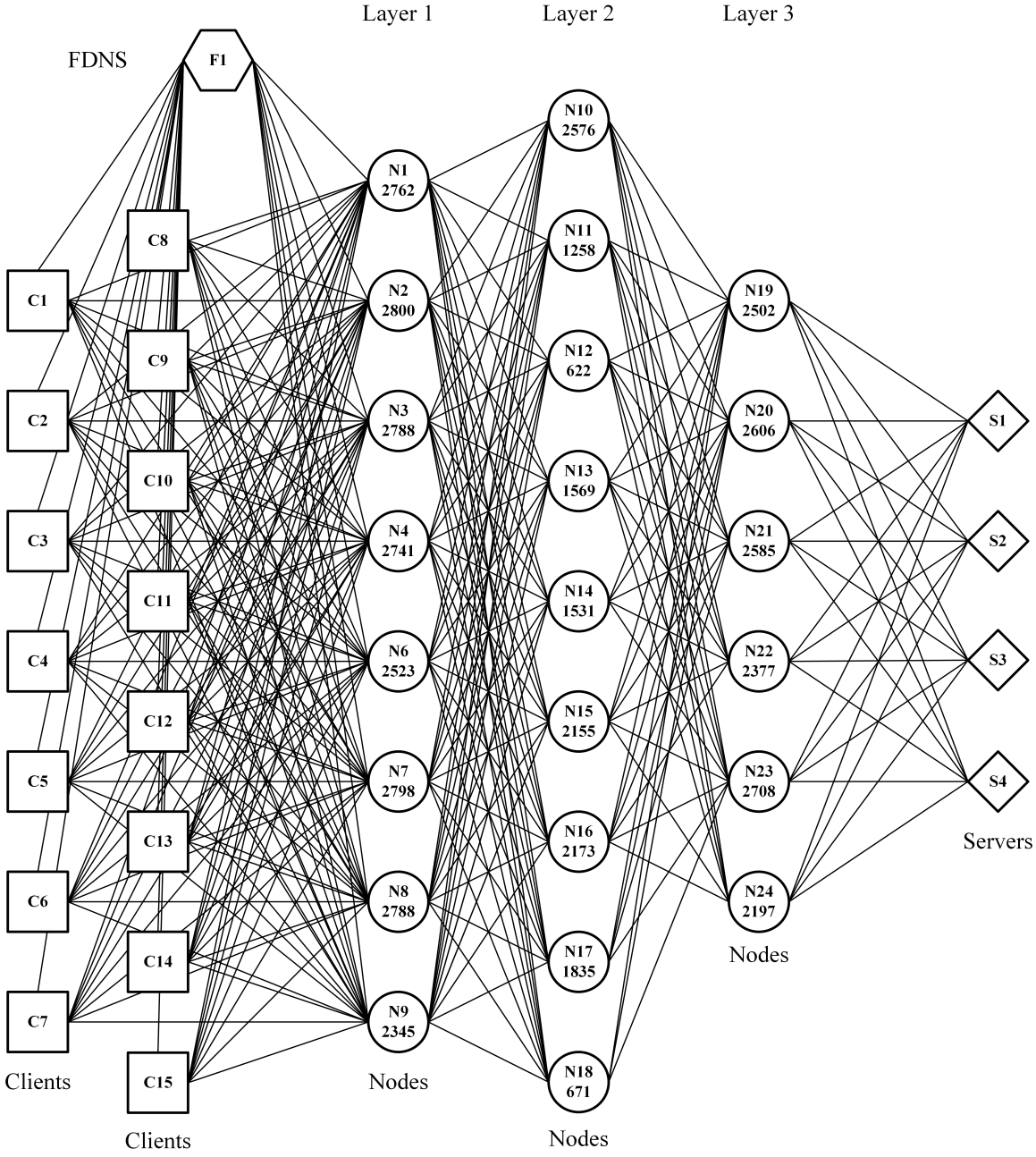

Figure 6.4: Overall connectivity

6.3.2 Generating traffic and benchmarking

Client use-case, session binding

To demonstrate a user base connecting to the network, each participating machine uses a browser. They are set to repeat a query as fast as possible.

Each user's operating system uses our FDNS node to resolve a gateway node address. The user is then bind-ed to particular node for Time-To-Live duration. In the following context, the TTL is set to 5 seconds. Any real world deployments should use a much larger TTL value to reduce over head at the FDNS [97]. This round-robin effect can be observed from Figure 6.5. Service rate of individual nodes varies significantly during the experiment. This shows that an attack is distributed among all the gateway nodes (1-6).

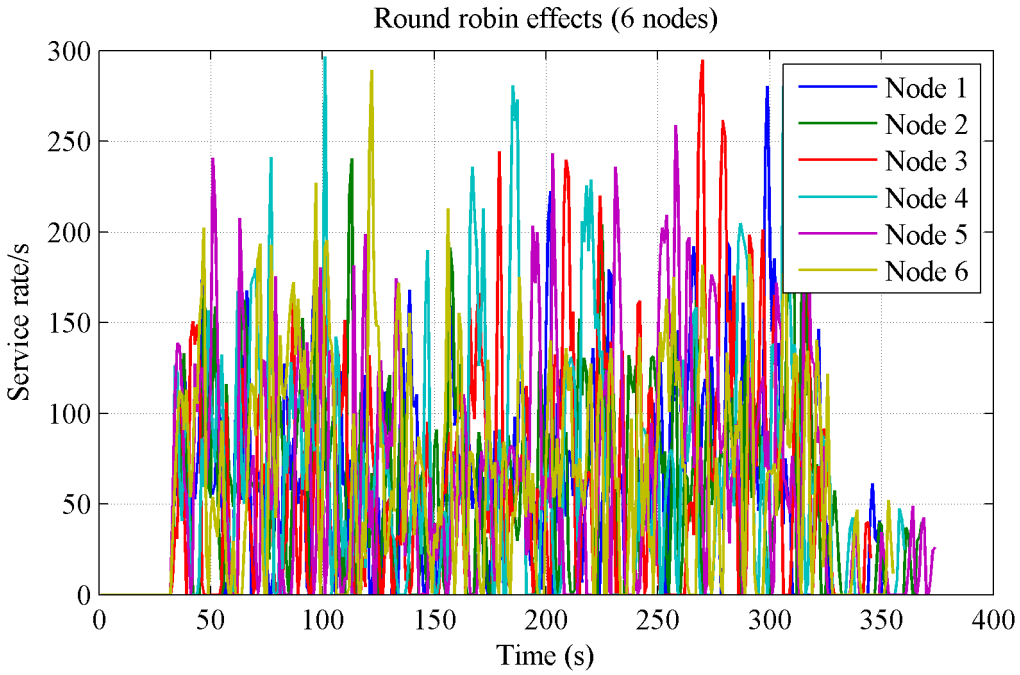

Figure 6.5: Round robin effects

Direct throughput benchmarking

To saturate available bandwidth, we use a collection of nodes to perform ApacheBench. ApacheBench (AB) was originally used to test Apache HTTP Server, but it is

generic enough to be used for our NGINX clusters. Each gateway node has at least one AB node attached to it. The average transactions speeds over 500K or 1M attempts are then recorded. We estimate each node's performance by executing (1) session binding and (2) retrieving data from web servers. This process is shown in Figure 6.6. (1) The traffic generators sends http requests to the node. (2) The node forwards the request to the server. (3) The node forwards the response back to the requester. (4) The performance is then benchmarked.

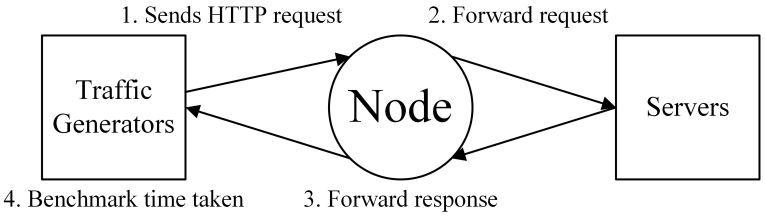

Figure 6.6: Benchmarking a node

6.3.3 Servers

Due to the large traffic volume, we have to provide sufficient capacity for the web servers. We prepared 4 servers running NGINX on top of CentOS 6.x. NGINX uses asynchronous event-driven approach to service incoming requests. Apache HTTP Server uses a threaded/process oriented design. It has been shown that NGINX can provide more predictable performance under high loads [2]. Furthermore NGINX has also shown to have a small memory footprint for each simultaneous connection. Furthermore NGINX's wide adoption and hosting of nearly 22.2M active sites such as Wordpress.com, GitHub, Netflix, Hulu and CloudFlare making it a suitable candidate for the following tests.

6.4 Reinforcement, Resistance and Flows

With 44 computers, we have the following setup as shown in Figure 6.3. Figure 6.7 shows a control experiment demonstrating a flash crowd effect on the network. After $t = 50s$, the traffic generators ramp up requests to simulate an attack. The

increase in traffic is evident in all 3 layers. To gather statistics about the network, we deploy additional probes[3] to sample the availability and utilization of the nodes.

After establishing a control, we demonstrate reinforcement discussed in Section 4.3 in Figure 6.8. We can observe that as the load increases, nodes from Layer 2 and 3 move to 1. When this occurs, load in Layer 1 reduces. Figure 6.9 is used to show auto resistance described in Section 4.6. We set a threshold for a source-destination pair. As the attack traffic increases, nodes start to auto-blacklist malicious clients by notifying the originating node. However, load at Layer 1 is not reduced, as it does not have a mechanism to stop malicious traffic from outside the network.

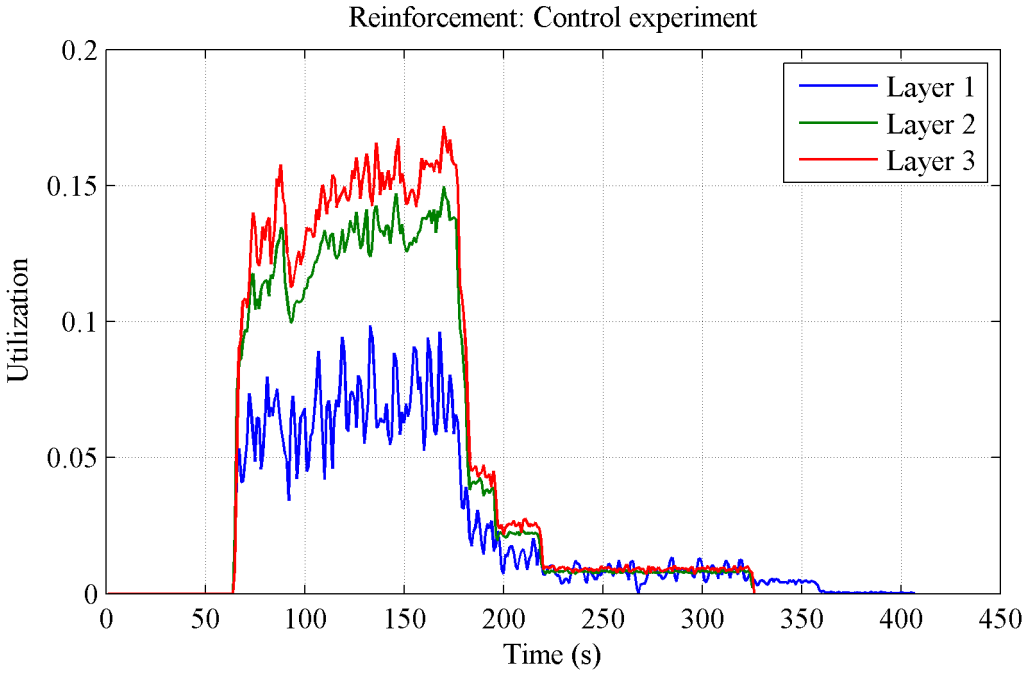

Figure 6.7: Reinforcement: Control experiment

The following Figure 6.10 and 6.11 shows the load perceived at all the layers. With auto-structure and auto-optimization strategies we were able to reduce deviation of loads among the layers. With approximately 1.5 million requests. Results are shown in Table 6.1.

Hence, the above results demonstrate how reinforcement and flow resistance can optimize and reduce loads in the network. Overhead over the implementation

[3]Layer 1 - +7, Layer 2 - +3, Layer 3 - +2).

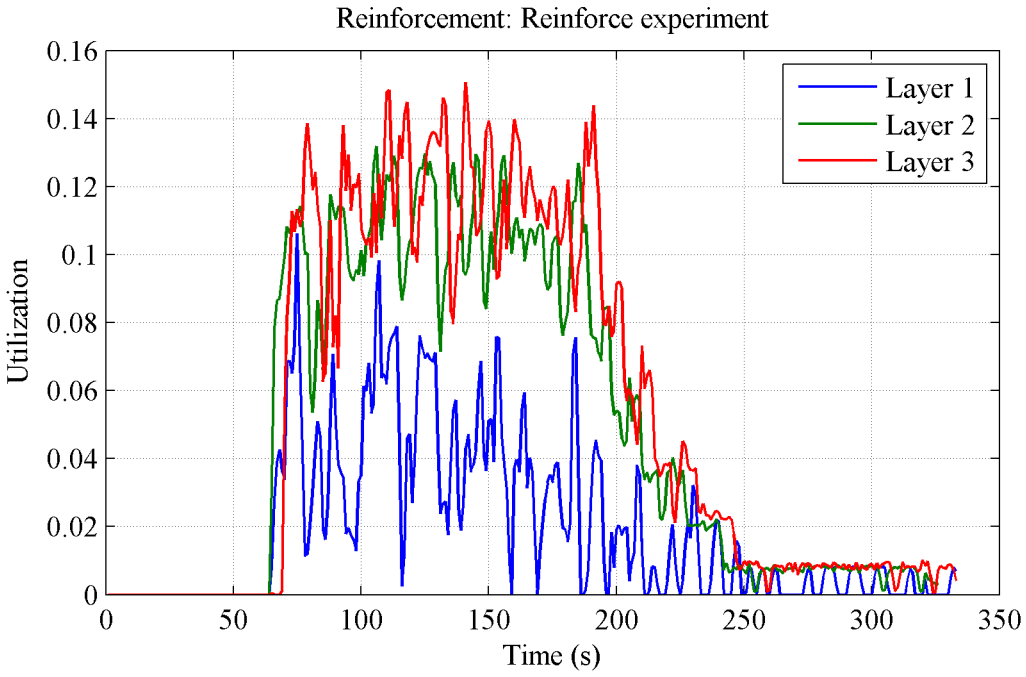

Figure 6.8: Reinforcement: Layer 2 and 3 reinforcing Layer 1

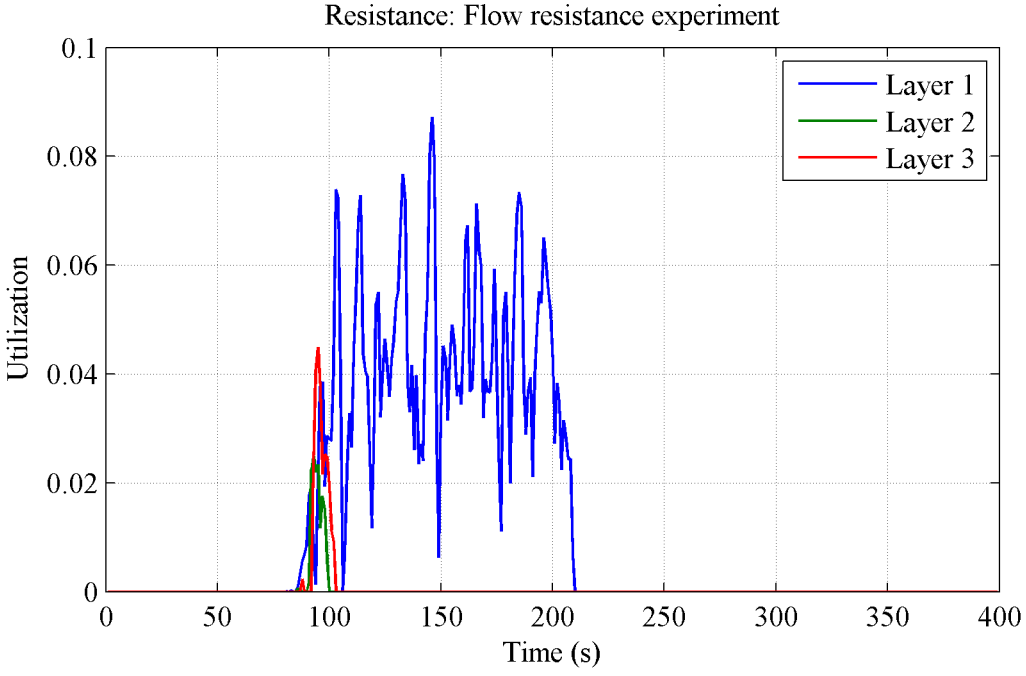

Figure 6.9: Resistance: Flow resistance

Table 6.1: Route performance

Strategies	Avg Transactions/s	Avg Speed (Mbps)
Control, SFQ	1519.05	39.253
Improvements	1624.85	41.99

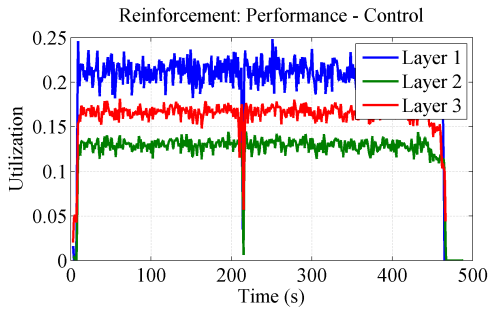

Figure 6.10: Reinforcement: Performance - Control

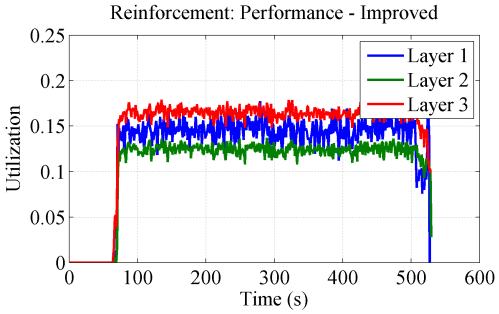

Figure 6.11: Reinforcement: Performance - Improved

follows those described earlier in sub section 4.3.2.

6.5 Flow Optimizations

To demonstrate how various configurations and flow strategies affect throughput, a micro perspective is shown below.

6.5.1 Control Scenarios

The following Figure 6.12 shows a simple 3-hop overlay network. We chose 3 nodes that have at least a performance of $k \geq 2500$. K values are an estimate of a node's average performance obtained in earlier benchmarks. There are other factors that influence the eventual throughput of the network.

To show the impact of involving a lower performing node, the following configuration in Figure 6.13 is used. We can increase the performance of the network by adding more nodes to 2nd layer as illustrated in Figure 6.14. Switching Node 2.2 for a higher performing node increases throughput as shown in Figure 6.15.

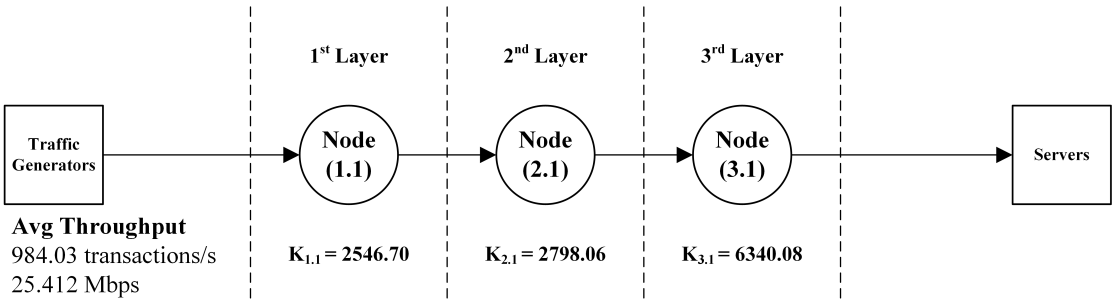

Figure 6.12: Control experiment (1) without bottleneck

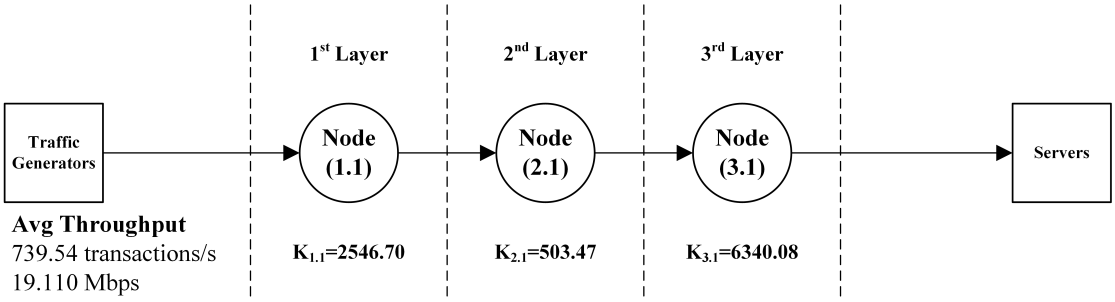

Figure 6.13: Control experiment (2) with bottleneck

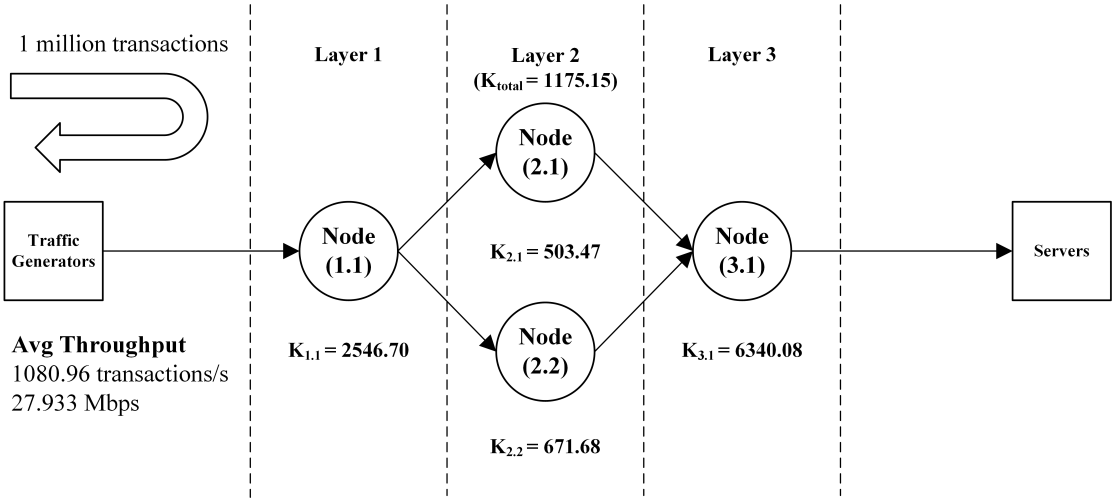

Figure 6.14: Control experiment (3) increasing nodes (2nd layer)

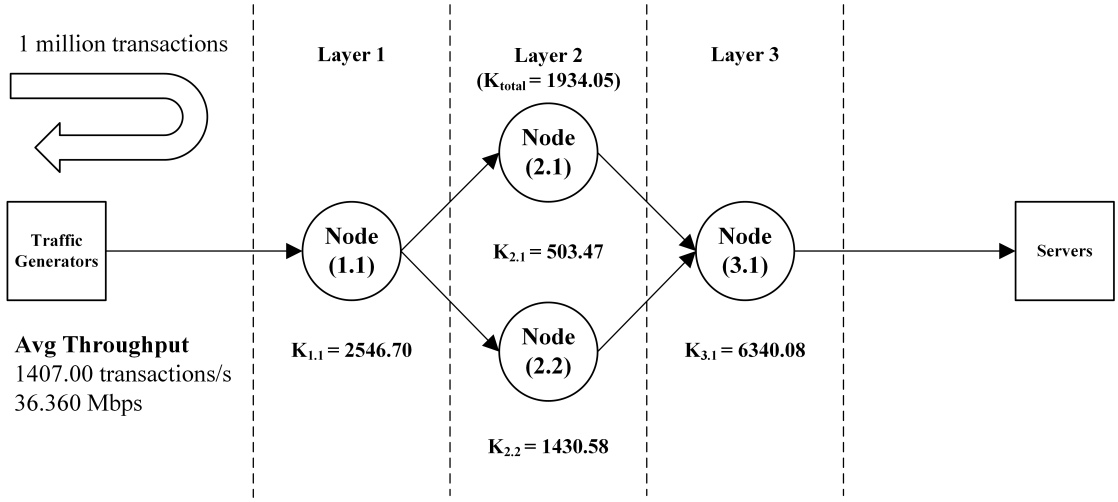

Figure 6.15: Control experiment (4) increasing performance (2nd layer)

More nodes can be added to both Layer 1 and Layer 2 and network throughput increases as shown in Figure 6.16 and 6.17.

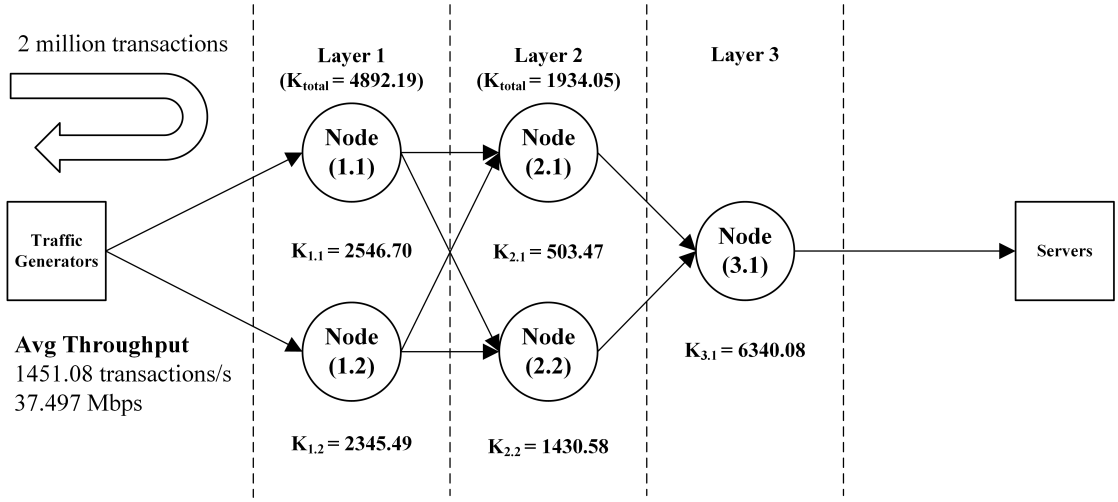

Figure 6.16: Control experiment (5) increasing nodes (1st layer)

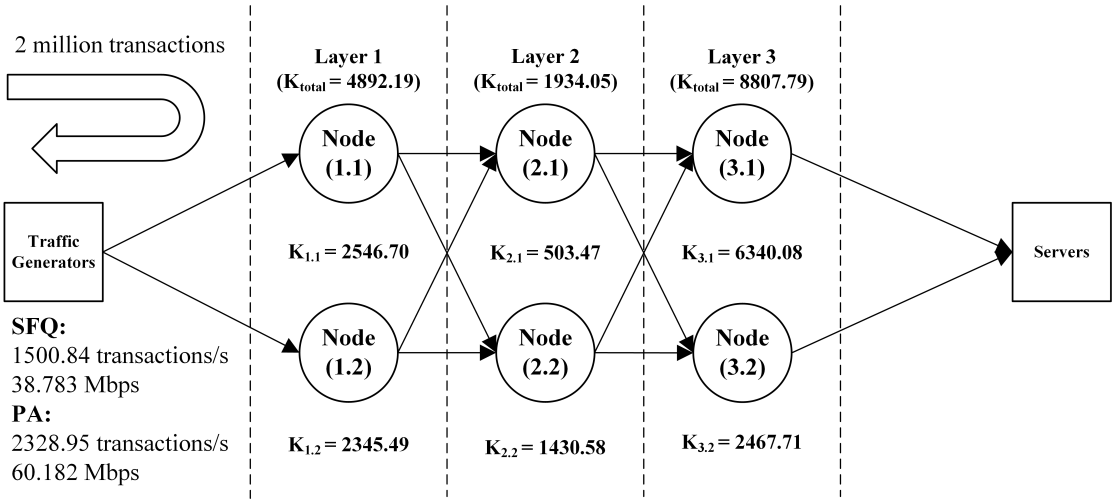

Figure 6.17: Control experiment (6) increasing nodes (3rd layer)

Table 6.2: Summary of control experiments

Topo	**Descriptions**	**Avg Trans/s**	**Avg Spd (Mbps)**
1-1-1	Without bottleneck	984.03	25.412
1-1-1	With bottleneck	739.54	19.110
1-2-1	Increasing nodes (2nd layer)	1080.96	27.933
1-2-1	Increasing performance (2nd layer)	1407.00	36.360
2-2-1	Increasing nodes (1st layer)	1451.08	37.497
2-2-2	Increasing nodes (3rd layer) - SFQ	1500.84	38.783
2-2-2	Increasing nodes (3rd layer) - PA	2328.95	60.182

6.5.2 Improving flows

With the following configuration shown in Figure 6.18 we evaluate how different route selection algorithms affect network throughput. Node 1.1 is connected to Node 2.1 and 2.2. While Node 1.2 is connected to 2.2 and 2.3.

From the previous section, we have shown that proportional allocation (PA) results in a higher throughput then stochastic fair queuing (SFQ). SFQ performs poorer because it does not take varying performance and capacities into consideration, hence does not prevent bottlenecks. However, PA's performance is best when it has sufficient knowledge of its peer nodes. This has been elaborated in Section 5.6. Hence, the following experiments show the benefits of using variants of water flow algorithm against proportional allocation.

PA is uses $P(x = y) = \frac{\frac{1}{u(t)_y}}{\sum_{i=1}^{n} \frac{1}{u_i(t)}}$ described in Section 5.5.1.

SWM route selection uses $P_i^{WF}(x = j) = \frac{f(soil(i,j))}{\sum_{k \forall n_i} f(soil(i,k))}$ described in Section 5.6.3.

Table 6.3 shows the results obtained from multiple runs. Figure 6.19 shows a comparison of various $|d_v|$ values.

Table 6.3: Flow optimizations: Searching for best solution

Strategies	**Avg Transactions/s**	**Avg Speed (Mbps)**	**Remarks**
PA	1911.62	49.398	0.490,0.95,0.468
SW(1)	1903.24	49.181	Bg=12k
SW(2)	1924.61	49.733	$\|d_v\| = 1.0$
SW(3)	1947.79	50.332	$\|d_v\| = 1.2$
SW(4)	1957.48	50.583	$\|d_v\| = 1.5$
SW(4)	1953.52	50.480	$\|d_v\| = 1.8$

$|d_v|$ adjusts the rate of accumulation of soil as described in $soil_{(i,j)}(t+1) = \big(e_v \times soil_{(i,j)}(t)\big) + \frac{a_s}{b_s + (c_s \times f(v_{(i,j)},t)^{d_v})}$. Figure 6.19 shows how throughput varies with $|d_v|$.

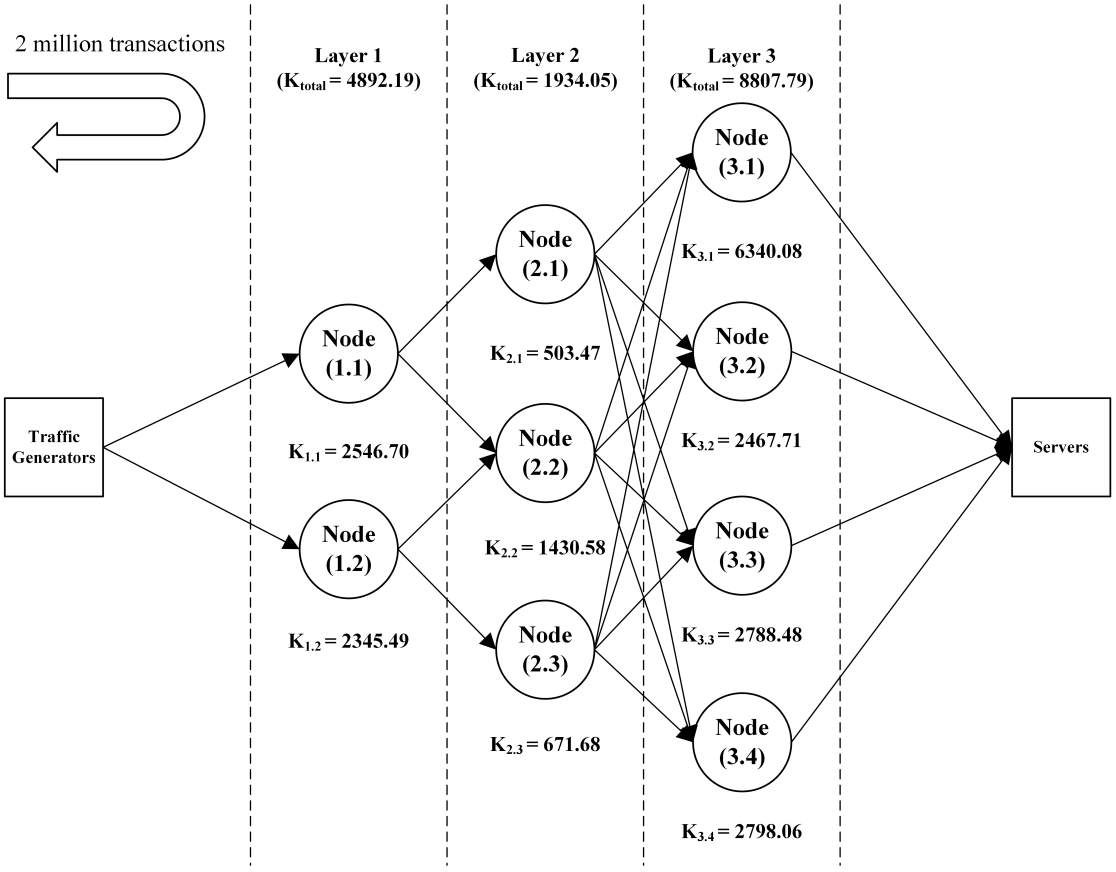

Figure 6.18: Flow optimizations: Setup

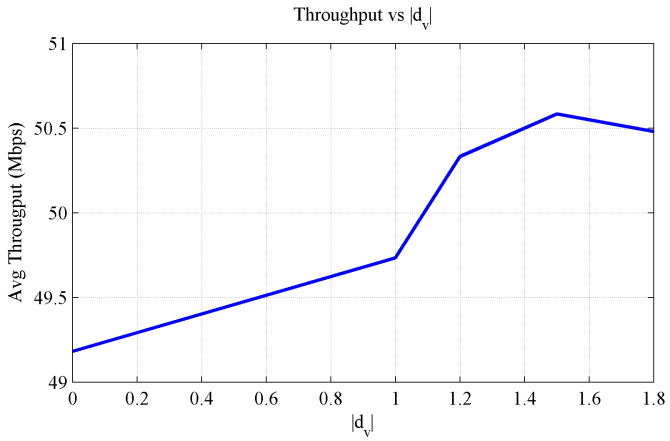

Figure 6.19: Throughput vs d_V

6.5.3 K decisions and Auto Sleeping

The following experiments show how k-iterative sampling and sleep wake can be used to achieve a target service/load level. Figure 6.20 shows the network topology used for the subsequent experiments.

Figures 6.23 to 6.28 show how the network can maintain a target service level. Figures 6.21 and 6.22 show a control experiment, where no auto sleep takes place. Figure 6.23 and 6.24 show a service level of 0.25 to 0.75. Figure 6.25 and 6.26 show a load threshold at 0.45 to 0.55. Figure 6.27 and 6.28 show a load threshold of 0.7-0.9. From all these figures we can observe that the network is able to maintain a pre-set threshold.

For example, 0.25 - 0.75 indicates that the target load will remain between 25% and 75%. When the network detects that the load is below 25%, nodes will be slept until load increases above 75%. Table 6.4 shows network throughput over various sleep/wake thresholds. Throughput remains fairly similar, hence showing that sleeping nodes within constraints results in efficient use of available resources.

Table 6.4: Flow optimizations: Sleep/Wake protocol

Runs	**Avg Transactions/s**	**Avg Speed (Mbps)**	**Remarks**
1	2104.95	54.393	Control
2	2110.38	54.541	0.25 - 0.75
3	2147.52	55.857	0.45 - 0.55
4	2029.15	36.810	0.7 - 0.9

6.6 Summary

In this chapter, we have shown our software approach for the prototype. We then deploy up to 44 systems to demonstrate features we discussed in Chapters 4 and 5. We show how reinforcement and auto resistance can optimize and reduce flooding in the network. We then show how various flow optimization strategies have an impact on overall network throughput. Finally, we show that using k-sensing described in Section 4.4 to decide if a node should sleep. We show that

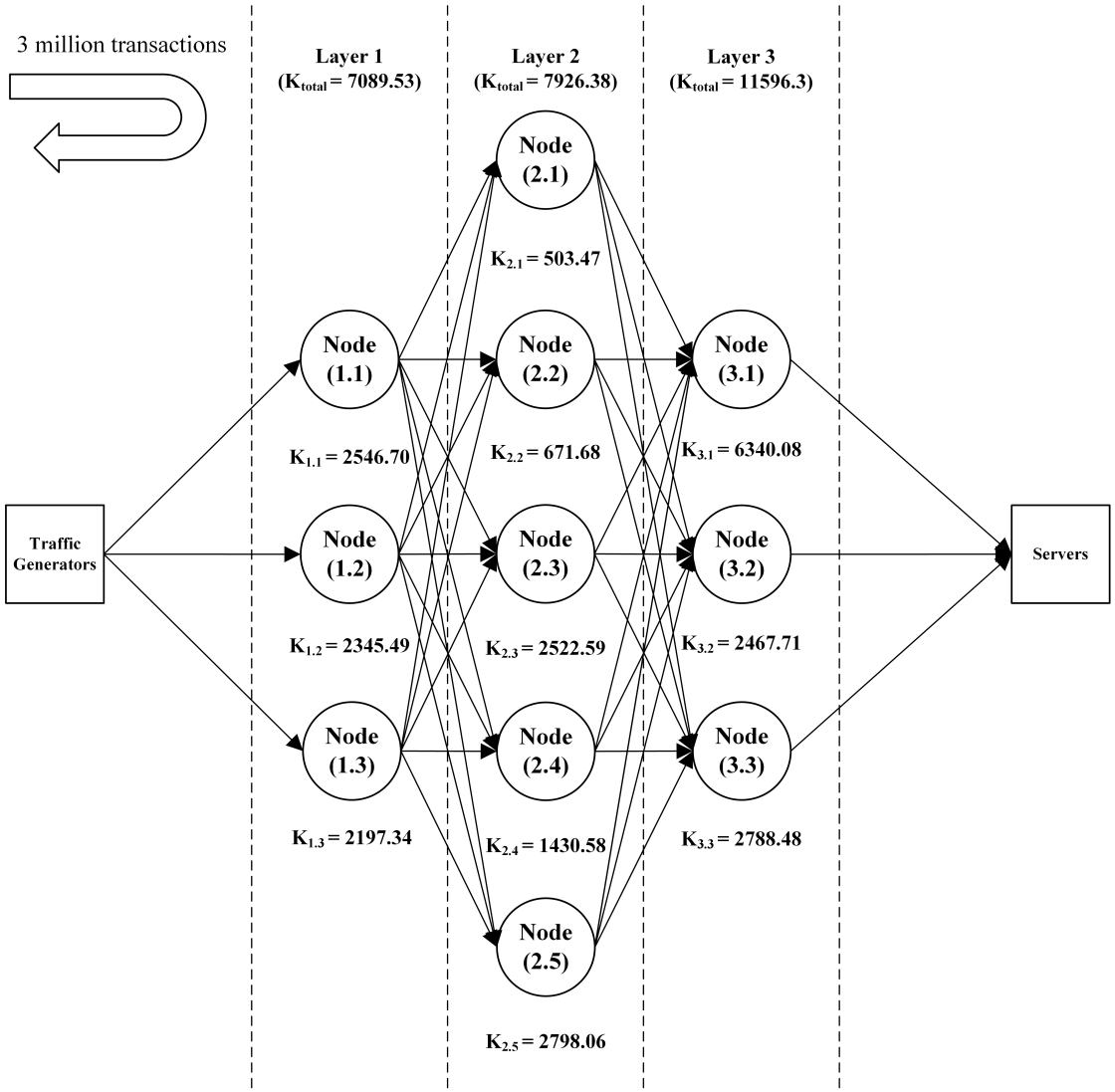

Figure 6.20: Network topologies for K-Decisions

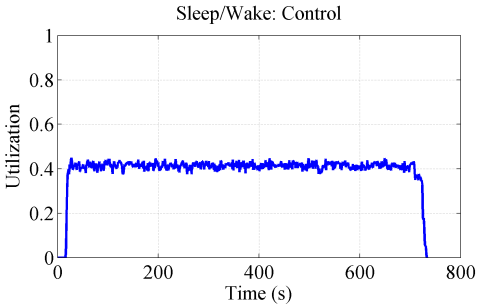

Figure 6.21: Sleep/Wake:
Control
Layer average

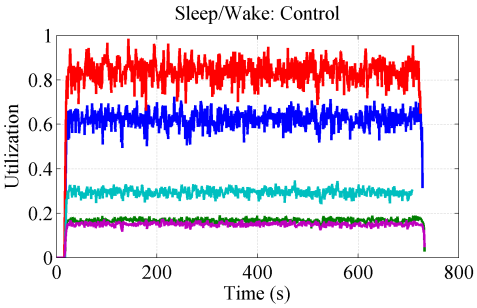

Figure 6.22: Sleep/Wake:
Control
Individual nodes

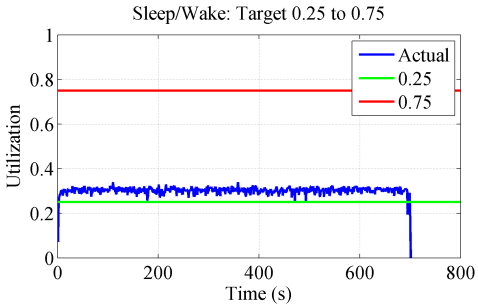

Figure 6.23: Sleep/Wake:
Target 0.25 to 0.75
Layer average

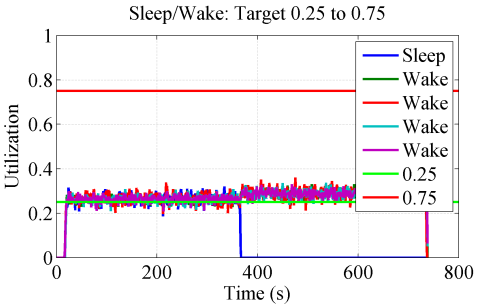

Figure 6.24: Sleep/Wake:
Target 0.25 to 0.75
Individual nodes

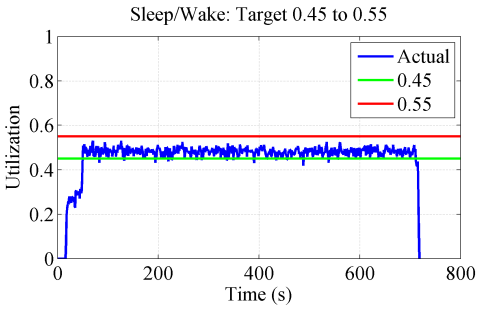

Figure 6.25: Sleep/Wake:
Target 0.45 to 0.55
Layer average

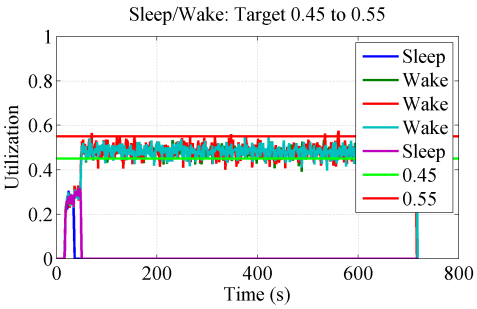

Figure 6.26: Sleep/Wake:
Target 0.45 to 0.55
Individual nodes

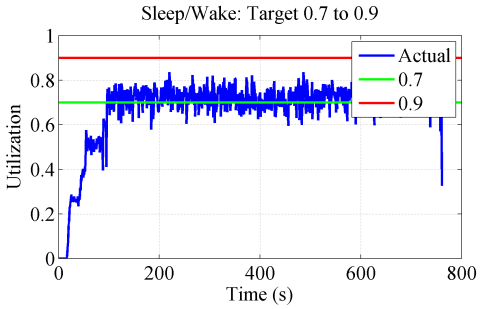

Figure 6.27: Sleep/Wake: Target 0.7 to 0.9 Layer average

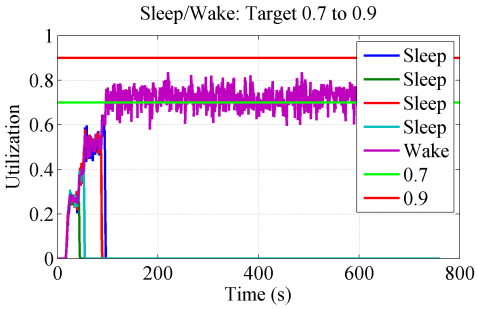

Figure 6.28: Sleep/Wake: Target 0.7 to 0.9 Individual nodes

with sleep/wake we are able to maintain a target utilization in a layer.

Chapter 7

Conclusions and future work

7.1 Discussions

Network filter

Among various overlay networks, our network structure is related to CLAD introduced by Du et al [27]. CLAD's network defence can be illustrated in Figure 7.1.

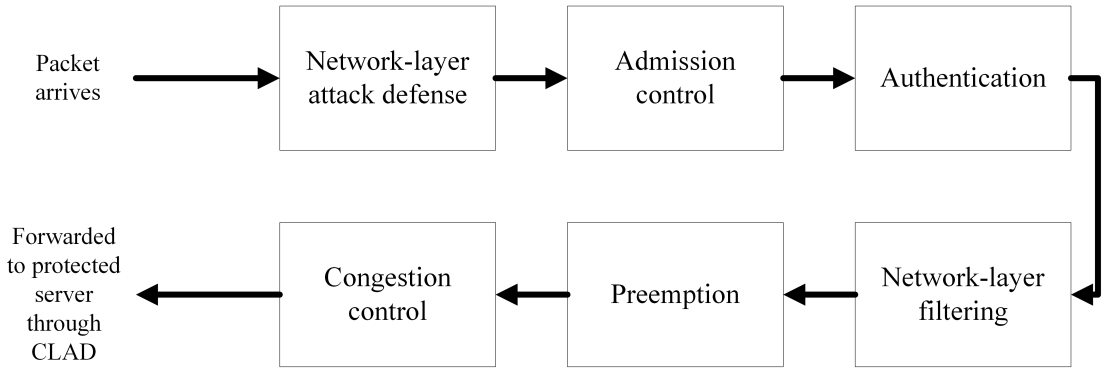

Figure 7.1: CLAD Architecture, Du et al

Cloud versus Crowd Du et al proposed deploying CLAD on a completely controlled virtual machines such as Amazon EC2 or Google's GAE. CLAD assumes that cloud computing is powerful enough to handle exhaustion attacks. However our work targets disparate volunteer nodes. Hence, we do not have a centralized,

or standardized platform to deploy nodes. Deploying our DDoS defence network on cloud computing infrastructures is possible but vulnerable to economic DDoS attacks. While Amazon and Google's infrastructure can scale to cope with an incoming DDoS attack, the cost of mitigation also sky-rockets. An observant attacker would then make use of that to exhaust credits and eventually put it out of service. CLAD's congestion control is on the tail end of the overlay service, and does not address the costs incurred from filtering incoming attacks. Our solution targets volunteer nodes from the crowd. We proposed that a distributed attack could be solved by crowd sourcing. Our work also goes beyond CLAD and specifies the type of service provided for each domain. Hence computers performing network-level (SYN, UDP flooding) or application level (E.g. SQL database connection) on a HTTP web server will be filtered at the gateway nodes.

Multiplexing Our approach is designed with multiplexing in mind. Unlike preceding designs, we protect multiple services simultaneously. By maintaining a service to end-point table, we route traffic for a group of domain names towards separate servers. Depending on the requests, traffic will be routed to its endpoint via a virtual circuit. This addresses our ideals of computing insurance, whereby cost of protection is reduced with larger number of participants. Hence our system will benefit from economies from scale as it grows.

Detection and authentication CLAD service assumes that detection mechanisms are in place to provide a white/black list. To further distinguish users from automated queries, graphical Turing tests are used. In our work, we explore an unsupervised method of deploying detectors affected by scattering effects of load balancing. We proposed using an inexpensive trigger to determine optimal number of detectors deployed. However, we only performed simple filtering based on request ratios and rates with same source-destination. We propose adding waiting room and puzzles to provide an intermediate filter against illegitimate traffic.

Fast flux services While previous works hinted on the use of Domain Name Services to direct clients to entry nodes, we extend this to perform dynamic allocations. We bring together fast-flux and overlay networks. Fast-flux networks are

commonly used for content hosting, we extend them to redirect client requests. As traffic flow increases, we scale the number of entry nodes accordingly. We also make use of our FDNS/Seed to demonstrate an iterative, self-configuring mechanism to allow incoming nodes to grow the network. We also focus on the flux of gateway nodes, a large number will prevent Botnets from gathering enough information to take down majority of gateway nodes simultaneously. As nodes are reassigned roles, address of gateway nodes changes frequently to avoid exposures.

Network-graphs

Selfish Overlay Network, SON-EGOIST [103] by Smaragdakis et al explores the formation of overlay topologies is similar to ours.

Network distance and cost metrics In this study we made use of (1) utilization, (2) throughput and (3) round-trip-time, `RTT`, `ping` as measures of cost. Similar to SON, we also propose using `ping` to measure delays. SON-EGOIST also uses Pyxdia as a virtual coordinate system as part of their work on PlanetLab's dataset. In contrast, we use a simple two-dimensional node addressing scheme[1].

Delay incurred is made up of both transmission and processing delay. SON-EGOIST uses the same cost for all outgoing links. Hence they only make use of local measurements of CPU load. We extend performance estimates to benchmarking links between nodes. This encompasses delays caused by (1) transmission, (2) processing and (3) background chatter. Utilization can be measure as instantaneous throughput over maximum throughput.

Network-Graphs and Neighbour selection SON-EGOIST explores the choice of direct neighbour size (n) and how adopting selfish neighbour selection will result in various pay-offs. The performance pay-off varies greatly when n is small and becomes less significant when it increases. SON-EGOIST shows how connecting to (1) random (2) regular and (3) closest graph influences performance. Connecting a random graph randomly yields the worse performance as the underlying graph is poorly optimized. However, connecting to a poorly optimized graph by select-

[1]Layer and Node ID

ing nearest neighbours gives good results. This is because it reduces registration time. When connecting to a network graph which was formed using 'Best response' (BR) the performance improves as for random or 'closest' connections. However connecting to the best node does not improve performance for mature regular (BR) graphs as there are lesser short cuts. Connecting to the closest neighbour in a regular (BR) graph gives good performance as it minimizes cost while leverage on optimized graph. However, if all nodes chooses its closest neighbour, performance will be poor. It is then observed that it 'pays to cheat' whereby a regular (BR) connection scheme should be used by others while a node exploits them by performing a closest neighbour selection

Our work uses a homogeneous distribution of nodes. Hence our work approximates randomly connecting to a regular (BR) network graph. With the nodes spread equally, 'closest' neighbour will not be any different from 'random'.

7.2 Future work

Network integrity and reliability In the scope of this work, we do not address node or network protection. Threats to P2P and overlay networks will affect our current system. Threats such as eclipse attacks [102] can be used to mislead legitimate nodes to adopt illegitimate ones. Successful eclipse attacks will isolate and cut off vital communications between nodes resulting in an eventual collapse. Multiple bogus registrations with the FDNS/seed node will accelerate the proliferation of illegitimate nodes. Control messages could also be poisoned to redirect traffic in-proportionally, resulting in congestion to take place.

The above effects could be reduced if some form of transmission guarantee is available. Our current set-up uses symmetric communication paths. A response to a request will be relayed through the same nodes for each transaction. If any node involved in this transaction failed, the request will time-out. An unresponsive node will then be removed from future use. While subsequent requests will be successful, the initial request will fail. Hence, if the node flux is too great, we will still experience a large number of failed requests.

We can address this by implementing a form of guarantee. We can enforce that if a timely response is not received, the request is repeated. This can come

from the (1) gateway node which is maintaining a persistent connection to the client or (2) each node along the transaction. The latter will be much more costly, but recovery will be faster. Spectrum-type communications can also be used to maximize chances of success [115]. Eclipse attacks can be addressed by (1) stronger structural constraints used by CAN and Pastry using a constrained routing table [20, 92]. (2) Proximity constraints [50] can also be used assuming that delay measurements cannot be manipulated by attackers.

Overlay topologies Various overlay network topologies such as Full Mesh (FM), K-Minimum Spanning Tree (KMST), Mesh Tree (MT), Adjacent-Connection (AC) and K-Shortest Path Tree (KSPT) are discussed. In a Full Mesh, all nodes are adjacent to all other nodes. Minimum spanning tree is the lowest cost tree. KMST [127] consists of K MSTs, where the k-th MST of the composite graph is the MST of the initial graph, excluding edges of previous MSTs. Hence the links of K trees are not overlapped. A Mesh Tree [66] is obtained by joining MSTs connecting all grandchild-grandparent or uncle-nephew relationship in a MST. AC [66] topology connects in a manner that reflects the IP-layer path between any overlay nodes. KSPT [5] is made up of minimum cost paths from a node towards all other nodes. The following metrics were used by Adami et al.

- **Bandwidth Rejection Ratio, BRR** - Probability an overlay path with bandwidth and delay requirements can not be created due to bandwidth unavailability. $\text{BRR} = \frac{\text{bandwidth rejected paths}}{\text{total number of paths}}$
- **Delay Rejection Ratio, DRR** - Probability that an path with bandwidth and delay requirements cannot be created due to delay limitations, although bandwidth constraint could be satisfied. $\text{DRR} = \frac{\text{delay rejected paths}}{\text{total number of paths}}$
- **Accepted Traffic Weighted Delay, ATWD** - Capability of topology to associate low delays to highest traffic demands.
  $\text{ATWD} = \frac{\sum_{ij}(\text{accepted traffic between } i,j) \times delay_{ij}}{\text{total accepted traffic}}$

Results from [5] have shown that MT performs worse than the other topologies. In a flat IP-layer network, the overlay topology should be chosen based on the

number of overlay nodes. If the quantity of both overlay and IP nodes is small, KMST is the best. If overlay nodes is small, but IP layer size is large, the best topology is KSPT. When number of overlay nodes is large, AC is the best.

Our work is similar to KSPT. However as we increase the number of nodes, AC is expected to be a better solution. As we expand the network, the IP-layer which the nodes would operate from would be hierarchical. In this scenario, AC topology outperforms the others because it benefits from understanding underlying physical topology. Hence, this has to be addressed for future scalability.

Routing schemes In our work, we did not take into spatial information into consideration. Neighbours are randomly assigned as the focus is on self-configuration and optimization. Routing schemes such as (1) Key-based routing, (2) Decentralized object location and routing and (3) Distributed hash tables are some approaches used. Future attempts in implementation has to incorporate any existing or new routing scheme.

Node protection To ensure safety of the network, we should implement a form of authentication scheme to prevent malicious eclipse or poisoning attacks. This could either make use of shared keys that are distributed from a key manager or involve a trust or reputation system to ensure legitimacy of members.

Waiting room and puzzles During a flash crowd event, servers at the endpoints may become overloaded. This occurs regardless of whichever congestion control is in place to shape traffic. In this occasion, we suggest building waiting rooms to engage users and provide a graceful method of queuing for resources. Waiting room content can be responded from the relay nodes themselves without reaching a congested server. These rooms will advise users of congestion at the server and designate a wait-time before request is repeated. This prevent servers from any further overloads. Puzzles described in Section 2.2.1 can also be implemented to throttle incoming flows. Graphical Turing tests can be used to determine if a request comes from a human or bot. Network puzzles can be used to assure that every request has a corresponding 'commitment' to avoid excessive attempts.

Better detectors While we have established an approach to deploy traffic monitors and scrubbers, our detection remains simple. A large variety of detection strategies are described in Section 2.3.1. These can be used to formulate a more efficient and effective detector.

Accelerating response with Content replication With our overlay network, we could also explore replication of static content to reduce traffic volume. Currently, every response has to be transmitted in full across all hops. Static content can be cached among some relay nodes. Upon request, these content can be delivered quickly without retrieving them from the server. These content could also have a `TTL` value to indicate 'liveness' of the content. Expired content will then be re-fetched from the server. This is briefly illustrated in Figure 7.2.

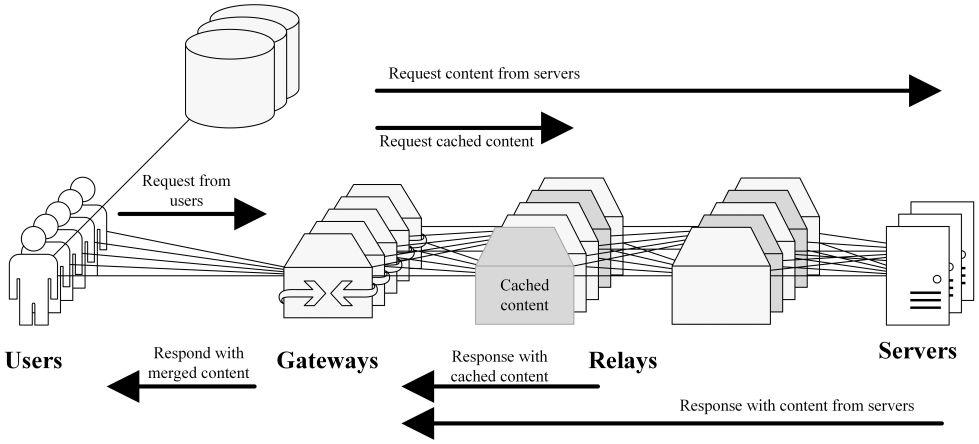

Figure 7.2: Cached content in ASN

We then have to select which content to be replicated. The amount of content cached will also be affected by available memory and availability of the nodes. We could also extent this to store a large percentage of a website's content within the swarm. This allows us to reconstruct a web page even though the originating endpoint server becomes unavailable.

Replication and individual node memory To extend the capability of our Autonomous Swarm Network to self-optimize its own parameters, we need to have an mechanism to store historical and state information. Our current approach in

optimization of flows involves a short-term memory of flow state and consumption. At each sampling interval, some parameters are reset to new values. To allow a better discovery process and use of previously discovered optimal parameters (e.g. neighbour size, water flow parameters etc.), we need to store them within the individual nodes.

However, nodes used are inherently unreliable. We propose mirroring or replicating historical information among nodes within a cluster. When a new node enters the network, it will retrieve a record of parameters used and fitness index (e.g. throughput) observed. This will be used as starting values that will be further optimized.

7.3 Final thoughts

Overall

We have proposed a distributed and autonomic system that exhibits anti-fragile traits. To reduce risk and cascade failures, we have designed a decentralized overlay network. This avoids any single point of failure from collapsing the entire system. An iterative structure allows us to re compose roles and functions as required. We keep functions of each node small to avoid over reliance on any particular node. We abstract computer resources with roles and functions. This allows us to fulfil any identified performance shortfall within the network. For example, when a certain pool or role becomes overloaded, we can reinforce by reallocating more nodes (resources).

Autonomous Swarm Networks

To adapt to uncertainties and model error, we propose developing adaptive feedback control systems. This leads us to the design of an autonomous system.

Fast flux session binding provides a mechanism for us to build the network iteratively. It also allows us to dynamically re-configure the allocation of gateway nodes. We perform these re-allocation by modifying both the return response of domain name resolutions and its Time-To-Live parameter. This mechanism is a

transparent add on to existing web services and it does not require any modification.

We extend this to allow us to automatically build up network structures. If a node tries to query for a DNS node but it does not receive a timely response, it will promote itself to take on the roles. Each node also maintains a neighbour list which is shared periodically with other nodes. This increases robustness of the network and avoid nodes from becoming unreachable and isolated. Having a continuous list of peers to choose from also allows us to find better solutions over time.

We developed an iterative sampling approach to estimate global traffic to form a feedback loop. Sampling a large population of nodes can be both exhaustive and time consuming if performed sequentially. We determine the higher order estimates by aggregating traffic matrices. For example, at the lowest level, $k = 0$, we aggregate traffic of neighbouring nodes. The next level $k = 1$, we then aggregate neighbouring $k = 0$ estimates. This can be performed iteratively to estimate global traffic for large networks. With these estimates, we can then use them to sleep/wake nodes to achieve certain level of efficiency.

We optimize route selection by proposing a Waterflow algorithm. We select the lowest cost links and model consumption of such links. Modelling consumption of links avoid over-taxation of certain nodes and routes. This further optimizes the overall network throughput. As traffic flows through the network from various sources, we proposed an auto detection and deployment technique to allocate traffic monitors and scrubbers. Detectors are deployed initially near end point servers to benefit from traffic aggregation. Load balancing results in scattering of malicious traffic. Aggregation allows detectors to have increased sensitivity to malicious requests. As the traffic flow increases, nodes nearer to endpoints will become overloaded. Before this occurs, an inexpensive trigger is used to recruit upstream nodes.

Prototyping

These features were implemented in a prototype and tested. We recruited up to 50 participants and 80 systems to join the network. Through the field test, we were

able to demonstrate the following.

- **Auto structure** - Self-configuration of nodes to form an iterative network.
- **Sensing** - Using our k-iterative sampling approach, we are able to estimate global traffic loads. This allows us to adapt the network to various traffic conditions.
- **Reinforcement** - When encountering high traffic flows, nodes can be reallocated to the entry layer to reduce layer utilization
- **Resistance** - When a threshold for a source-destination pair is exceeded, a blacklist is propagated upstream to filter malicious traffic.
- **Flow optimization** - Using our Waterflow algorithm, we are able to tune the network to have higher throughput
- **Sleep/Wake** - With a sense of global traffic loads, we can sleep or wake nodes as required to maintain a predefined quality of service.

Closing

Our work attempts to illustrate how autonomic and swarm computing can come together to form an overlay network. Its resilience against flash crowds and DDoS is enhanced by its ability to adapt to changing network conditions. An iterative structure allows nodes to reform the network in many ways avoiding over reliance on any particular node and role. Using inexpensive sensing and triggers, we are able to achieve desired effects on the larger network.

References

[1] Domain names - implementation and specification. URL `http://www.ietf.org/rfc/rfc1035.txt`.

[2] Basic nginx configuration. URL `https://library.linode.com/web-servers/nginx/configuration/basic`.

[3] Dns support for load balancing. URL `http://tools.ietf.org/html/rfc1794`.

[4] Alert (ta-017a) udp-based amplification attacks. URL `http://www.us-cert.gov/ncas/alerts/TA14-017A`.

[5] Davide Adami, Christian Callegari, Stefano Giordano, Gianfranco Nencioni, and Michele Pagano. Design and performance evaluation of service overlay networks topologies. In *Performance Evaluation of Computer & Telecommunication Systems, 2009. SPECTS 2009. International Symposium on*, volume 41, pages 296–303. IEEE, 2009.

[6] Mohammed Alenezi and Martin J Reed. Ip traceback methodologies. In *Computer Science and Electronic Engineering Conference (CEEC), 2011 3rd*, pages 98–102. IEEE, 2011.

[7] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. *ACM SIGCOMM Computer Communication Review*, 32(1):66–66, 2002.

[8] David G Andersen et al. Mayday: Distributed filtering for internet services. In *USENIX Symposium on Internet Technologies and Systems*, pages 20–30, 2003.

[9] Spyros Antonatos, Kostas G Anagnostakis, Evangelos P Markatos, and Michalis Polychronakis. Performance analysis of content matching intrusion detection systems. In *Applications and the Internet, 2004. Proceedings. 2004 International Symposium on*, pages 208–215. IEEE, 2004.

[10] Xuhua Bao and Hai Hong. Nsfocus ddos threat report 2013. URL `http://en.nsfocus.com/SecurityReport/NSFOCUS%20DDoS%20Threat%20Report%202013.pdf`.

[11] Martin Beckmann, CB McGuire, and Christopher B Winsten. Studies in the economics of transportation. Technical report, 1956.

[12] Vaclav E Benes. *Mathematical theory of connecting networks and telephone traffic*, volume 68. Academic press New York, 1965.

[13] DP Bertsekas, A Nedic, and A Ozdaglar. Convex analysis and optimization. 2003. *Athena Scientific*.

[14] Jun Bi, Ke Xu, Xing Li, Mark Williams, Jianping Wu, and Gang Ren. A source address validation architecture (sava) testbed and deployment experience. 2008.

[15] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*. Number 1. Oxford university press, 1999.

[16] Robert S Boyer and J Strother Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

[17] Christian Callegari, Michele Pagano, Stefano Giordano, and Teresa Pepe. Combining wavelet analysis and information theory for network anomaly detection. In *Proceedings of the 4th International Symposium on Applied Sciences in Biomedical and Communication Technologies*, page 69. ACM, 2011.

[18] Christian Callegari, Stefano Giordano, Michele Pagano, and Teresa Pepe. Wave-cusum: Improving cusum performance in network anomaly detection by means of wavelet analysis. *Computers & Security*, 31(5):727–735, 2012.

[19] Ross W Callon and Frank Kastenholz. Rate limiting data traffic in a network, June 18 2013. US Patent 8,468,590.

[20] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S Wallach. Secure routing for structured peer-to-peer overlay networks. *ACM SIGOPS Operating Systems Review*, 36(SI):299–314, 2002.

[21] Jeffrey R Cobb, Erik L Eidt, Alan W Lillich, and Wayne N Meretsky. Method and apparatus for patching operating systems, April 8 1997. US Patent 5,619,698.

[22] C Jason Coit, Stuart Staniford, and Joseph McAlerney. Towards faster string matching for intrusion detection or exceeding the speed of snort. In *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings*, volume 1, pages 367–373. IEEE, 2001.

[23] Mads Dam and Karl Palmskog. Location independent routing in process network overlays. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pages 715–724. IEEE, 2014.

[24] Eric M Dashofy, André Van der Hoek, and Richard N Taylor. Towards architecture-based self-healing systems. In *Proceedings of the first workshop on Self-healing systems*, pages 21–26. ACM, 2002.

[25] Colin Dixon, Thomas E Anderson, and Arvind Krishnamurthy. Phalanx: Withstanding multimillion-node botnets. In *NSDI*, volume 8, pages 45–58, 2008.

[26] Thomas W Doeppner, Philip N Klein, and Andrew Koyfman. Using router stamping to identify the source of ip packets. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 184–189. ACM, 2000.

[27] Ping Du and Akihiro Nakao. Ddos defense as a network service. In *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pages 894–897. IEEE, 2010.

[28] Wesley M Eddy. Tcp syn flooding attacks and common mitigations. 2007.

[29] Belson et al. Akamai: state of the internet.

[30] Wu-chang Feng. The case for tcp/ip puzzles. In *ACM SIGCOMM Computer Communication Review*, volume 33, pages 322–327. ACM, 2003.

[31] Wu-chi Feng, E Kaiser, and Antoine Luu. Design and implementation of network puzzles. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 4, pages 2372–2382. IEEE, 2005.

[32] Zebo Feng, Xiaoping Wu, Liangli Ma, and Wei Ren. Establishing the security foundations for network protocol design. In *Communication Technology (ICCT), 2012 IEEE 14th International Conference on*, pages 789–793. IEEE, 2012.

[33] Martin Florian, Fabian Hartmann, and Ingmar Baumgart. A socio-and locality-aware overlay for user-centric networking. In *Computing, Networking and Communications (ICNC), 2014 International Conference on*, pages 327–333. IEEE, 2014.

[34] Nikos Fotiou, Dirk Trossen, and George C Polyzos. Illustrating a publish-subscribe internet architecture. *Telecommunication Systems*, 51(4):233–245, 2012.

[35] Aman Garg and AL Reddy. Mitigation of dos attacks through qos regulation. *Microprocessors and Microsystems*, 28(10):521–530, 2004.

[36] Thomer M Gil and Massimiliano Poletto. Multops: a data-structure for bandwidth attack detection. In *USENIX Security Symposium*, 2001.

[37] Virgil D Gligor. Guaranteeing access in spite of distributed service-flooding attacks. In *Security Protocols*, pages 80–96. Springer, 2005.

[38] Cisco Global. Cisco global cloud index: Forecast and methodology, 20132018. URL `http://www.cisco.com/c/en/us/solutions/collateral/`

service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.pdf.

[39] Chao Gong and Kamil Sarac. Ip traceback based on packet marking and logging. In *Communications, 2005. ICC 2005. 2005 IEEE International Conference on*, volume 2, pages 1043–1047. IEEE, 2005.

[40] Yu Gu, Andrew McCallum, and Don Towsley. Detecting anomalies in network traffic using maximum entropy estimation. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 32–32. USENIX Association, 2005.

[41] SN Gujar, SR Gupta, and MS Ali. Protocol scrubbing: network security through transparent flow modification using active real time database. In *Proceedings of the International Conference and Workshop on Emerging Trends in Technology*, pages 345–350. ACM, 2010.

[42] Ihab Hamadeh and George Kesidis. A taxonomy of internet traceback. *International Journal of Security and Networks*, 1(1):54–61, 2006.

[43] Dan Harkins, Dave Carrel, et al. The internet key exchange (ike). Technical report, RFC 2409, november, 1998.

[44] Debiao He, Jianhua Chen, and Jin Hu. A pairing-free certificateless authenticated key agreement protocol. *International Journal of Communication Systems*, 25(2):221–230, 2012.

[45] Egon Hilgenstieler, Elias P Duarte Jr, Glenn Mansfield-Keeni, and Norio Shiratori. Extensions to the source path isolation engine for precise and efficient log-based ip traceback. *Computers & Security*, 29(4):383–392, 2010.

[46] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C Freiling. Measuring and detecting fast-flux service networks. In *NDSS*, 2008.

[47] Fu-Hau Hsu and Tzi-cker Chiueh. A path information caching and aggregation approach to traffic source identification. In *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, pages 332–339. IEEE, 2003.

[48] Jianwei Huang, Zhu Han, Mung Chiang, and H Vincent Poor. Auction-based resource allocation for cooperative communications. *Selected Areas in Communications, IEEE Journal on*, 26(7):1226–1237, 2008.

[49] Eva Ibarrola, Jin Xiao, Fidel Liberal, and Armando Ferro. Internet qos regulation in future networks: a user-centric approach. *Communications Magazine, IEEE*, 49(10):148–155, 2011.

[50] John Jannotti, David K Gifford, Kirk L Johnson, M Frans Kaashoek, et al. Overcast: reliable multicasting with on overlay network. In *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation-Volume 4*, pages 14–14. USENIX Association, 2000.

[51] Sepandar D Kamvar, Mario T Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web*, pages 640–651. ACM, 2003.

[52] Reyhaneh Karimazad and Ahmad Faraahi. An anomaly-based method for ddos attacks detection using rbf neural networks. In *Proceedings of the International Conference on Network and Electronics Engineering*, pages 16–18, 2011.

[53] Charlie Kaufman. Internet key exchange (ikev2) protocol. 2005.

[54] Frank Kelly. The mathematics of traffic in networks. *The Princeton Companion to Mathematics URL www. statslab. cam. ac. uk/frank/PAPERS/princeton. html*, 2008.

[55] Jeffrey O Kephart. Autonomic computing: the first decade. In *ICAC*, pages 1–2, 2011.

[56] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[57] AD Keromytis, V Misra, and D Rubenstein. Secure overlay services (sos): A critical analysis. In *2 nd IEEE International Conference on Parallel, Distributed and Grid Computing*, pages 457–462, 2012.

[58] Angelos D Keromytis, Vishal Misra, and Dan Rubenstein. Sos: Secure overlay services. *ACM SIGCOMM Computer Communication Review*, 32(4): 61–72, 2002.

[59] Gene H Kim and Eugene H Spafford. Experiences with tripwire: Using integrity checkers for intrusion detection. 1994.

[60] Dmitry Korzun and Andrei Gurtov. Indirection infrastructures. In *Structured Peer-to-Peer Systems*, pages 325–343. Springer, 2013.

[61] Shilpa Lakhina, Sini Joseph, and Bhupendra Verma. Feature reduction using principal component analysis for effective anomaly–based intrusion detection on nsl-kdd. 2010.

[62] Jonathan Lemon et al. Resisting syn flood dos attacks with a syn cache. In *BSDCon*, volume 2002, pages 89–97, 2002.

[63] Bai-nan Li, Dong Yao, and Ye-kui Qian. Incremental principal component analysis method on online network anomaly detection. In *Information and Network Security (ICINS 2013), 2013 International Conference on*, pages 1–6. IET, 2013.

[64] Ming Li. Change trend of averaged hurst parameter of traffic under ddos flood attacks. *Computers & security*, 25(3):213–220, 2006.

[65] Ming Li and Wei Zhao. Representation of a stochastic traffic bound. *Parallel and Distributed Systems, IEEE Transactions on*, 21(9):1368–1372, 2010.

[66] Zhi Li and Prasant Mohapatra. On investigating overlay service topologies. *Computer Networks*, 51(1):54–68, 2007.

[67] Huihui Liang, Jiwen Chai, and Yong Tang. Polymorphic worm detection using position-relation signature. In *Computer Engineering and Networking*, pages 1365–1372. Springer, 2014.

[68] Alan W Lillich. Methods, apparatus, and data structures for data driven computer patches and static analysis of same, August 4 1998. US Patent 5,790,856.

[69] Xin Liu, Xiaowei Yang, David Wetherall, and Thomas Anderson. Efficient and secure source authentication with packet passports. In *USENIX SRUTI*, 2006.

[70] Xin Liu, Ang Li, Xiaowei Yang, and David Wetherall. Passport: Secure and adoptable source authentication. In *NSDI*, volume 8, pages 365–378, 2008.

[71] Yang Liu and Kevin M Passino. Swarm intelligence: Literature overview. *Department of Electrical Engineering, the Ohio State University*, 2000.

[72] Eng Keong Lua and Ruichuan Chen. A holistic immune system against active p2p worms. In *Information Networking (ICOIN), 2013 International Conference on*, pages 24–29. IEEE, 2013.

[73] Ratul Mahajan, Steven M Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. Controlling high bandwidth aggregates in the network. *ACM SIGCOMM Computer Communication Review*, 32(3):62–73, 2002.

[74] G Robert Malan, David Watson, Farnam Jahanian, and Paul Howell. Transport and application protocol scrubbing. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1381–1390. IEEE, 2000.

[75] Mark P Mattson and Edward J Calabrese. *Hormesis: a revolution in biology, toxicology and medicine*. Springer, 2009.

[76] Paul E McKenney. Stochastic fairness queueing. In *INFOCOM'90, Ninth Annual Joint Conference of the IEEE Computer and Communication Societies. The Multiple Facets of Integration. Proceedings, IEEE*, pages 733–740. IEEE, 1990.

[77] Jonathan K Millen. A resource allocation model for denial of service. In *Research in Security and Privacy, 1992. Proceedings., 1992 IEEE Computer Society Symposium on*, pages 137–147. IEEE, 1992.

[78] Jelena Mirkovic and Peter Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[79] Jelena Mirkovic and Peter Reiher. D-ward: a source-end defense against flooding denial-of-service attacks. *Dependable and Secure Computing, IEEE Transactions on*, 2(3):216–232, 2005.

[80] Akihiko Miyoshi and Ragunathan Rajkumar. Protecting resources with resource control lists. In *Real-Time Technology and Applications Symposium, 2001. Proceedings. Seventh IEEE*, pages 85–94. IEEE, 2001.

[81] Neustar and IDG Research Services. Ddos imposes 1m hit on website and e-commerce performance.

[82] George Nychis, Vyas Sekar, David G Andersen, Hyong Kim, and Hui Zhang. An empirical evaluation of entropy-based traffic anomaly detection. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pages 151–156. ACM, 2008.

[83] Peyman Oreizy, Nenad Medvidovic, and Richard N Taylor. Architecture-based runtime software evolution. In *Proceedings of the 20th international conference on Software engineering*, pages 177–186. IEEE Computer Society, 1998.

[84] Peyman Oreizy, Nenad Medvidovic, and Richard N Taylor. Runtime software adaptation: framework, approaches, and styles. In *Companion of the 30th international conference on Software engineering*, pages 899–910. ACM, 2008.

[85] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling tcp throughput: A simple model and its empirical validation. In *ACM SIGCOMM Computer Communication Review*, volume 28, pages 303–314. ACM, 1998.

[86] Kihong Park and Heejo Lee. On the effectiveness of probabilistic packet marking for ip traceback under denial of service attack. In *INFOCOM 2001.*

Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, volume 1, pages 338–347. IEEE, 2001.

[87] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23):2435–2463, 1999.

[88] Christina Popper, Mario Strasser, and Srdjan Capkun. Anti-jamming broadcast communication using uncoordinated spread spectrum techniques. *Selected Areas in Communications, IEEE Journal on*, 28(5):703–715, 2010.

[89] Matthew Prince. Technical details behind a 400gbps ntp amplification ddos attack, . URL `http://blog.cloudflare.com/technical-details-behind-a-400gbps-ntp-amplification-ddos-attack/`.

[90] Matthew Prince. Technical details behind a 400gbps ntp amplification ddos attack, . URL `http://blog.cloudflare.com/the-ddos-that-knocked-spamhaus-offline-and-ho/`.

[91] Jarno Rajahalme, Mikko Särelä, Kari Visala, and Janne Riihijärvi. On name-based inter-domain routing. *Computer Networks*, 55(4):975–986, 2011.

[92] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. *A scalable content-addressable network*, volume 31. ACM, 2001.

[93] J Riden. How fast-flux service networks work, 2008.

[94] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, pages 229–238, 1999.

[95] Usman Asghar Sandhu, Sajjad Haider, Salman Naseer, and Obaid Ullah Ateeb. A survey of intrusion detection & prevention techniques. In *Shaheed Zulfiqar Ali Bhutto Institute of Science and Technology, Islamabad.(SZABIST), University of the Punjab Gujranwala Campus, 2011 International Conference on Information Communication and Management IPCSIT*, volume 16, 2011.

[96] Hamed Shah-Hosseini. Problem solving by intelligent water drops. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 3226–3231. IEEE, 2007.

[97] Anees Shaikh, Renu Tewari, and Mukesh Agrawal. On the effectiveness of dns-based server selection. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1801–1810. IEEE, 2001.

[98] Rao Shen, Wang Yong, and Tao Xiao-ling. The comprehensive trust model in p2p based on improved eigentrust algorithm. In *Measuring Technology and Mechatronics Automation (ICMTMA), 2010 International Conference on*, volume 3, pages 822–825. IEEE, 2010.

[99] Dan Siemon. Queueing in the linux network stack. *Linux Journal*, 2013(231): 2, 2013.

[100] Kwang Mong Sim and Weng Hong Sun. Ant colony optimization for routing and load-balancing: survey and new directions. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 33(5):560–572, 2003.

[101] Ioannis Simaiakis, Hasmsa Balakrishnan, Harshad Khadilkar, Tom G Reynolds, R John Hansman, Brendan Reilly, and Steve Urlass. Demonstration of reduced airport congestion through pushback rate control. 2011.

[102] Atul Singh et al. Eclipse attacks on overlay networks: Threats and defenses. In *In IEEE INFOCOM*. Citeseer, 2006.

[103] Georgios Smaragdakis, Nikolaos Laoutaris, Vassilis Lekakis, Azer Bestavros, John W Byers, and Mema Roussopoulos. Selfish overlay network creation and maintenance. *Networking, IEEE/ACM Transactions on*, 19(6):1624–1637, 2011.

[104] Augustin Soule, Kavé Salamatian, and Nina Taft. Combining filtering and statistical methods for anomaly detection. In *Proceedings of the 5th ACM*

SIGCOMM conference on Internet Measurement, pages 31–31. USENIX Association, 2005.

[105] Oliver Spatscheck and Larry L Peterson. Defending against denial of service attacks in scout. In *OSDI*, volume 99, pages 59–72, 1999.

[106] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. *IEEE/ACM Transactions on Networking (TON)*, 12(2):205–218, 2004.

[107] W Timothy Strayer, Christine E Jones, Fabrice Tchakountio, Alex C Snoeren, Beverly Schwartz, Robert C Clements, Matthew Condell, and Craig Partridge. Traceback of single ip packets using spie. In *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, volume 2, pages 266–270. IEEE, 2003.

[108] Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan, and Randy H Katz. Overqos: offering internet qos using overlays. *ACM SIGCOMM Computer Communication Review*, 33(1):11–16, 2003.

[109] Nassim Nicholas Taleb. A map and simple heuristic to detect fragility, antifragility, and model error, june 4, 2011. *As of December*, 7:165–206, 2011.

[110] Nassim Nicholas Taleb. *Antifragile: things that gain from disorder*. Random House LLC, 2012.

[111] Chee Wei Tan, Dah-Ming Chiu, John CS Lui, and David KY Yau. A distributed throttling approach for handling high bandwidth aggregates. *Parallel and Distributed Systems, IEEE Transactions on*, 18(7):983–995, 2007.

[112] Julian S Taylor. Software patch architecture, December 12 2000. US Patent 6,161,218.

[113] Richard N Taylor, Nenad Medvidovic, and Peyman Oreizy. Architectural styles for runtime software adaptation. In *Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*, pages 171–180. IEEE, 2009.

[114] Roshan Thomas, Brian Mark, Tommy Johnson, and James Croall. Netbouncer: client-legitimacy-based high-performance ddos filtering. In *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, volume 1, pages 14–25. IEEE, 2003.

[115] Don Torrieri. *Principles of spread-spectrum communication systems*. Springer, 2011.

[116] Joe Touch. Dynamic internet overlay deployment and management using the x-bone. *Computer Networks*, 36(2):117–135, 2001.

[117] Lanjia Wang, Zhichun Li, Yan Chen, Zhi Fu, and Xing Li. Thwarting zero-day polymorphic worms with network-level length-based signature generation. *IEEE/ACM Transactions on Networking (TON)*, 18(1):53–66, 2010.

[118] Yongge Wang. Efficient identity-based and authenticated key agreement protocol. *arXiv preprint arXiv:1207.5438*, 2012.

[119] John Glen Wardrop. Road paper. some theoretical aspects of road traffic research. In *ICE Proceedings: Engineering Divisions*, volume 1, pages 325–362. Thomas Telford, 1952.

[120] Brent Waters, Ari Juels, J Alex Halderman, and Edward W Felten. New client puzzle outsourcing techniques for dos resistance. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 246–256. ACM, 2004.

[121] David Watson, Matthew Smart, G Robert Malan, and Farnam Jahanian. Protocol scrubbing: network security through transparent flow modification. *IEEE/ACM Transactions on Networking (TON)*, 12(2):261–273, 2004.

[122] Yang Xiang and Wanlei Zhou. Mark-aided distributed filtering by using neural network for ddos defense. In *GLOBECOM'05: IEEE Global Telecommunications Conference, 28 November-2 December 2005 St. Louis, Missouri, USA, discovery past and future*, pages 1701–1705. IEEE Globecom, 2005.

[123] Ke Xu, Meng Shen, Yong Cui, Mingjiang Ye, and Yifeng Zhong. A model approach to the estimation of peer-to-peer traffic matrices. 2013.

[124] Ming-Hour Yang and Ming-Chien Yang. Riht: a novel hybrid ip traceback scheme. *Information Forensics and Security, IEEE Transactions on*, 7(2): 789–797, 2012.

[125] Xiaowei Yang, David Wetherall, and Thomas Anderson. A dos-limiting network architecture. *ACM SIGCOMM Computer Communication Review*, 35 (4):241–252, 2005.

[126] Yu Yao, Lei Guo, Hao Guo, Ge Yu, Fu-xiang Gao, and Xiao-jun Tong. Pulse quarantine strategy of internet worm propagation: modeling and analysis. *Computers & Electrical Engineering*, 38(5):1047–1061, 2012.

[127] Anthony Young, Jiang Chen, Zheng Ma, Arvind Krishnamurthy, Larry Peterson, and Randolph Y Wang. Overlay mesh construction using interleaved spanning trees. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 1. IEEE, 2004.

[128] Shui Yu, Wanlei Zhou, Song Guo, and Minyi Guo. A dynamical deterministic packet marking scheme for ddos traceback. In *Global Communications Conference (GLOBECOM), 2013 IEEE*, pages 729–734. IEEE, 2013.

[129] Wei Yu, Xun Wang, Prasad Calyam, Dong Xuan, and Wei Zhao. Modeling and detection of camouflaging worm. *Dependable and Secure Computing, IEEE Transactions on*, 8(3):377–390, 2011.

[130] Saman Taghavi Zargar, James Joshi, and David Tipper. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *Communications Surveys & Tutorials, IEEE*, 15(4):2046–2069, 2013.

[131] Lei Zhang, Qianhong Wu, Bo Qin, and Josep Domingo-Ferrer. Provably secure one-round identity-based authenticated asymmetric group key agreement protocol. *Information Sciences*, 181(19):4318–4329, 2011.

[132] Ben Y Zhao, Ling Huang, Jeremy Stribling, Sean C Rhea, Anthony D Joseph, and John D Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *Selected Areas in Communications, IEEE Journal on*, 22(1):41–53, 2004.

[133] Zaihong Zhou, Dongqing Xie, and Wei Xiong. A p2p-based distributed detection scheme against ddos attack. In *Education Technology and Computer Science, 2009. ETCS'09. First International Workshop on*, volume 2, pages 304–309. IEEE, 2009.

[134] Andre Zuquete. Improving the functionality of syn cookies. In *Advanced Communications and Multimedia Security*, pages 57–77. Springer, 2002.

Appendix A

Network variability and Congestion

The results shown in Figure 5.11 is explained by the following. Table A.1 describes a host with 8 peers each with a mean $capacity_{max}$ of 50 units/tokens.

Table A.1: Highest Capacity (Absolute), 8 peers

Node	0	1	2	3	4	5	6	7
$Capacity_{Max}$	50	55	48	52	53	49	60	62
$Capacity_{t=n}$	43	45	46	51	50	44	52	59
Load	0.1400	0.1818	0.0417	0.019	0.057	0.1020	0.1333	0.0484

If transmission volume before each sampling tends toward the maximum capacity, link utilization would fluctuate greatly. Considering a single source of traffic results in the following scenarios. α represents the traffic load as a percentage of maximum capacity. β represents the sustained service (refill) rate of the nodes.

At $t = n$, node 3 change is $+2(+1)$ as the maximum token is capped at 52, hence only 1 token is added. Node 7 is chosen as it has the most tokens compared to others. As shown only Node 6 and 7 are used and the number of tokens do not vary significantly.

As observed in later Tables A.3 A.4 and A.5, as the traffic flow increases, the token count starts to vary significantly.

Table A.2: $\alpha = 0.1(5)$, $\beta = 0.05$

Node	0	1	2	3	4	5	6	7
$Capacity_{Max}$	50	55	48	52	53	49	60	62
$Capacity_{t=n}$	43	45	46	51	50	44	52	59
Load	0.1400	0.1818	0.0417	0.019	0.057	0.1020	0.1333	0.0484
Change	+2	+2	+2	+2(+1)	+2	+2	+2	+2,-5
$Capacity_{t+1}$	45	47	48	52	52	46	54	56
Change	+2	+2	+2(+0)	+2(+0)	+2(+1)	+2	+2	+2,-5
$Capacity_{t+2}$	47	49	48	52	53	48	56	53
Change	+2	+2	+2(+0)	+2(+0)	+2(+0)	+2(+1)	+2,-5	+2
$Capacity_{t+3}$	49	51	48	52	53	49	53	50
Change	+2(+1)	+2	+2(+0)	+2(+0)	+2(+0)	+2(+0)	+2	+2
$Capacity_{t+4}$	50	53	48	52	53	49	55	52

Table A.3: $\alpha = 0.2(10)$, $\beta = 0.05$

Node	0	1	2	3	4	5	6	7
$Capacity_{Max}$	50	55	48	52	53	49	60	62
$Capacity_{t=n}$	43	45	46	51	50	44	52	59
Change	+2	+2	+2	+2(+1)	+2	+2	+2	+2,-10
$Capacity_{t+1}$	45	47	48	52	52	46	54	51
Change	+2	+2	+2(+0)	+2(+0)	+2(+1)	+2	+2,-10	+2
$Capacity_{t+2}$	47	49	48	52	53	48	46	53
Change	+2	+2	+2(+0)	+2(+0)	+0,-10	+2(+1)	+2	+2
$Capacity_{t+3}$	49	51	48	52	43	49	48	52
Change	+2(+1)	+2	+2(+0)	+2,-10	+2	+2(+0)	+2	+2
$Capacity_{t+4}$	50	53	48	44	45	51	50	54

Table A.4: $\alpha = 0.4(20)$, $\beta = 0.05$

Node	0	1	2	3	4	5	6	7
$Capacity_{Max}$	50	55	48	52	53	49	60	62
$Capacity_{t=n}$	43	45	46	51	50	44	52	59
Change	+2	+2	+2	+2(+1)	+2	+2	+2	+2,-20
$Capacity_{t+1}$	45	47	48	52	52	46	54	41
Change	+2	+2	+2(+0)	+2(+0)	+2(+1)	+2	+2,-20	+2
$Capacity_{t+2}$	47	49	48	52	53	48	36	43
Change	+2	+2	+2(+0)	+2(+0)	+0,-20	+2(+1)	+2	+2
$Capacity_{t+3}$	49	51	48	52	33	49	38	45
Change	+2(+1)	+2	+2(+0)	+0,-20	+2	+2(+0)	+2	+2
$Capacity_{t+4}$	50	53	50	32	35	51	40	47

Table A.5: $\alpha = 0.8(40)$, $\beta = 0.05$

Node	0	1	2	3	4	5	6	7
$Capacity_{Max}$	50	55	48	52	53	49	60	62
$Capacity_{t=n}$	43	45	46	51	50	44	52	59
Load	0.1400	0.1818	0.0417	0.019	0.057	0.1020	0.1333	0.0484
Change	+2	+2	+2	+2(+1)	+2	+2	+2	+2,-40
$Capacity_{t+1}$	45	47	48	52	52	46	54	21
Change	+2	+2	+2(+0)	+2(+0)	+2(+1)	+2	+2,-40	+2
$Capacity_{t+2}$	47	49	48	52	53	48	16	23
Change	+2	+2	+2(+0)	+2(+0)	+0,-40	+2(+1)	+2	+2
$Capacity_{t+3}$	49	51	48	52	13	49	28	25
Change	+2(+1)	+2	+2(+0)	+0,-40	+2	+2(+0)	+2	+2
$Capacity_{t+4}$	51	53	50	12	15	51	30	27

Appendix B

Additional Waterflow performance

The following are additional results comparing various route selections.

Capacity Distribution, 1-5x

We then run further tests and increase the spread of the capacity distribution for the following configurations. Figures B.1, B.2, B.3 and B.4 are results of 1-5x capacity distribution. 1-5x correspond to a refill rate of 64-320 units (tokens).

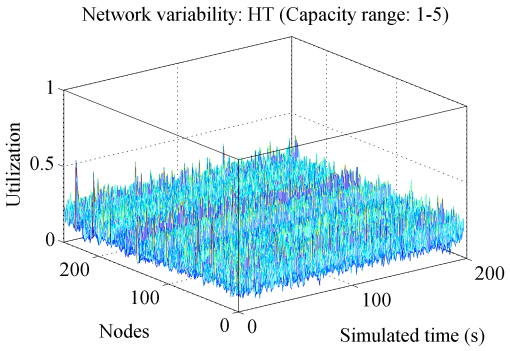

Figure B.1: WF Compare: HT
8000 clients, 1-5x
$\mu = 0.16762$, $\sigma = 0.05288$

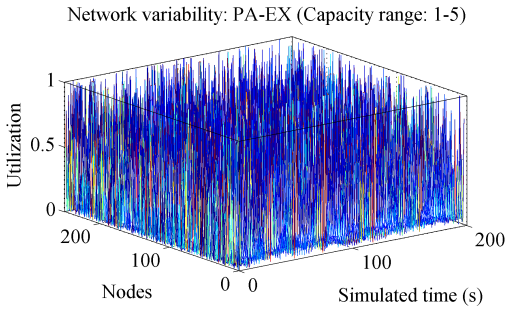

Figure B.2: WF Compare: PA-EX
8000 clients, 1-5x
$\mu = 0.18414$, $\sigma = 0.18795$

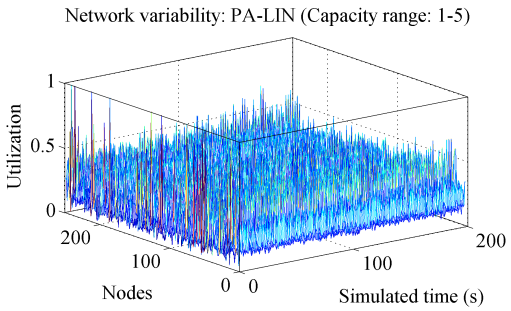

Figure B.3: WF Compare: PA-LIN
8000 clients, 1-5x
$\mu = 0.19208$, $\sigma = 0.09237$

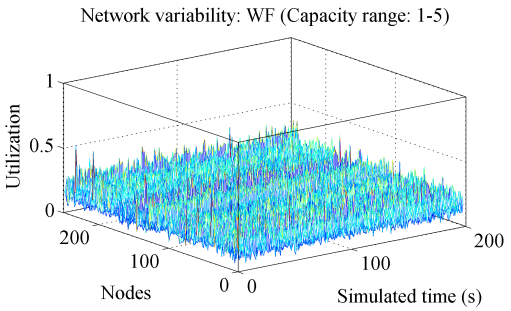

Figure B.4: WF Compare: WF
8000 clients, 1-5x
$\mu = 0.16754$, $\sigma = 0.05324$

Capacity Distribution, 1-10x

The following Figures B.5, B.6, B.7 and B.8 are results of 1-10x capacity distribution. 1-10x correspond to a refill rate of 64-640 units (tokens).

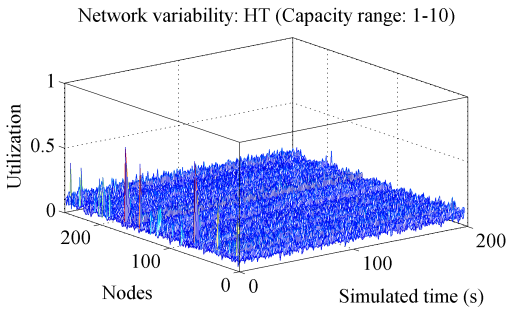

Figure B.5: WF Compare: HT
8000 clients, 1-10x
$\mu = 0.09006$, $\sigma = 0.02959$

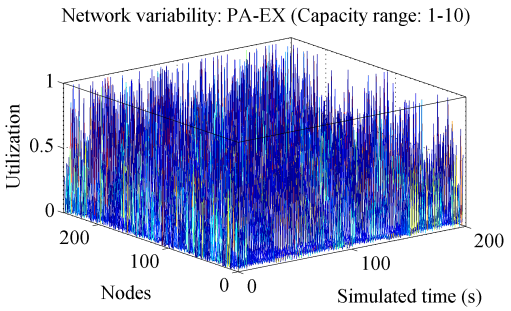

Figure B.6: WF Compare: PA-EX
8000 clients, 1-10x
$\mu = 0.11087$, $\sigma = 0.14048$

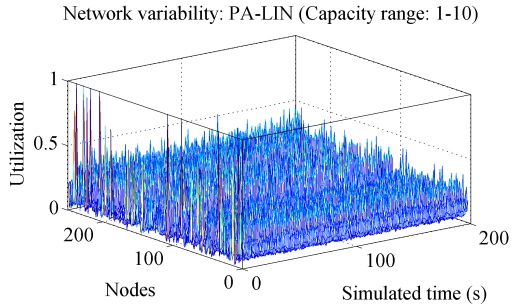

Figure B.7: WF Compare: PA-LIN
8000 clients, 1-10x
$\mu = 0.11610$, $\sigma = 0.07482$

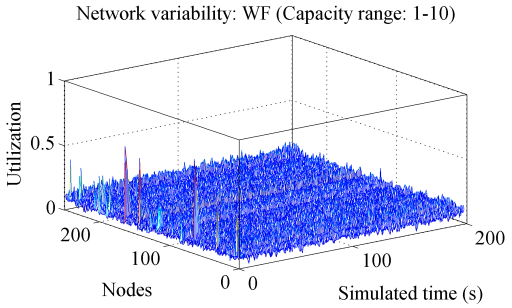

Figure B.8: WF Compare: WF
8000 clients, 1-10x
$\mu = 0.09011$, $\sigma = 0.02995$