ANGLIA RUSKIN UNIVERSITY

THE RESILIENCE AND OPTIMISATION OF CLOUD COMPUTING

RAZVAN-IOAN DINITA

A thesis in partial fulfilment of the requirements of
Anglia Ruskin University for the degree of
Doctor of Philosophy

February 2015

Acknowledgements

I would like to express my special appreciation and thanks to my advisor Professor Dr. Marcian Cirstea, you have been a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow as a research scientist. Your advice on both my research as well as on my career has been priceless. I would also like to thank Dr. George Wilson and Mr. Adrian Winckles for serving as my Supervisors and offering me support and guidance throughout my research. All of you have been there to support me when I conducted my experiments and collected data for my PhD thesis.

A special thanks to my family, words cannot express how grateful I am to my mother and father for all of the sacrifices that you've made on my behalf. Your prayers for me were what sustained me thus far. I would also like to thank all of my friends who supported me in writing, and encouraged me to strive towards my goal. At the end I would like express appreciation to my beloved partner Brindusa who spent sleepless nights by my side and was always my support in the moments of need.

ANGLIA RUSKIN UNIVERSITY

ABSTRACT

FACULTY OF SCIENCE AND TECHNOLOGY

DOCTOR OF PHILOSOPHY

THE RESILIENCE AND OPTIMISATION OF CLOUD COMPUTING

RAZVAN-IOAN DINITA

February 2015

The field of Cloud Computing is relatively new branch of Information Technology in which various services are devolved from a centralised local location to a de-centralized remote Intranet/Internet environment. It has recently experienced rapid growth and acceptance with academia and industry, presenting new challenges worthy of fundamental research. Some of the largest challenges today revolve around achieving higher levels of sustainability and infrastructure performance.

This work investigates an optimised and novel approach to an Autonomous Virtual Server Management System in a 'Cloud Computing' environment through designing and building an Autonomous Management Distributed System (AMDS). The AMDS helps reduce hardware power consumption through autonomously moving virtual servers around a network to balance out hardware loads, as well as being easily configurable and extendable, made possible by its software infrastructure. Through use of an internally configured Cloud Computing test-bed rig, the AMDS makes use of several physically and logically defined networks to communicate with all devices that are a part of the cloud infrastructure. Once connected, the AMDS monitors these devices and issues optimisation commands accordingly. Experimental results show an overall power consumption reduction of up to 8%, which in a typical datacentre of several thousand servers translates into a significant cost reduction.

This work also presents an initial design, along with proof-of-concept implementation as an AMDS module, of a Botnet heuristic detection algorithm. Experimental results show an overall malicious data packet detection rate of 52%, a significant figure for only 5000 data samples analysed by the module. Another strength is that this design allows an abstract software model to be constructed, which can then be implemented using a multitude of programming languages.

This research shows how the carbon footprint of a Cloud Computing datacentre can be reduced and reveals a significant impact on issues of sustainability with respect to both energy efficiency and economic viability. It also shows how datacentre security can be enhanced by detecting Botnet activity and preventing the disruption of day-to-day operations through a highly scalable, flexible, and autonomous software implementation.

Key words: Distributed Software, Carbon Footprint, Sustainability

Table of Contents

Copyright

Chapter 1: Introduction

Chapter 1: Introduction

Computing is a term referring to goal-oriented activities that revolve around algorithmic processes (JTFCC, 2005). Ever since the middle of last century, the scientific computing domain has evolved more rapidly than any other technology. It began with highly simplistic computing devices, quickly moving to more complex devices, and, for the better part of last half of the 20th century, evolving into very large clusters of interconnected computing devices working towards the same pre-set computational goals, almost identical to utility computing (Carroll et al., 2012).

"Cloud Computing" is a higher-level term used to describe distributed computing, which involves data flows across real-time, high-speed networks. This enables the possibility of running intensely computational applications across multiple physical devices in parallel. This term, when used by laypeople, generally refers to generally available online services. These services, although appearing to be supported by physical hardware, are in fact running on virtualised hardware simulated by software operating on real machines. One of the main advantages of this approach is the ability to scale, up or down, these virtual infrastructures on the fly without any noticeable service disruptions, similar to a cloud (CORDIS, 2013).

Cloud Computing is also viewed as the sum of multiple technology iterations and paradigms. Its scope is to provide benefits from all involved technologies without requiring their complete understanding. It aims to reduce costs and shift user focus to the business side rather than keeping it on potential IT barriers (Hamdaqa, 2012).

A central part of Cloud Computing is virtualisation, that is, the deployment of software versions of hardware implementations, such as Storage drives, CPU and RAM modules, network devices etc. This enables physical infrastructure generalisation, transforming it from the most immutable component to a highly mutable, manageable collection of computing resources. As such, IT operation costs are being reduced through improved implementation, speed and maximised infrastructure utilisation. In addition, by employing autonomic computing processes, clients are able to provide computing resources on-demand. Furthermore these automations minimise user involvement, which in turn improves the speed of operations whilst reducing human errors (Hamdaqa, 2012).

By employing notions from Service-Oriented Architecture (SOA), Cloud Computing helps business clients avoid and overcome daily problems through computing solutions. Through the services it provides that make use of tried-and-tested SOA standards and best practices, Cloud Computing enables easy, global access to cloud services in a homogenised manner (Hamdaqa, 2012).

Cloud Computing shares concepts with 'utility computing' which allows for cloud pay-per-use models and services to employ metrics, so making them an integral part of various autonomic computing processes. This allows for highly scalable and fault-tolerant cloud services. Cloud Computing is also similar to 'grid computing', but throughout its technological lifetime has managed to address potential quality of service (QoS) and reliability issues. Cloud Computing also provides a large set of tools that enable development of data and/or computer intensive parallel applications at a greatly reduced cost compared to traditional parallel computing approaches (Hamdaqa, 2012).

Due to the relatively infant nature of Cloud Computing, currently standards are still yet to be definitive. Consequently several cloud platforms and services employ proprietary standards, tools, and protocols developed in-house and fit for purpose. This impacts heavily on the application migrations between cloud platforms making the process complicated and expensive (McKendrick, 2011). These discrepancies are generally referred to as vendor lock-ins, which can be of three types: platform, data, and tools (Hinkle, 2010). Each of these lock-in types provides different challenges, ranging from technical (platform lock-in) to data ownership (data lock-in) to management tools (tool lock-in) issues.

A new Cloud Computing type has been defined in recent years called 'heterogeneous Cloud Computing'. This type of cloud environment hinders vendor lock-in, aligning itself with enterprise hybrid cloud model based datacentres (Staten, 2012). The nonexistence of vendor lock-in creates a choice of hypervisors (virtualized operating systems) for specific tasks by removing the necessity of hypervisor flavour client considerations (Vada, 2012). Heterogeneous clouds are typically made up of private, public, and software-as-a-service clouds, all co-existing on-site, having the ability to integrate well with traditional, non-virtualised datacentres. (Geada et al., 2011) They also are composed of multiple vendor-supplied hypervisors, servers, and storage (Burns, 2012). Cloud systems expose Application Programmable Interfaces (APIs) that are quite often ill matched with each other. This raises numerous technical compatibility issues when migrating applications between vendors or systems. A pertinent solution would be the adoption of common standards (Livenson et al., 2011).

Cloud Computing services were first made available to the general public in 2006[1] followed by proprietary Cloud services as offered by Amazon EC2, Google and Microsoft for example. The main driver for this technology has come from the need to reduce operation costs and at the same time maximise hardware resource utilisation, which led to years of individual company efforts to further develop these in-house software tools and platforms. All of the key companies in this field have developed proprietary software to help better manage the Cloud infrastructure.

As such, due to the highly commercial nature of these companies, they have not at any point in time fully shared their recent technological advances amongst themselves as well as with the wider public, being academic or otherwise. Such technology designs have not been fully made public and are not available in any of the available academic literature (journal and conference papers, books). Whilst similar technological solutions are available, most are Field-Programmable Gate Arrays (FPGA) based, making them very rigid in terms of adaptability and flexibility. This has left a significant gap in knowledge, which fundamental research is able to fill.

1.1 Research Scope and Objectives

The question this research work is attempting to answer is:

Can the efficiency, security, and robustness of Cloud Computing be enhanced through improved software design?

As such, the main aim is to create a software entity capable of interfacing with existing infrastructure (Hypervisors, Routers, Switches, ILOs etc.) and obtain information from each relevant network device / software. The entity would then analyse this data and transmit commands back to the linked devices in order to increase overall operational efficiency.

To achieve this aim, the following objectives will be pursued:

[1] Barr, J.; (August 25, 2006). “Amazon EC2 Beta”, Amazon Web Services Blog.

Objective 1. Critically evaluate pattern of disruption across a Cloud infrastructure as a result of an overloaded service request.

Objective 2. Conceptually develop a software optimization technique by which a Cloud could autonomously manage the workloads placed on that infrastructure.

Objective 3. Implement and test a software application to achieve *Objective 2* for a specific Cloud scenario (VMWare Hypervisors).

Objective 4. Innovatively develop metrics that quantify Cloud vs. centralized service provision in terms of environmental sustainability.

Objective 5. Conceptually develop an application that will identify virtualised system hijacking and undertake a range of appropriate activities from simple notification to service suspension.

Objective 6. Test the method/software and compare against other alternatives, e.g. FPGA/hardware and other software systems.

The detail of these objectives will now be presented in terms of their specific contribution to knowledge.

1.2 Research Objectives Breakdown – Filling the Knowledge Gap

Objective 1:

- The protocols that enable current network topologies to interface in such a way as to support Cloud functionality are well established. However the effect of an un-anticipated amount of people trying to access the same file/service is poorly understood (Williams et. al., 2011). This effect will be investigated and thoroughly documented.

Objective 2:

- Key causes of disruption will be examined and a solution will be designed and developed to ensure the resilient and optimal operation of a cloud network.

The design will be of a modular in nature and, as such, easily extensible to accommodate any type of Cloud infrastructure.

Objective 3 refers to the management of a Cloud Computing network and is comprised of three parts:

- The optimised running of a cloud will be investigated and a new set of metrics that can be tested will be developed to quantify the efficiency (carbon footprint) of an optimised cloud network compared to a non-optimised cloud network and to a non-cloud infrastructure (Anderson, 2010; Armbrust et.al., 2009; Chou et al., 2011).
- Management tools are available to enable Cloud Administrators to re-configure hardware loadings according to service demands, but the approach is largely by-trial-and-error (Moretti et. al., 2008; Tsutomu, 2010). This research will test existing tools and design and develop a new set of tools aimed at helping Cloud Administrators improve overall cloud energy efficiency, while at the same time maintain an acceptable level of service.
- Existing management technology designs are available, however most are a FPGA based, making them very rigid in terms of adaptability and flexibility (Murakami, 2008; Cirstea, 2003; Junyoung et al., 2009). This research, through the software it will produce, will present a viable and easy to use alternative to existing approaches.

Objective 4:

- One consequence of the adoption of Cloud Computing in the commercial sector is that companies do not need to invest in their own hardware infrastructure. This has environmental consequences and their quantification is important for developing strategies for a sustainable environment (Chou et al., 2011). This research will attempt to quantify the environmental sustainability of Cloud Computing.

Objective 5:

- A botnet is a group of compromised computers connected to the Internet. Each compromised computer is called a bot, and could include individual virtual bots within a virtualised system. Whilst there are tools to protect a Cloud from such malware attacks, a very poorly researched area is how a successful hijacking of a Cloud's virtual operating system could be identified (Zeidanloo, et. al., 2010; Chandrashekar, 2009; Ke et. al., 2009; Murakami, 2008; Rutkowska et al., 2008). This research will develop a technique to identify the successful hijacking of a Cloud's virtual operating system.

Objective 6:

- In the context of this research a VMWare based cloud technology is being used to test Objective 3. This research will use Objective 3's results to compare the proposed software solution to other similar, but FPGA based, technology test results in terms of reliability, efficiency and flexibility.

At this point, an overview of the methodological steps undertaken towards filling the research gap will be presented.

1.3 Approach

Since the focus of this research is on software, the author has chosen to follow a traditional Software Development methodology from the point of view of designing, implementing, and testing the AMDS. The steps undertaken and the reasoning behind each of these steps will be presented in a later chapter, but broadly speaking the following has been undertaken:

- Review of the State-of-the-Art
- Developing the Software
- Setting it up on Hardware
- Optimising the Software

The Gantt Chart in Appendix A presents further information on the project management of this research.

1.4 Original Contributions to Knowledge

This research work has resulted in several contributions to knowledge in this subject area and are briefly listed below.

i. A novel method of optimising cloud networks in terms of energy consumption and system operation. The novelty consists of the software's ability to reconfigure itself on the fly based on live network readings.

ii. A novel method to prevent, detect and stop network intrusions and malicious behaviour in a cloud infrastructure.

iii. A flexible software solution (as opposed to the general approach of using rigid hardware) to a general communications/networking problem. The software solution will automatically acquire data about traffic and hardware loads of a cloud infrastructure, analyse them and redistribute loads for efficient energy management and optimal data communication parameters (security, data transfer speed, access wait times, power consumption).

1.5 Overview of the Thesis

This document is structured into nine chapters; the content and purpose of each are briefly described as follows:

Chapter 1**,** 'Introduction' sets the research context by providing a brief look into the history of Cloud Computing. It identifies several key issues still in need of research as well as present the research questions that the author has set out to investigate. The benefits of creating an autonomous system capable of optimising a Cloud Computing environment from an energy consumption point of view are briefly described, as is how such a system is capable of detecting unauthorised access of a Cloud infrastructure network.

Chapter 2 'A review of the state-of–the-art of Cloud Computing' then follows which includes the relevant literature background that has shaped the research questions, and the optimization implications for efficiency (carbon footprint), management and security.

A detailed explanation of what a Cloud Computing environment is composed of and several Botnet monitoring techniques are described in Chapter 3 entitled 'Cloud Principles of Operation and Botnet Monitoring Techniques'. It describes each typical component of a cloud and how they interact with one another within that overall Cloud structure, as well as going over some of the better-known Botnet detection techniques.

Chapter 4, the 'Research Methodology' chapter focuses on the approach and strategy taken to answering the research questions. The design and configuration of a Cloud Test Bed is described including both the virtual (network and software) and the physical (hardware and physical parameters) aspects of the infrastructure.

The title of Chapter 5 is 'Autonomous Management Distributed System (AMDS) – The Software' which focuses on the design, development and the implementation of the AMDS. It considers the following main points:

- the network optimization technique that serves as basis for the software implementation;
- the design of all system components and the relationship between them ;
- the development strategy, detailing the steps undertaken towards the implementation;
- the implementation process, including the choice of programming language and what specific features were utilised.

The AMDS deployment strategy is described in Chapter 6; 'Cloud Computing Test Bed – Software Deployment on Hardware', and also describes the detail as to how the AMDS was installed on a Virtual Machine and what steps have been undertaken to achieve interconnectivity between the Virtual Network interfaces and the underlying physical hardware. It also focuses on how data results from the experimental work is generated and presented in the form of logs over a period of several months, and also presents some analyses of these results.

Chapter 7 'AMDS System Enhancement – Botnets' utilises the Chapter 6 results in the context of Network Hijacking. It presents, in depth, a novel prevention strategy that could be implemented as a module of the AMDS.

The test results from the AMDS and other FPGA based solutions are compared side-by-side in Chapter 8 'AMDS in Comparison with Alternative Solutions'. The comparisons take into account how each solution performs in terms of security, reliability, data transfer speed, access times and power consumption efficiency.

Finally, Chapter 9 summarises how energy costs can be driven down, operational efficiency improved and security enhanced by deploying the AMDS on any Cloud environment (proof of concept was developed on a VMWare backed test bed). The work reflects on the initial research questions and evaluates to what extent they have been answered.

It also presents several ideas that could be part of future research projects. Such projects would utilise and expand the AMDS to further benefit a Cloud Computing environment.

Chapter 2: A Review of State-of-the-Art in Cloud Computing

Chapter 2: A Review of State-of-the-Art in Cloud Computing

An overview of the main themes and issues emerging from reviewing the research literature related to Cloud Computing will now be presented, highlighting those key issues that helped define this research field. The history and definitions comprising Cloud Computing will also be addressed.

2.1 Initial Background Considerations

The first theme considered relates to the efficiency quantification (carbon footprint) of an optimised cloud network compared to a non-optimised cloud network and to a non-cloud infrastructure.

In their work on the future of Cloud Computing, Anderson et al. (2010) present the views of several academics and researchers with respect to the efficiency of running applications in a Cloud environment compared to traditional, localised applications running on a desktop computer. They argue that running applications in the Cloud improves overall operational efficiency of an application due to it not requiring dedicated hardware, which most of the time translates into wasted resources if the hardware is not running at full capacity. This also implies a higher than needed carbon footprint and higher costs due to the suboptimal configuration.

Armbrust et al. (2009) also discusses the operational efficiency of cloud applications compared to traditional approaches. It is argued that cloud environments provide a much higher operational efficiency through optimally configured hardware resources, and, as such, they render much lower operating costs and reduced carbon footprint.

Chou et al. (2011) argues that efficient computing is an area of development for Green Computing – an energy and efficiency focused approach to Cloud Computing. From there, Sustainable Computing emerged as a new development focused area of Cloud Computing. In any case, they both focus on reduced carbon footprint through improved efficiency through hardware and software design.

Another aspect investigated refers to the effect of an unanticipated number of people trying to access the same file/service. This work will present the results of several hardware overload experiments and their implications.

Williams et al. (2011) discusses the effects such an event would have in a datacentre

environment. It defines an overload as being caused by a Virtual Machine (VM) having access to less memory (RAM) than needed, usually due to hosting a service accessed by more people than initially anticipated. The solution presented is a software-based approach to VM Management capable of mitigating such overload events on-demand.
A tool designed to manage multiple VM clusters (queues) is presented by Bagci (2014). It employs two algorithms managing the addition and removal of VMs to and from the clusters, as well as client request redirection to the least utilised cluster.

This work will also look into what management tools are available to enable Cloud Administrators to re-configure hardware loadings according to service demands. One such tool is proposed by Moretti et al. (2008), a user-friendly abstraction engine/tool, “All-Pairs”, to be used in data-intensive tasks. This would allow users to specify a “high-level structure of the workload” (Moretti et al., 2008, Fig. 2 Caption), while the abstraction engine takes care of partitioning and dispatching resources towards completing the task.
Another similar tool, but more comprehensive as it is designed to act as a sole entity on a cloud infrastructure, is presented by Feller et al. (2014). This tool, called Snooze, builds on the strengths of existing open source cloud management solutions to deliver a highly scalable and flexible complete set of tools for managing large scale Infrastructure as a Service (IaaS) infrastructures.

The consequences and effects of the adoption of Cloud Computing in the commercial sector are also addressed. Chou et al. (2011) looks at trends and characteristics and makes a case for companies adopting a sustainable Cloud Computing, presenting increased automation, flexibility, and reduced operational costs as clear benefits.

Finally, this work also presents strategies/solutions towards how a successful hijacking of a Cloud’s virtual operating system could be identified and mitigated.
It builds on work by Chandrashekar et al. (2009), which presents an overview of botnets and stealthy malware and then goes on to discuss several defence strategies for “current and emerging trends in botnet development”.
This research also looks at work done by Ke et al. (2009), which presents a software based botnet detection tool implemented as a standalone application looking at network traffic.
Murakami (2008) and Rutkowska et al. (2008) on the other hand, both present software approaches as VM Hypervisor enhancements. They attempt to discover

botnet activity by having the Hypervisor discover malicious behaviour in already infected VMs.
Zeidanloo et al. (2010) and Karim et al. (2014) looks at existing botnet detection approaches and presents their strengths and weaknesses, while at the same time categorising them into honey-pot and Intrusion Detection System (IDS) techniques.

2.2 Cloud Computing History

This section will present a timeline of the Cloud Computing developments throughout history, from its early days until more recent times.

2.2.1 Early History

Cloud Computing concepts first emerged in the 1950s, when large-scale mainframes were made available to be used in different size organisations, ranging from schools to big corporations. Since the initial design of the mainframe required significant amount of physical space to operate, they generally resided within their own room, and network access was facilitated via simple design machines known as dumb terminals. Due to the high costs of purchasing and maintaining mainframes, multiple users shared the same data storage layer and CPU power from any station. This practice yielded better return on investments (Strachey, 1959).

In the 1970s, IBM designed an operating system called VM that made it possible for several virtual environments (Virtual Machines - VMs) to coexist on the same physical mainframe system. These VMs were capable of running different guest operating systems, each with virtual access to a part of the total available physical RAM and CPU, as well as direct access to CD-ROMs, Keyboards, Networking, etc. It was at this point in time when virtualisation became the biggest technology driver in the recent history of communications and computing areas (Smith et al., 2005).

As hardware costs slowly came down, it became necessary for a new paradigm shift to occur. This was achieved through the development of a software system called the Hypervisor, which was capable of bringing multiple physical nodes together and presenting them as a single node operating under the same paradigm as the earlier virtualisation principles. For visualisation purposes, this infrastructure design was assigned the term of *Cloud Computing* (Ryan et al., 2013).

2.2.2 Recent history

Cloud Computing has only recently become a commercial reality. During the last decade, the term “Cloud Computing” has become customary among large companies having a major impact on how software is viewed in current computing paradigms. It has shown big potential to innovate the IT industry and to make software more attractive by taking into account that a large capital is no longer required to build a customized hardware local infrastructure (Armbrust et al., 2010). Cloud Computing thus offers companies a new way of using web services for their computing needs (Breeding, 2012).

A significant number of companies have looked upon cloud services as a low cost solution for data storage and computing needs (Corrado et al., 2011). According to Armbrust et al. (2010), the inherent ability of cloud providers to sell their services on a pay-as-you-go basis means that businesses are able to write off large investments and maximise profits. A direct consequence is the reduction of their carbon footprint by having less hardware on location.

In recent times, Cloud Computing has had a major economic impact and it is seen as key to boosting the world’s competitiveness (Bornico et al., 2011). An advantage of using cloud services is that cloud resources are virtualised and the users can have private access to their infrastructure. Another advantage is that cloud applications are device-independent when compared to traditional approaches, meaning that any device with a web browser (Desktop, Laptop, Table, Smartphone) is capable of accessing them over the Internet (Baun et al., 2011; Barnatt, 2012).

As a prediction for the year 2020 made by the IT market research company Pew in 2010, services such as Yahoo!, Twitter, YouTube, Facebook, Hotmail or Google Apps will end up implementing cloud technology because most people will want to easily upload and share information (Anderson at al., 2010). Forrester Research expects the cloud industry will grow from $41 billion in 2010 to $241 billion in 2020 (Breeding, 2012).

2.3 Cloud Computing – Definition and Terminologies

Cloud Computing is based on the principle of a host company developing software (a Hypervisor) capable of managing multiple physical and virtual hardware at the same time. This allows it to virtually portion physical hardware resources (CPU, RAM, Storage) and securely assign them to different customers (Virtual Machines), thus eliminating the need for one physical hosting device per customer. Multiple Hypervisors are typically controlled by another higher-level software entity (e.g.: VMWare vSphere).

More recently the term Cloud Computing has been used to describe a paradigm shift in the Internet hosting industry. Since it was first mentioned in 2007, researchers tried to define Cloud Computing from different points of view, but did not manage to reach a consensus regarding its definition. One of the most clear and accepted definitions for the term cloud given by NIST (Mell et al., 2011) is: "Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction".

Another definition for the term "Cloud Computing" is given by Baun et al. (2011): "By using virtualized computing and storage resources and modern Web technologies, Cloud Computing provides scalable, network-centric, abstracted IT infrastructures, platforms and applications as on-demand services. These services are billed on a usage basis." This definition is not as specific as the NIST definition and it does not specify whether the services are provided by a distributed system or by a mainframe.

A core principle of Cloud Computing is resource sharing towards achieving scalable, coherent economies over a network (Mell et al., NIST, 2011). As such, the cloud employs processes that focus on maximising shared resources effectiveness. The main drive for these processes is the cloud resource requirement of being dynamically reallocated on-demand (Amazon Web Services, 2013). For example, the cloud could allocate resources for European clients during European business times towards supporting a specific application, and reallocate the same resources for North American clients during North American business times to support an entirely different application. This enables maximising of computing resource usage while at the same time reducing environmental impact through the lessened need for power (e.g. air conditioning, rack space) by different computational functions.

In conclusion, the author has chosen to use NIST as the de facto reference in this field as this gives the most comprehensive definition of Cloud Computing as well as a clear breakdown of the different components it comprises.

2.4 Cloud Essential Characteristics and Service and Deployment Models

According to NIST (Mell et al., 2011), Cloud Computing is in its most basic form a model for enabling secure, reliable, on-demand network access to a shared pool of computing resources, ranging from physical systems (routers, switches, servers, storage) to logical systems (applications, services). These can all be provisioned and released autonomously, as required by different scenarios and situations. This model employs five basic characteristics, three service models, and four deployment models.

2.4.1 Essential Characteristics

According to NIST (Mell et al., 2011), cloud clusters evidence several defining characteristics.

On-demand self-service. A client can, through the use of a fully automated and autonomous Cloud management service, provision computing capabilities. These translate into active server time (services, data analysis) and network storage.

Broad network access. Generally, over-the-network services are available through standardised means to a range of different thin and thick client devices. These range from small mobile devices (phones, tablets, laptops) to fully fledged workstations (PCs) (Mell et al., NIST, 2011).

Resource pooling. The Cloud’s resources are pooled together to service multiple clients simultaneously. This is achieved by dynamically assigning and reassigning physical and logical resources in accordance to current demand. Although the client has, most of the time, no actual control of their data’s physical location, they can in some cases specify a location at a higher abstraction level. Options tend to be limited to country, state, or datacentre. Available resources include storage, computation power, memory (RAM), and network bandwidth (Mell et al., NIST, 2011).

Rapid elasticity. Cloud management systems typically offer the client scalability options, which allow for rapid outward and inward resource scaling. This is, however, hidden from the client who is given the impression of an unlimited amount of resources put at their disposal (Mell et al., NIST, 2011).

Measured service. The autonomous management service responsible for the entirety of Cloud resources makes use of metering capabilities at different levels of abstraction appropriate to the type service (CPU, RAM, storage, network bandwidth, active user accounts) to control, optimise, and maximise resource usage. These capabilities provide both the consumer and the aforementioned management service with means to monitor, control, and generate reports on the Cloud services employed at a given time (Mell et al., NIST, 2011).

2.4.2 Service Models

Referring back to the NIST Cloud Computing overview (Mell et al., 2011), cloud clusters employ three main service models.

Software as a Service (SaaS). SaaS is focused on software appliances as services. It is defined as software that can be purchased without the need to purchase other infrastructure in order to run it. The main difference between traditional software and cloud-based software is that users do not need to update the system or fix bugs any more. Users just have to register for an account and to log in through a simple interface such as Google Chrome or Internet explorer over the Internet. The client is given the capability to use Cloud services running on a physical network infrastructure. Although applications are accessible through a typically wide range of different devices, the client does not at any given point in time have direct management access to the underlying infrastructure or core application capabilities, except for limited user-specific configuration settings (NIST, 2011). Any problem involving the software is now pushed out of the way and taken care of by the vendor

(Corrado et al., 2011). For instance, many applications such as social media platforms (Facebook) or office software (Google Docs) are SaaS based services.

Platform as a Service (PaaS). PaaS is based on computational resources as high-level application platforms. This concept is difficult to understand by first time clients. PaaS users have the choice of deciding which software platform (developed by a software provider) to use. PaaS was developed to facilitate application development and better manage potential issues. The client is given the capability to deploy consumer applications created using software tools (programming languages, libraries, services) supported by the Cloud service provider. The client does not at any given point in time have direct control over the underlying infrastructure, but has control over their own deployed applications as well as some configuration settings within the application hosting environment (Mell et al., NIST, 2011). Examples of PaaS are Microsoft Azure and Google App Engine (Corrado et al., 2011).

Infrastructure as a Service (IaaS). IaaS refers to a flexible, virtualised and scalable service that facilitates access to hardware resources. Using IaaS, users can create, maintain and keep backup copies of their servers in different physical datacentres. The client is given the capability to provision CPU, storage, bandwidth, and other core computing resources used to deploy and manage arbitrary software (operating systems, system based applications). The client does not have direct control of the underlying infrastructure, but does however fully control the applications or operating systems they have deployed. The client might also have limited control over some specific networking components (firewalls) (Mell et al., NIST, 2011). This type of infrastructure service is usually associated with Amazon Web Services (AWS) and its services, Elastic Compute Cloud or EC2 and Simple Storage Service or S3. IaaS also includes Virtual Private Cloud (VPC, described in section 2.4.3), IBM Blue Cloud or FlexiScale (part of Flexiant after being acquired in 2007[2]) (Corrado et al., 2011).

2.4.3 Cloud deployment models

The Public Cloud is a type of cloud service that is available for general use, where users just have to create an account and to sign in in order to run their applications.

[2] https://www.flexiant.com/news/flexiant-targets-web-hosting-sector-with-cloud-infrastructure-software/

This is typically free of charge within certain storage limits set up by the cloud provider, but can also be subject to a monthly subscription. Providers also offer the possibility of creating a VPN connection (Corrado et al., 2011), securely linking in-house servers to public cloud resources. There are many providers that offer public cloud services such as, Amazon, Google, Microsoft or Apple.

The Community Cloud is a cloud infrastructure used by various organizations that share common aims. This type of cloud can be managed by the organizations using it or by a third party (Mell et al., NIST, 2011). Clients like the European Union or the U.S. Government are using community clouds for the purpose of sharing resources around the world and within different agencies. Community Clouds are also used in higher education environments and libraries, some of these institutions having developed a complex network they share with one another.

Private Cloud, or Virtual Private Cloud (VPC), is defined as an on demand service that provides infrastructural resources and maintained solely by an IT department within private businesses other than known cloud providers. The aim of these private services is not to sell an amount of storage over the Internet through its interfaces (Sotomayor et al., 2009), but are focused on providing a flexible and infrastructure for private use within the organization.

The Hybrid Cloud is a type of cloud service that allows users to run their applications using more than one cloud service or provider, which allows them to take full advantage of each deployment model (Corrado et al., 2011). More often than not, a hybrid cloud is a combination between private cloud and public cloud (Zhang et al., 2010). A key advantage hybrid clouds possess is the VI management interface where specific requirements such as providing a homogeneous view of virtualised resources or managing the full lifecycle of a virtual machine must be fulfilled (Sotomayor et al., 2009).

2.5 Logical Break-down of Cloud Computing Environments

Cloud Computing systems are generally regarded as two separate, but interconnected entities. The *Backend* is comprised of the entire Cloud network (wired physical network and server hardware), while the *Front-End* represents the user side, which aids in administration as well as service usage (Brodkin, 2007).

The Front-End typically makes use of either specialised or more commonly available software. Specialised software tends to be created for the sole purpose of managing a particular cloud service (Google App Engine), while the latter is used with wider availability services, such as Email (GMail).

The Backend makes use of numerous specialised interconnected hardware systems designed to facilitate network access (cables, routers, switches), computational needs (servers), and storage requirements (NAS - Network Attached Storage). These systems can, in theory, provide a very wide range of services from data processing to video games to simple websites. Often, each such service would benefit from a dedicated part of the Cloud system (Brodkin, 2007).

A Cloud network is generally autonomously administered by one server or a pool of servers acting as a single entity. It handles tasks ranging from traffic monitoring to service provision management. It follows a set of standardised protocols (rules) and makes use of helper software called *Middleware* (Middleware.org). Middleware is specialised software, comprised of one or more software entities, that facilitates system communication across the Cloud network between servers and storage mediums.

Generally servers are not utilised to their full potential, thus generating important resource waste. To counter this, a technique called *Virtualisation* is used to split a server into multiple logical, smaller servers that run independently from each other, but linked together through one or more logical networks, which allows for maximised performance yield (Smith et al., 2005).

With regards to *Storage* needs, the required data capacity of a particular service is, in practice, double its size. This is because in a Cloud environment each piece of data is stored in at least two different places: main storage mediums, and backup storage mediums. This practice is referred to as *Redundancy*. Redundancy is a principle that allows for data recovery in the case of hardware failure, which is a likely possibility in high demand environments. It is automatically employed by the Cloud management software by duplicating each piece of information and storing them in different physical locations across the Cloud network. This allows for quick recovery in data loss scenarios (Singh, 2009).

All Cloud Computing environment components and principles presented (Front-End, Backend – Middleware, Virtualisation, Storage, Redundancy) aid in creating a secure, robust, and redundant system that can be used for highly mission critical as well as non-mission critical services.

2.6 Datacentre Structural Design

Housing high amounts of computational and storage power, the datacentre underpins Cloud Computing instances and contains thousands of networking hardware (servers, firewalls, routers, switches). As such, careful consideration must be given to the design and physical layout of this network architecture, as it is key in the successful and optimal operation of virtual applications running on top. Furthermore, aspects such as resiliency, scalability and security require special consideration during this design phase.

At the moment, most network architecture designs employ tried and tested concepts of layered approaches. These approaches consist of three basic layers: access, aggregation, and core. The *access* layer consists of the physical interconnected hardware. They share data over fast and very fast network links of 1 and 10 Gbps routed accordingly through switches, each backed by two additional aggregation switches which provide redundancy fail-safes. The *aggregation* layer deals with critical aspects such as the domain, location, load balancing, and other important services. The *core* layer facilitates connectivity between several aggregation switches. This ensures resiliency through no single point of failure. Routers within this layer deal with inwards and outwards traffic management.

A common infrastructure design makes use of many Ethernet routers and switches. This design generally meets the following criteria, as described in Al-Fares et al., (2008); Greenberg et al., (2009); Guo et al., (2008); Guo et al., (2009); Mysore et al., (2009):

- *Uniform high capacity*: The maximum rate of a server-to-server traffic flow should be limited only by the available capacity of the network-interface cards of the communication aware servers, and server assignment to a service should be independent of the network topology. It should be possible for an arbitrary host in the datacentre to communicate with any other host in the network at the full bandwidth of its local network interface.

- *Free VM migration*: Virtualisation allows the entire VM state to be transmitted across the network in order to facilitate migration of a VM from one physical machine to another. A Cloud Computing hosting service may migrate VMs for statistical multiplexing or dynamically changing communication patterns to achieve high bandwidth for tightly coupled hosts or to achieve variable heat distribution and power availability in the datacentre. The communication topology should be designed so as to support rapid virtual machine migration.
- *Resiliency*: Failures tend to be common in highly scalable environments. The network infrastructure must be fault-tolerant against various types of server failures, link outages, or server-rack failures. Existing unicast and multicast communications should not be affected to the extent allowed by the underlying physical connectivity.
- *Scalability*: The network infrastructure must be able to scale to a large number of servers and allow for incremental expansion.
- *Backward compatibility*: The network infrastructure should be backward compatible with switches and routers running Ethernet and IP. Because existing datacentres have commonly leveraged commodity Ethernet and IP based devices, they should also be used in the new architecture without major modifications.

2.7 Cloud Computing Standards

This section presents an overview of current Cloud Computing standards that have influenced the author's decision of which cloud architecture to deploy on the cloud test bed.

Even though Cloud Computing has rapidly spread among both public and private sector, widely embraced standards have not yet been developed. As such, from this point of view, a pressing concern is the *Vendor lock-in*. This is a practice whereby, due to the nature of the cloud infrastructure setups among the bigger Cloud Service Providers (CSP), moving from one CSP to another is made a difficult task to complete. Unfortunately, the rivalrous economic nature of these CSPs business relations with one another push them to continue wanting even more to "lock-in" their clients in order to win a bigger portion of the already thriving "cloud market", which further intensifies the concern and the need for change (JISC, 2010).

According to several JISC case studies, research communities are highly concerned with developing new interoperability protocols and standards. Their concerns are related to current research applications being limited due to their proprietary nature. Higher education institutions wanting to move their research into the cloud are required to redesign their code to ensure correct operability with the CSP cloud APIs of their choice, regardless of whom the chosen CSP might be. The choice is difficult because once these research applications have been deployed on a specific cloud, moving them to another CSP's cloud required a tremendous amount of effort. As such, the only answer to many of these interoperability issues lies in developing rigorous standards-based control and management protocols. As such, two specifications have been developed towards achieving widely accepted standards: the Open Virtual Format (OVF), championed by the Distributed Management Task Force (DMTF), and the Open Cloud Computing Interface (OCCI), championed by the Open Grid Forum (OGF).

The OVF specification defines an open virtual machine format. It has been designed to provide a standard format for packaging software based virtual solutions. It is used to add to a "user's infrastructure a self-contained, self-consistent, software application that provides a particular service or services". (DMTF, 2011)

The OCCI "comprises a set of open community-led specifications" with an aim to deliver open management and control cloud Application Programmable Interfaces (APIs). Since its inception, it has "evolved into a flexible API with a strong focus on integration, portability, interoperability and innovation while still offering a high degree of extensibility". (OGF, 2011)

Recently, several leading IT companies, such as IBM, Cisco, HP, and Sun Microsystems, have begun pushing for wider acceptance of mentioned specifications by becoming active members of the Open Cloud Manifesto (OCM) group[3], while others, such as Amazon, Google, Microsoft, and Salesforce.com, have rejected the manifesto on account of not being given the opportunity to participate in developing the document, but just the option to join[4]. The main reason behind the OCM's existence is to "showcase the abilities and requirements that should be there in the cloud

[3] http://www.opencloudmanifesto.org/

[4] http://blogs.wsj.com/digits/2009/03/28/a-cloud-manifesto-controversy/

environment so that the processes like portability, ease of workflow management and interoperability are done properly"[5].

2.8 Commercial Cloud Offerings

This section presents several commercial cloud projects that were considered by the author when choosing which cloud architecture to deploy on the test bed.

2.8.1 VMWare vSphere and vCenter Operations Management Suites

vSphere and vCenter are both products developed by VMWare used in cloud infrastructure management. They are designed to be used together, each dealing with a different aspect of a datacentre.

According to the VMWare vSphere Features page[6], *vSphere* has been designed to manage physical resources (servers, network devices) and deploy, automate, and maintain virtualised hardware (virtual machines, devices, infrastructures). Through a large set of tools at its disposal across all Cloud Computing infrastructure logical areas, such as Computing, Availability, Automation, Network, Security, and Storage, vSphere is able to deliver high availability and fault tolerant services, as well as maintain network, security, and storage control at the same time.

According to the VMWare vCenter Features page[7], *vCenter*, on the other hand, has been designed to work in parallel with vSphere by being capable of generating reports on current network and operations. It enables deep application monitoring through an extensive dashboard that makes use of self-learning analytics and automated correlation of application and infrastructure performance. It also provides deep visibility into the storage infrastructure through several topology, statistics, and events views.

[5] http://www.opencloudmanifesto.org/cloud-computing-manifesto-white-paper/38
[6] http://www.vmware.com/uk/products/vsphere-operations-management/features.html
[7] http://www.vmware.com/uk/products/vcenter-operations-management/features.html

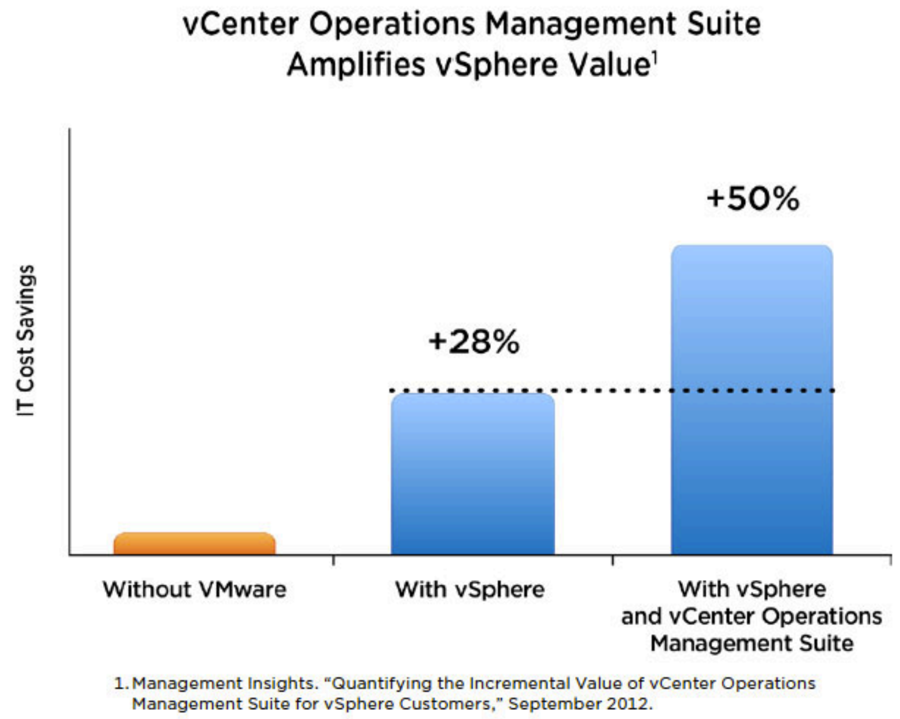


Figure 1 - vSphere and vCenter Cost Savings Representation[8]

As can be seen in Figure 1 (VMWare, 2014), compared to only using vSphere by itself, having a vCenter instance running in parallel increases IT cost savings by 22%. This can, theoretically, be achieved through careful monitoring of datacentre resource utilisation by using the Automated Cost Metering functionality provided by vCenter to gain insight into infrastructure running cost. This would allow infrastructure micro management decision making in relation to actual business needs.

Finally, from a development point of view, looking at the APIs exposed by VMWare vSphere[9] it is clear that virtually every aspect of the vSphere operations can be controlled through an external software entity making authenticated API calls. From accessing and setting Alarms to obtaining overview and detailed views of any given VM Cluster, a complete software solution can be built to act as a manager for the vSphere client.

[8] http://www.vmware.com/files/images/products/vmw-dgrm-vcenter-operation-management-suite-amplifyvsph-lg.jpg

[9] https://www.vmware.com/support/developer/vc-sdk/visdk41pubs/ApiReference/

2.8.2 Xen Project Hypervisor

According to the project wiki page[10], the Xen Project Hypervisor is an open source baremetal/type-1 hypervisor, the only one of its kind. It can be used for a large range of projects dealing with: server virtualisation, Infrastructure as a Service (IaaS), desktop virtualisation, security applications, and embedded and hardware appliances[10].

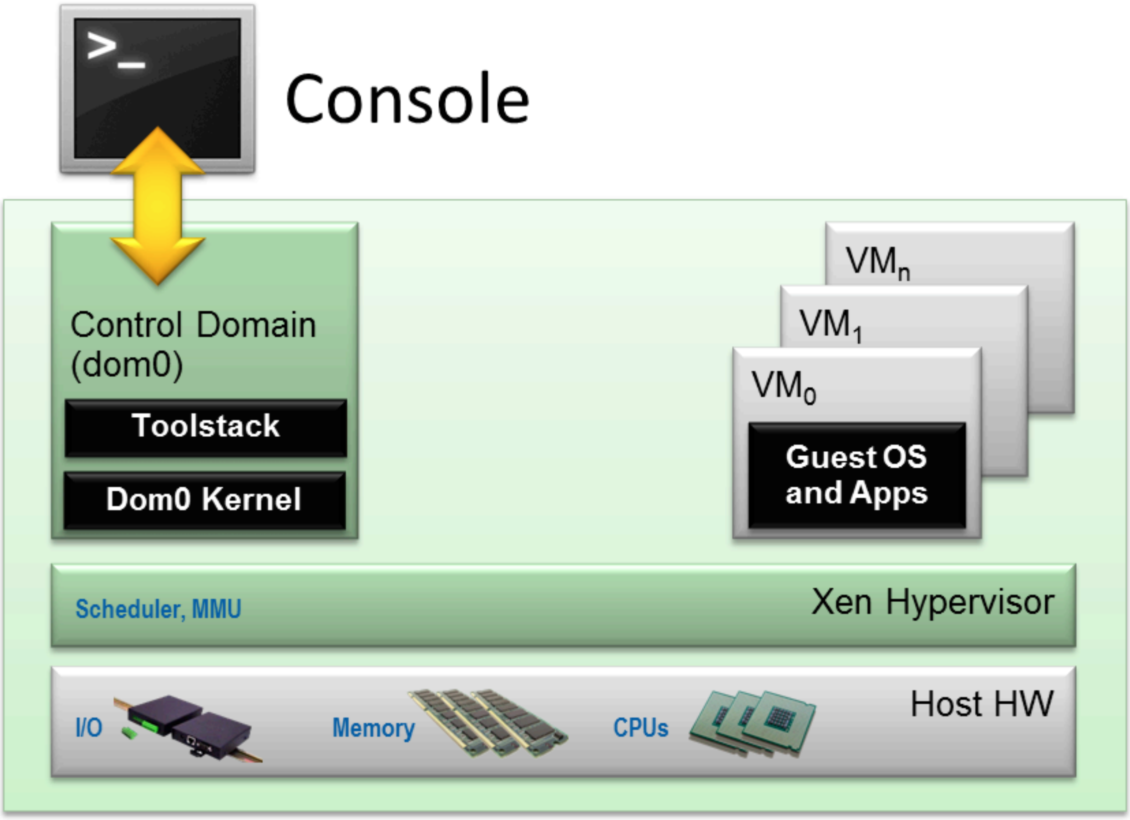


Figure 2 - Xen Project Architecture Diagram[11]

As can be seen in Figure 2 and according to the project wiki[12], the Xen Architecture is comprised of five different components:

- Xen Hypervisor – a very lean-written software layer (<150,000 lines of code) that runs directly on hardware and is responsible for managing CPUs, Memory, and interrupts, but with no knowledge of any networking of storage systems.
- Guest VMs and Apps – virtualised environments, each running their own operating systems and applications.

[10] http://wiki.xenproject.org/wiki/Xen_Overview#What_is_the_Xen_Project_Hypervisor.3F
[11] http://wiki.xenproject.org/mediawiki/images/6/63/Xen_Arch_Diagram.png
[12] http://wiki.xenproject.org/wiki/Xen_Overview#What_is_the_Xen_Project_Hypervisor.3F

- Control Domain – specialised VM with higher than normal privileges, capable of accessing hardware directly and interacting with other VMs.
- Toolstack and Console – the Control Domain contains a set of tools that allow the user to manage VM creation, destruction, and configuration.
- Virtualised Operating System – OS with Xen Project-enabled and ParaVirtualised-enabled kernels.

The Xen Project Hypervisor is a modern and fully featured Hypervisor capable of managing an unlimited number of VMs, hardware permitting. According to the project feature page[13], the latest version (4.5) can manage up to 4095 physical CPUs, both ARM and x86, and 16TB of physical RAM, which makes it more than suitable for projects of any scale.

Finally, from a development point of view, looking at the APIs exposed by XenServer[14] it appears that most day-to-day operations can be controlled through an external software entity making authenticated API calls.

2.8.3 VMWare vs. Xen Project Hypervisor – Why VMWare was chosen

Both VMWare and the Xen Project Hypervisor offer comprehensive tools meant to aid administrators in managing an extensive cloud infrastructure. The author found that both VMWare and Xen offer similarly capable APIs, essential in the initial design and development of the AMDS software, one of the key outputs of this research. Using either toolset allow for the development of a comprehensive software management solution.

Therefore, the reasons behind the author's choice are purely practical. The VMWare software suite was readily available through the academic institution being a VMWare Regional Academy. This allowed the author access to both the software and support on an on-going basis. Furthermore, the main output of this research is a generic software tool that offers platform-independent cloud management and, as such, either one of the two main cloud software providers would have made a good choice.

[13] http://wiki.xenproject.org/wiki/Xen_Project_Release_Features
[14] http://docs.vmd.citrix.com/XenServer/6.2.0/1.0/en_gb/sdk.html

2.9 Alternative Solutions to Cloud Management and Security Enhancement

The author compared the AMDS to other related academic works in the field taking into account security, reliability, data transfer speed, access times and power consumption efficiency. Logs produced by the AMDS facilitate the comparison.

2.9.1 Web-based Software Approach

Ramos-Paja et al. (2010) presents an integrated learning platform, intended for academic purposes, built to support Internet-based control-engineering education. This software approach to a multi-purpose engineering tool makes use of Web programming languages (PHP, HTML, JavaScript) and server software (Apache).

As can be seen in Figure 3, the authors' software approach is used as the link between internal systems (MATLAB) and the outside world (Web Clients). The approach is highly modular and supports distributed computing deployment infrastructures, such as a Cloud Computing environment with several Virtual Machines. This allows for easy reconfiguration and ensures high extensibility to accommodate future, potential problems in need of resolving.

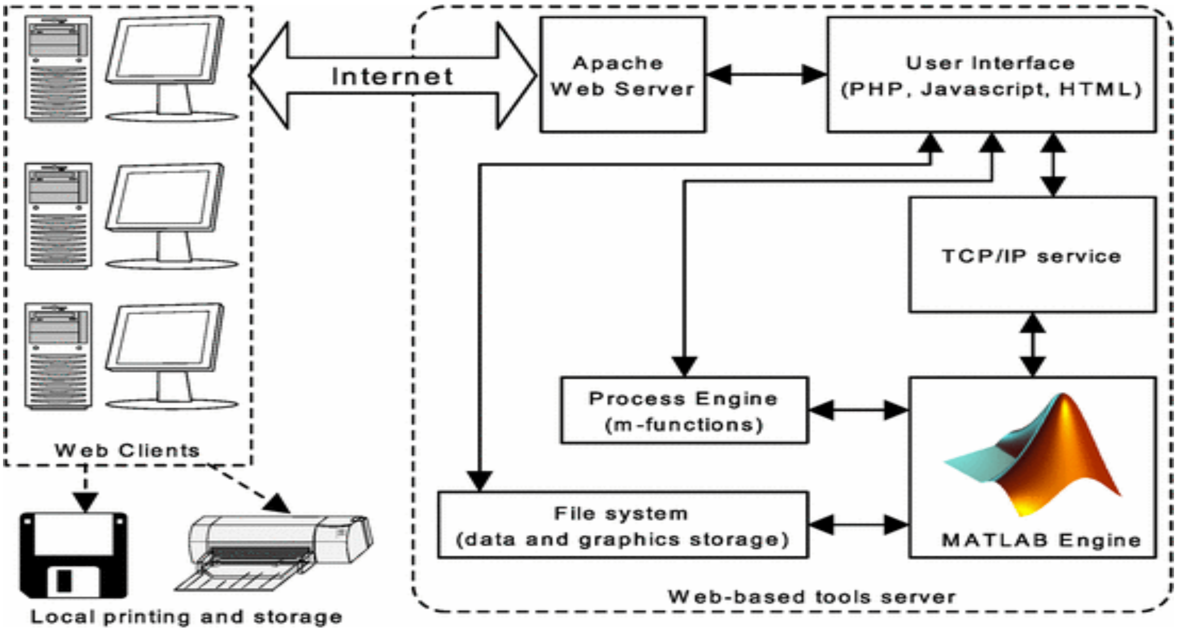

Internet
Apache Web Server
User Interface (PHP, Javascript, HTML)
TCP/IP service
Process Engine (m-functions)
File system (data and graphics storage)
MATLAB Engine
Web Clients
Local printing and storage
Web-based tools server

Figure 3 - Web tools structure - Analys and SimWeb (Ramos-Paja et al., 2010, Figure 6)

Potentially, this solution could be altered to interface with datacentre hypervisors or management clients, however there is not much information on how to extend it.

2.9.2 Reconfigurable Datacentres using Virtual Computing Laboratory (VCL)

In Vouk et al. (2009), the authors describe a full vendor-oriented implementation using an open source cloud platform called Virtual Computing Laboratory (VCL). The paper discusses how using an open source based system can be a viable alternative to, and even surpass in some cases, proprietary cloud solutions.

As can be seen in Figure 4, the entire solution is based on VCL. This approach consists of several VLANs (Virtual Local Area Network), each dealing with a particular part of the networked cluster (public access, network management, private network).

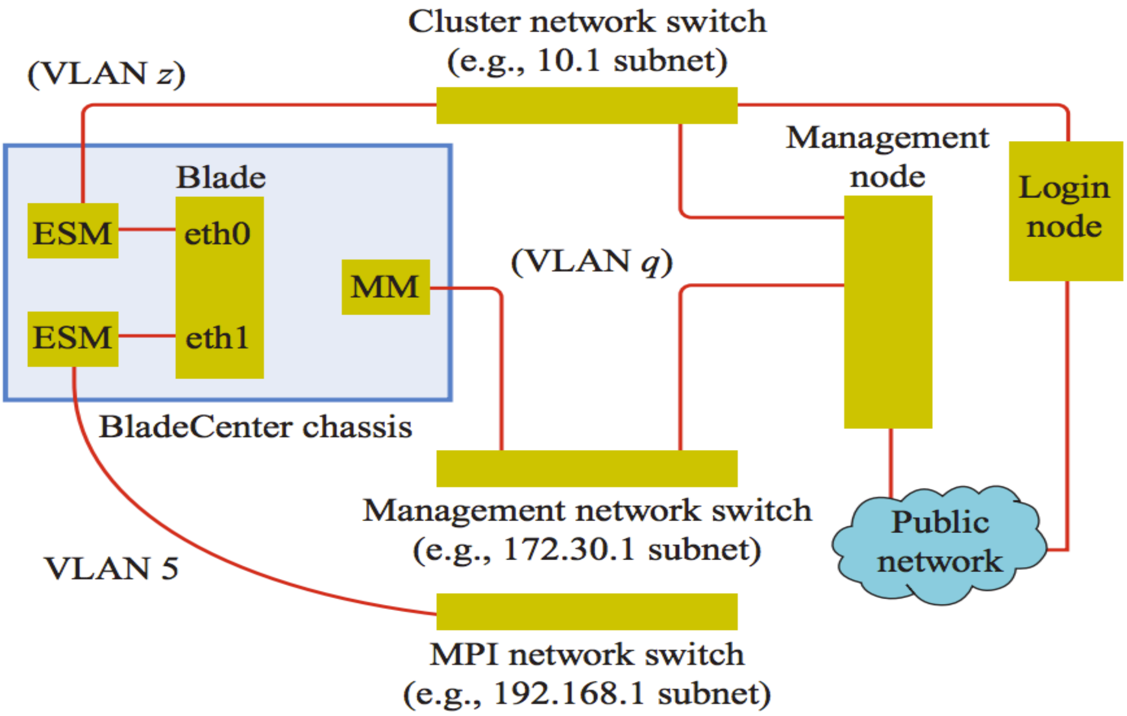


Figure 4 - VCL high-performance computing physical network setup (Vouk et al., 2009, Figure 4)

This is a similar solution to this thesis' author's VMWare test bed, however the results it is able to achieve are limited by the VCL features and limitations, whereas the AMDS is highly configurable and easily deployable onto any type of cloud infrastructure.

2.9.3 Dynamic, custom-built Hypervisor

In the approach taken by Tsutomu et al. (2010), the authors have created a custom Linux kernel module, intended to be loaded on the fly, in an attempt to improve runtime security and protect against buffer overflow attacks. This solution, called HyperShield, can be applied to, and removed from, running Linux OSes (operating system), as necessary.

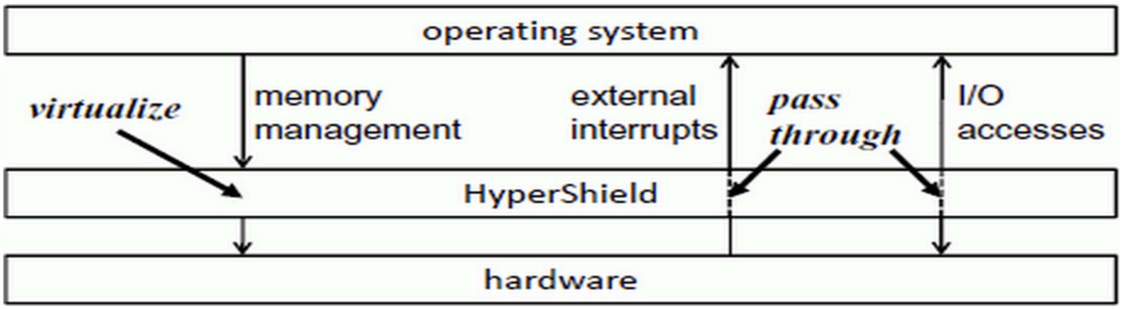


Figure 5 - Design of HyperShield (Tsutomu et al., 2010, Figure 3)

As can be seen in Figure 5, the authors have developed a software approach to securing a Linux based OS. This HyperShield, upon activation, would position itself between the underlying hardware and the operating system processes, practically virtualising the OS on the fly. This would ensure that all critical operations flow through a secure, stable layer rather than through the regular OS channels.

This approach, although a novel approach to securing a running OS, requires the operating system to reside on a physical server rather than a virtualised appliance. It continues to be a valuable solution, however it has a reduced impact on cloud infrastructures, unless it can be modified to operate within a VM.

2.9.4 A full-stack Cloud Management Solution

A different approach to the ones already discussed is presented by Feller et al. (2014). It proposes a full-stack Cloud Management solution called Snooze. This would be directly in charge of all hardware and virtual resources available within the cloud infrastructure it manages. In other words, it is an IaaS application similar in nature to the Xen Project and VMWare, but without some of the limitations.

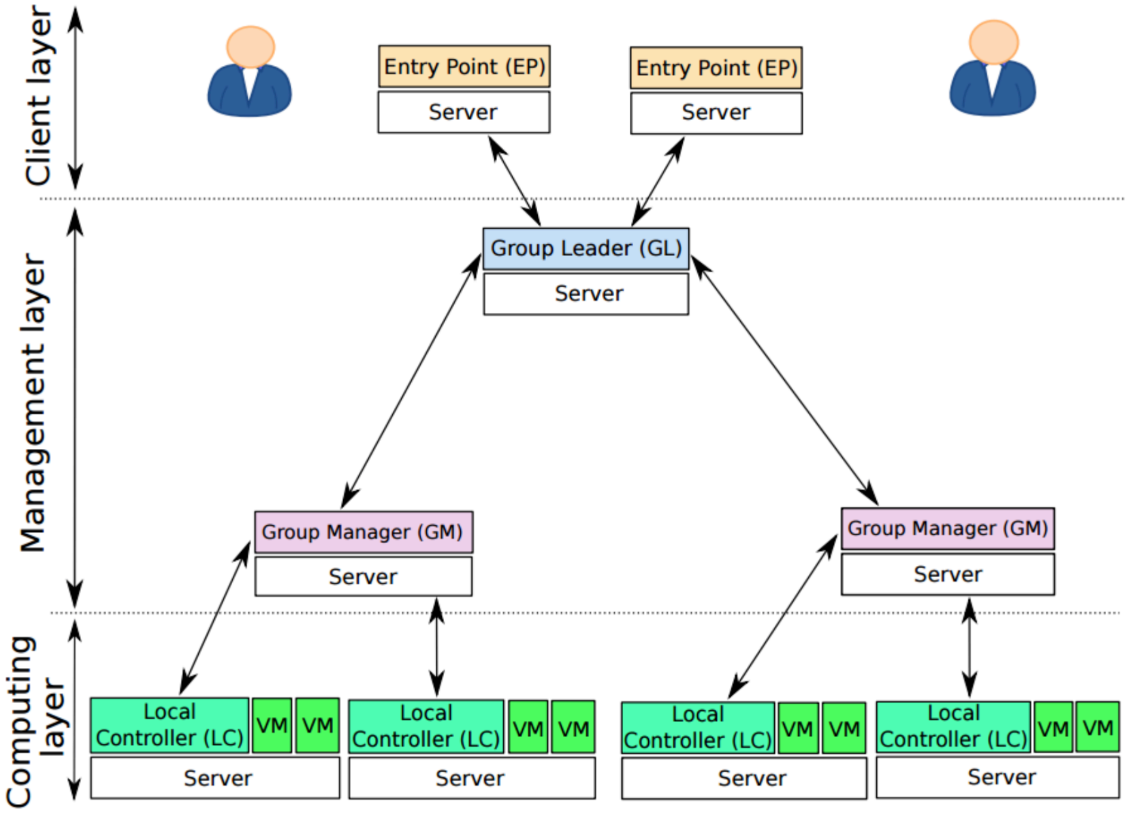


Figure 6 - Snooze high-level system architecture overview (Feller et al., 2014, Figure 1)

As can be seen in Figure 6, the proposed solution is highly flexible and scalable, capable of managing over 10,000 system services. The design is modular in nature, comprising clusters of masters and slaves, capable of managing cloud infrastructures across continents, given the appropriate network connections, which the authors assume to be available.

Overall this is a very capable solution, building on work done by many other similar solutions (OpenStack, OpenNebula, Nimbus, etc.), utilising all of their strengths and disregarding many of their drawbacks.

2.9.5 Queuing Methodology for Reducing Datacentre Power Consumption

The work by Fagci (2014) presents a client queuing methodology to be applied in a cloud environment with the aim to reduce overall power consumption. This is to be applied to existing cloud infrastructures as a VM management policy. The proposed approach is comprised of two algorithms in charge of maintaining several queues (logical clusters) of VMs, routing client requests on the fly to the least utilised cluster.

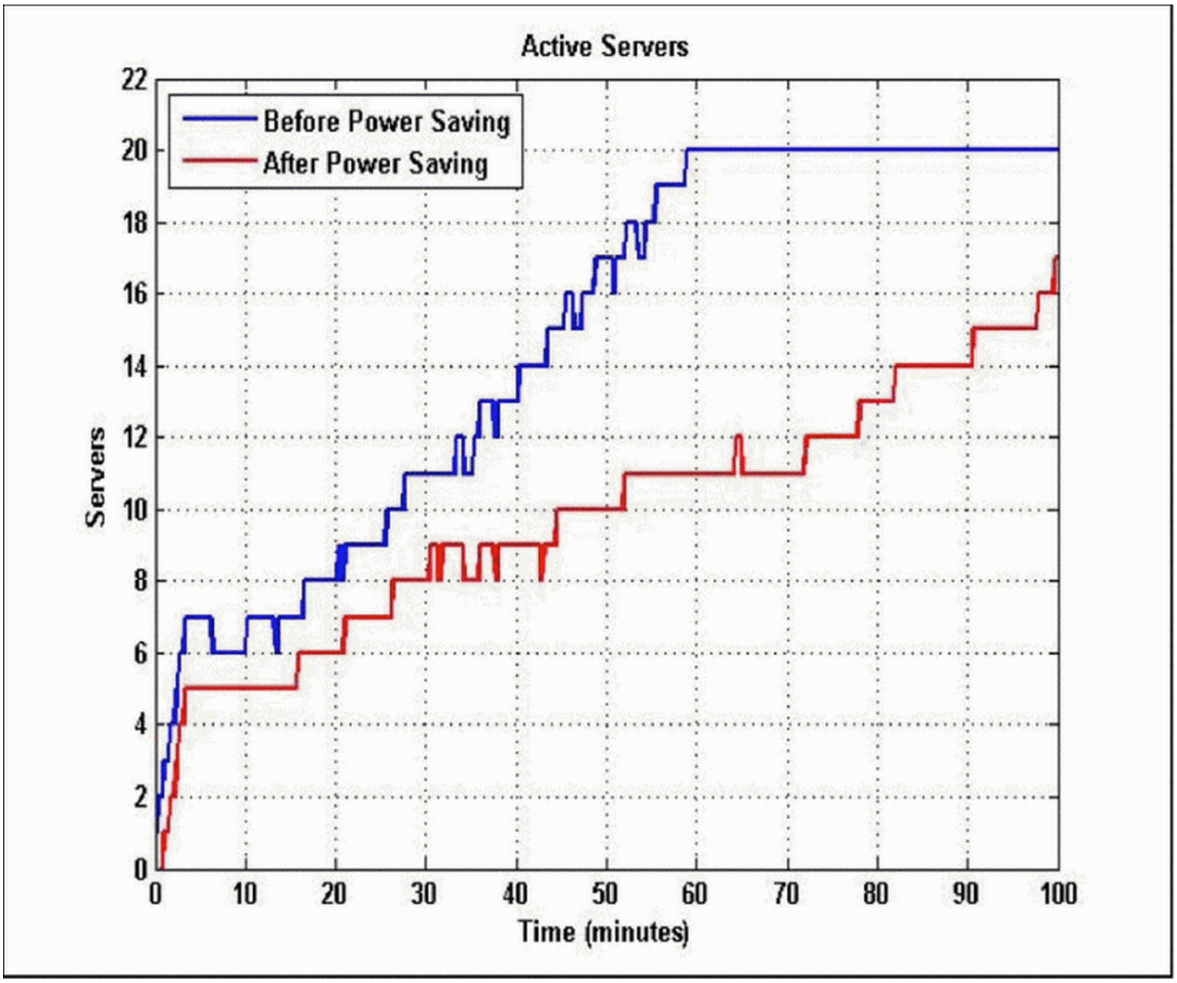


Figure 7 - Number of running servers before and after power saving algorithms (Bagci, 2014, Figure 4.1)

As can be seen in Figure 7, based on two algorithms discussed in the paper, the author has managed to reduce overall power consumption by up to 25%. This is achieved by employing different states as defined by the proposed solution, each with their own variables and configuration. The algorithms define stages when VMs would be added or removed from their managed queues.

Although initial results appear promising, the proposed solution heavily depends on management functionality being made available for achieving its tasks. Should any part of the required features not be available (e.g.: control over client request in real time; administrative power over VMs <startup, shutdown>), then the algorithms would become less than functional.

This concludes Chapter 2, A Review of State-of-the-Art in Cloud Computing. The Research Methodology chapter now follows which discusses steps undertaken towards answering the Research Scope and Objectives.

Chapter 3: Botnets and Botnet Monitoring Techniques

Chapter 3: Botnets and Botnet Monitoring Techniques

Some underlying Botnet theoretical principles used in the current research will now be presented, as well as some well-known Botnet detection techniques.

3.1 Botnets

The term *bot* is short for *robot*. Criminals distribute malicious software (also known as malware) that can turn any personal computer into a bot (also known as a zombie). When this occurs, the computer can perform automated tasks over the Internet, without the user's knowledge. Bots are typically used to infect large numbers of computers. These computers form a network, or a botnet, and are used to send out spam email messages, spread viruses, attack computers and servers, and commit other kinds of crime and fraud (Microsoft).

3.1.1 Botnet Communication Architectures

Peer-2-Peer bots are split into two categories: Masters and Slaves. Each of these uses secure channels to pass information between one another containing a command and the originating control module. This allows the Bot Masters to issue commands to the Bot Slaves. Once the control channel is established, it is used for communications that employ two different channel operation architectures, the *centralised architecture* and the *decentralised architecture* (Liu et al., 2008).

The *centralised architecture* has been used in the past by IRC-based (Internet Relay Chat, communication protocol) botnets, which used IRC servers to issue commands to all malware-infected machines. This mode has many different variations (Liu et al., 2008) e.g. the final destination could contain a text document with a list of static IP addresses and lists of URLs, so that flexible IP addresses can utilised.

The *decentralised architecture*, on the other hand, is a newer communication architecture that enables infected hosts to exchange information via distributed networks such as Peer-2-Peer. This method may reduce the rate of firewall and antivirus detection (Liu et al., 2008).

Regardless of the type of architecture, there are two types of command and control channels, the *Persistent Channel* and the *Periodic Channel* (Liu et al., 2008).

The *Persistent Channel* maintains a direct connection with the Bot Master. This connection type is normally employed by IRC bots. However, it is increasingly becoming obsolete due to periodic channels (Liu et al., 2008).

The *Periodic Channel*, on the other hand, connects to the Bot Master periodically in order to avoid detection. Typically, the destination has had no prior communication with the host. This is normally used to throw network security devices and probes off track (Liu et al., 2008).

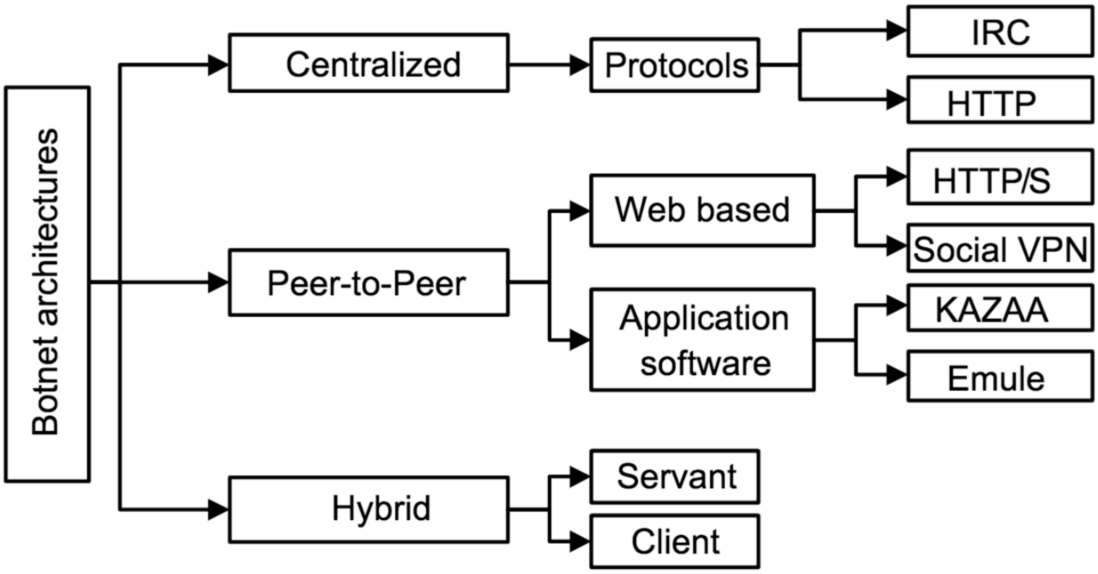


Figure 8 - Taxonomy of botnet architectures (Karim et al., 2014, Fig. 3)

Karim et al. (2014) also discusses the different architectures of botnets and also presents a taxonomy of botnet architectures, as can be seen in Figure 8. In addition to the already mentioned centralised and peer-to-peer architectures, Karim et al. (2014) also discusses a third type – the hybrid botnet, comprising of several servants and clients. “Servant bot acts as client and server simultaneously” and is configured with a static IP address (routable), whereas the client bot does not listen for incoming connections and is configured with a dynamic IP address (non-routable).

3.1.2 Botnet Exposure Techniques

Denial of Service (DoS) is a common cyber attack technique in which very a big network data burst is sent to the target (virtual) server cluster, typically much greater than the cluster network capacity. The result of this renders the target unable to respond to client requests, which makes it appear offline. This can be done either in a Cloud context or the outside world.

The outside world DoS consists of a network of physical Bots acting as one entity against a single server or a cluster of servers, consisting of physical hardware. The Cloud DoS consists of one or more infected VMs attacking other, one or more, VMs from within the same datacentre. These attacks are known as *Distributed Denial of Service (DDoS)* attacks.

Towards identifying and preventing such attacks, several detection and prevention techniques have been proposed.

The *counter-based* detection method (Lee et al., 2011) counts all client URL requests. It then compares them to the relatively frequent HTTP GET DDoS attack's URL requests, which tend to contain less data and display a higher rate. If these higher frequency requests are in large volume over a pre-set period of time, then the data packets they carry are dropped and logged for further analysis.

The *access pattern-based* detection method (Lee et al., 2011) looks at client behaviour to differentiate a legitimate client from an infected one. This is based on the assumption that a compromised client will, in most cases, operate in the same manner as the bot that originally infected it. All network data is compared against client profiles in an attempt to spot and block the compromised ones.

Zakarya et al. (2012) proposes a new Cloud Computing architecture that also features a mechanism for detecting and preventing DDoS attacks. The *entropy-based anomaly detection* method makes every data packet in a way that allows the client to trace its source. Any malicious data packet can be confirmed on a second attempt by a different or the same client node. Data is shared between clients and the detection mechanism, allowing for quick malicious data packet blocking. This method does, however, create many false positives.

Xu et al. (2012) presents a novel Peer-2-Peer botnet detection *min-vertex cover* method that makes use of both session-based analysis and minimum vertex cover theory when looking at data packets. This allows for only the packet header to be included in the security analysis, thus reducing computation overhead. It is able to learn from every malicious data packet, making its internal detection algorithm better over time. This method tends to yield positive results even if the bots change their behaviour and features.

At account creation, the Cloud vendor stores a template image of the client's VM. These images are considered to have high integrity and valid. Meena et al. (2012) presents a *file allocation table* Botnet detection technique that compares every application instance that is due to run against the initial VM template stored on the network. Any discrepancies found would immediately trigger an instance shutdown.

3.1.3 Botnet detection techniques

Karim et al. (2014) discusses several botnet detection techniques. It also presents a taxonomy of such techniques, as can be seen in Figure 9. It presents the following approaches:

- Honeynet – used to collect information on botnet activity for further analysis in an attempt to discover more on what technology the botnet is using, characteristics, and the intensity of the attack. However, this approach has several drawbacks: limited scalability; can only discover threats if directly interacting with them; can be taken over by attacker. (Karim et al., 2014)
- Intrusion Detection System (IDS) – "software application or hardware machine", used to monitor services for malicious activities and report findings to a management entity. Even though it contains signatures of several known botnets, it is not capable of coping with zero-day attacks and it also sometimes "ignore[s] identical bots with marginally different signatures". (Karim et al., 2014)

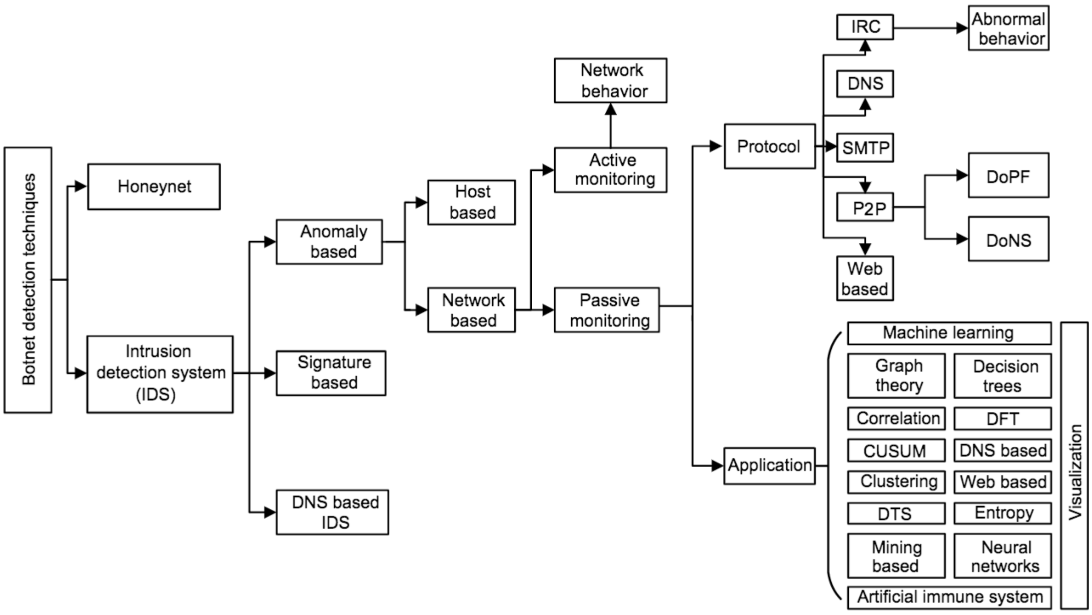


Figure 9 - Taxonomy of botnet detection techniques (Karim et al., 2014, Fig. 4)

Based on the detection techniques described by Karim et al. (2014) and Xiaobo et al. (2010), the author of this work has chosen to base the initial Botnet Detection module design on the IRC based anomaly detection approach. This allows for a quick implementation of a proof of concept application that would server as a platform for future developments.

3.2 Botnet Monitoring Techniques and Tools for Use in a Virtual Environment

This section describes various monitoring techniques and tools that could possibly be used to detect botnets in virtual environments. Bot masters will use root-kits and anti-VM static- DLL/binary code (Oh et. al., 2010) along with secure channels in order to avoid detection.

3.2.1 NetFlow based Traffic Monitoring

According to Cisco[15], NetFlow has been developed in-house initially to provide improved packet switching capabilities to some of their network devices in 1990. It

[15] http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html

then has steadily grown into a more complex tool used in network operational analysis.

As a network monitoring tool, NetFlow is capable of providing maximum network awareness, and, if used as part of a complex cloud infrastructure, is capable of giving insight into the different types of data packets flowing through the network at any given time. It makes use of 5 to 7 IP packet attributes to generate traffic flow reports, which can be later on analysed by system administrators or, for this current research, by the AMDS.

NetFlow utilises the following information in its traffic flow caches: packet size, IP addresses and ports (for both packet source and destination), class of service, device interface, and protocol type. In addition, it also records flow timestamps, next hop IP address, subnet mask, and TCP flags. All of these flow parameters aim to provide a holistic network view used in many different scenarios, of which the one of interest is detecting malicious behaviour in a cloud network infrastructure.

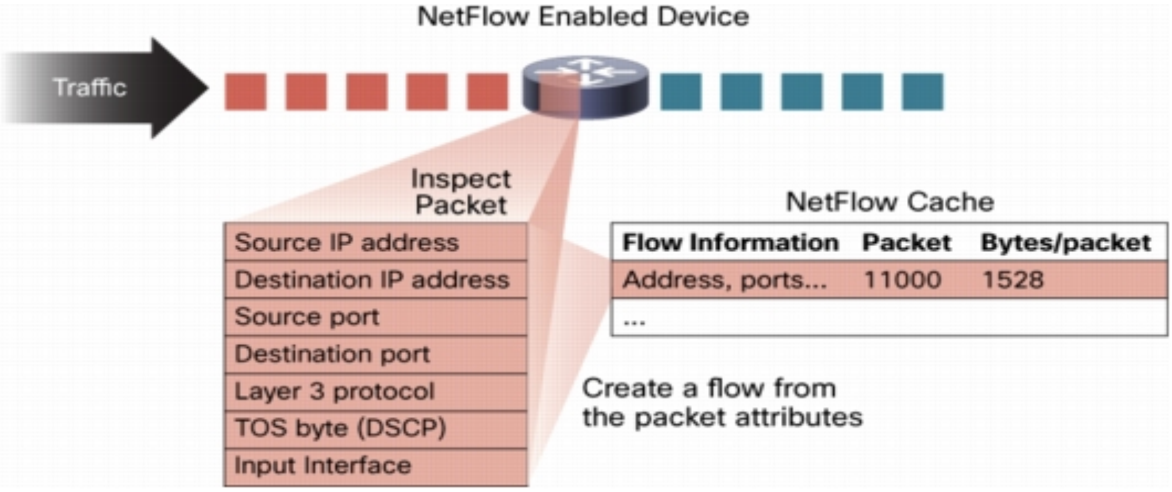


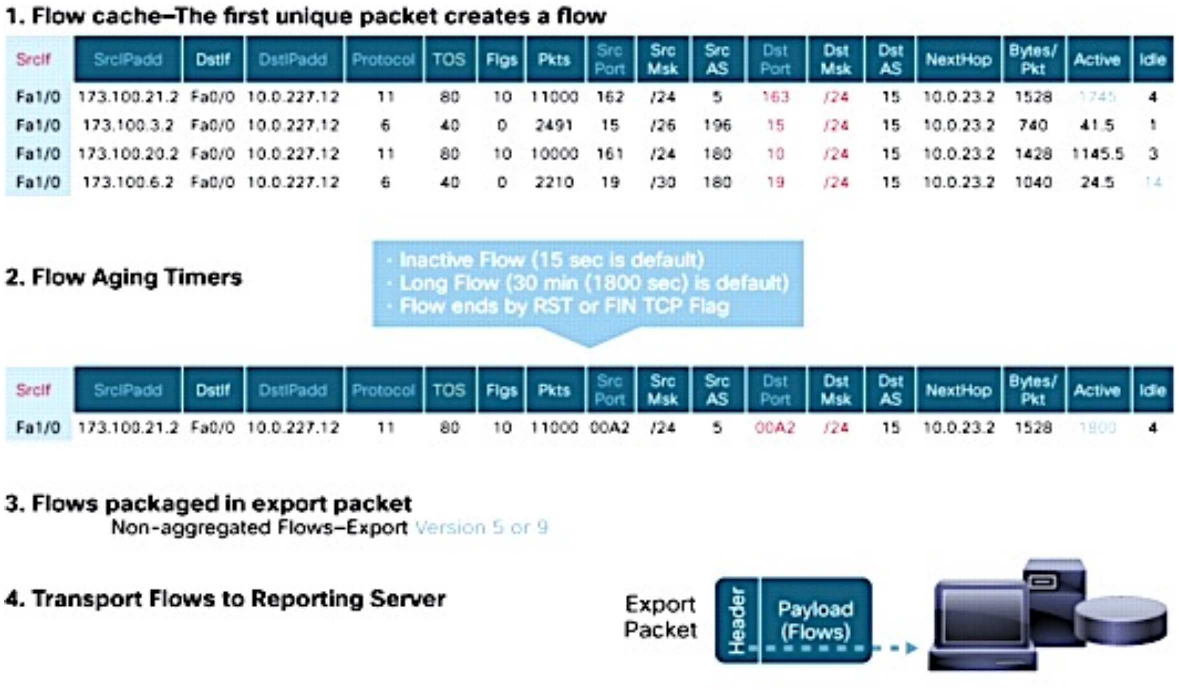
Figure 10 - NetFlow Flow Cache Generation[16]

As can be seen in Figure 10, upon enabling a device to utilise NetFlow, it then proceeds with recording all traffic coming into the network. These recordings, after undergoing NetFlow processing, are stored in a database entitled NetFlow Cache in the form of a table, each row containing one data flow cache. These flows have a shorter or longer lifetime as determined by the flow timestamp, depending on

[16] http://www.cisco.com/c/dam/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.doc/_jcr_content/renditions/prod_white_paper0900aecd80406232-1.jpg

whether traffic has stopped flowing, a FIN signal has been received indicating the end of the flow, or they have exceeded their pre-set lifetime.

The *NetFlow Enabled Device* (Figure 10) is, in this case, the NetFlow Exporter. Its primary concern is capturing traffic data and transforming it into flows. This data is then transferred over to a *NetFlow Collector Database Flow Cache* (Figure 11) where it is analysed and converted into meaningful reports that present an overview of the processed network data traffic.



1. Flow cache–The first unique packet creates a flow

SrcIf	SrcIPadd	DstIf	DstIPadd	Protocol	TOS	Flgs	Pkts	Src Port	Src Msk	Src AS	Dst Port	Dst Msk	Dst AS	NextHop	Bytes/ Pkt	Active	Idle
Fa1/0	173.100.21.2	Fa0/0	10.0.227.12	11	80	10	11000	162	/24	5	163	/24	15	10.0.23.2	1528	1745	4
Fa1/0	173.100.3.2	Fa0/0	10.0.227.12	6	40	0	2491	15	/26	196	15	/24	15	10.0.23.2	740	41.5	1
Fa1/0	173.100.20.2	Fa0/0	10.0.227.12	11	80	10	10000	161	/24	180	10	/24	15	10.0.23.2	1428	1145.5	3
Fa1/0	173.100.6.2	Fa0/0	10.0.227.12	6	40	0	2210	19	/30	180	19	/24	15	10.0.23.2	1040	24.5	14

2. Flow Aging Timers

- Inactive Flow (15 sec is default)
- Long Flow (30 min (1800 sec) is default)
- Flow ends by RST or FIN TCP Flag

SrcIf	SrcIPadd	DstIf	DstIPadd	Protocol	TOS	Flgs	Pkts	Src Port	Src Msk	Src AS	Dst Port	Dst Msk	Dst AS	NextHop	Bytes/ Pkt	Active	Idle
Fa1/0	173.100.21.2	Fa0/0	10.0.227.12	11	80	10	11000	00A2	/24	5	00A2	/24	15	10.0.23.2	1528	1800	4

3. Flows packaged in export packet
Non-aggregated Flows–Export Version 5 or 9

4. Transport Flows to Reporting Server

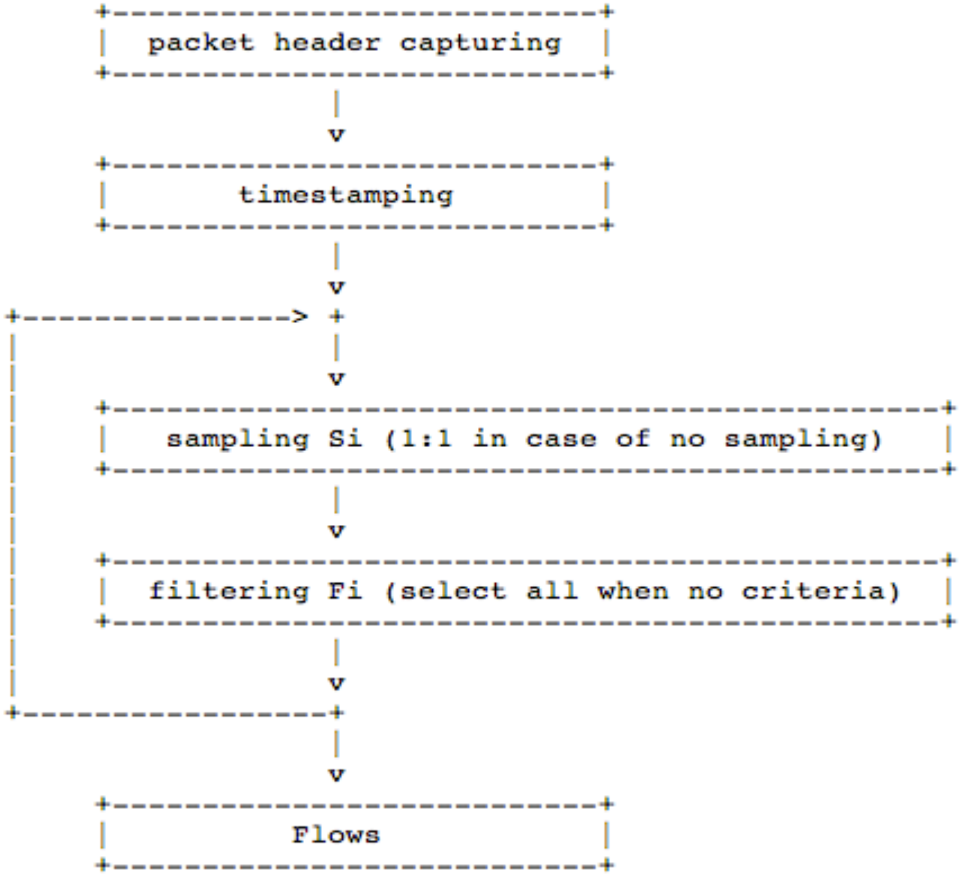
Figure 11 - Example NetFlow Cache[17]

As can be seen in Figure 11, a typical NetFlow Cache database table contains many different columns generated based on parameters both taken from the packet itself and created by NetFlow in its flow generation. It keeps track of both new and old flows by separating them into two different tables: one for active flows, another for ended flows (either because they had exceeded their pre-set lifetime or the traffic had stopped).

[17] http://www.cisco.com/c/dam/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.doc/_jcr_content/renditions/prod_white_paper0900aecd80406232-2.jpg

3.2.2 IPFIX based Traffic Monitoring

The IP Flow Information Export (IPFIX) format is an Internet Engineering Task Force (IETF) protocol designed to be a universally accepted standard for capturing traffic flow through network devices. IPFIX is responsible for defining the format of captured flows as well as detailing the flow method of transfer between the exporter and collector.



```
          +---------------------------+
          |  packet header capturing  |
          +---------------------------+
                       |
                       v
          +---------------------------+
          |       timestamping        |
          +---------------------------+
                       |
                       v
     +--------------> +
     |                 |
     |                 v
     |    +----------------------------------------------+
     |    |  sampling Si (1:1 in case of no sampling)   |
     |    +----------------------------------------------+
     |                 |
     |                 v
     |    +----------------------------------------------+
     |    |  filtering Fi (select all when no criteria) |
     |    +----------------------------------------------+
     |                 |
     |                 v
     +-----------------+
                       |
                       v
          +---------------------------+
          |           Flows           |
          +---------------------------+
```

Figure 12 - IPFIX Packet Selection Criteria[18]

IPFIX, similarly to NetFlow, is capable of breaking down data packets within the observed network traffic according to their attributes. As such, the flow metering process offers several configuration possibilities, which can be used to narrow down the packets included in the flow. There can be defined different sampling and filtering functions, each of which is dealing with a certain aspect of data selection.

[18] https://tools.ietf.org/html/draft-ietf-ipfix-architecture-12

As can be seen in Figure 12, the packet selection process happens immediately after a packet has had its header (addressing and destination information) examined and a timestamp assigned.

First, the sampling functions are applied that determine whether the packet needs to be included in the flow generation process through straightforward selection preferences. A sample function' duty might be to only select every 100th packet. If there is no sampling function defined, then all packets are included.

Next, the filtering functions are applied that determine whether a packet needs to be included based on selection patterns applied to its attributes. For example, a packet could only be included in the flow generation process if its associated protocol is TCP and destination port is less than 1024. If no filtering functions are defined, then all packets are included.

3.2.3 Netflow vs. IPFIX – Why Netflow was chosen

From the author's evaluation, there is little to no difference between Netflow and IPFIX. Netflow has been developed by Cisco and is a closed source tool, while IPFIX is a Netflow spin-off developed as an open source tool.

As such, the choice behind using Netflow as opposed to IPFIX is a purely practical one. The test bed comprises mainly Cisco switches and routers and, as such, it was logical to use a tool available by default on the test bed hardware.

In conclusion, this chapter has looked at what Botnets are and how they operate, as well as discuss several detection techniques, defence strategies, and software tools that could be used to detect and prevent malicious activity in a datacentre environment.

Chapter 4: Research Methodology

Chapter 4: Research Methodology

This chapter focuses on presenting the approach taken towards answering the Research Questions. It describes the Cloud Test Bed that was installed and configured by the author, from its initial design to implementation. It covers both the virtual (network and software configuration) and the physical aspects (hardware, physical parameters) of the infrastructure. It also describes the different approaches taken towards answering the Research Questions by providing an overview of:

- The Autonomous Management Distributed System (AMDS) including what techniques and technologies have been used in its development;
- The AMDS deployment strategy;
- The hijacking attempt identification strategy.

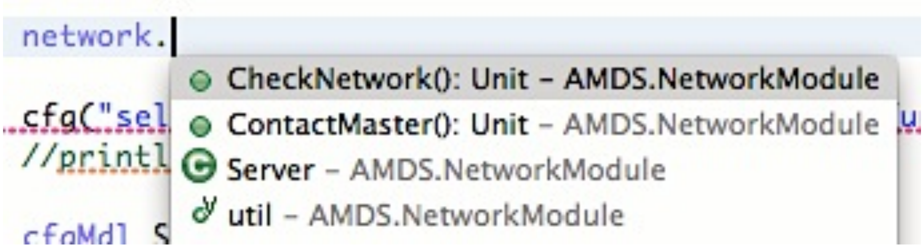
4.1 Software Package Review

The author has made extensive use of the NetBeans Integrated Development Environment (NetBeans IDE) throughout the AMDS development process. There are several reasons behind this decision, all of which will be presented further, with some information summarised from the Netbeans.org website.

NetBeans, originally called Xelfi, is an established software development environment, with its first release in 1997. It started as a student project, but over the course of two years, in 1997, it had become the basis for what today is a complex IDE used in creating applications using numerous programming languages, one of which utilised for the current research is Scala.

NetBeans is inherently a modular platform that offers a stable and solid ground for new features to be bolted on. These features range from small tools to fully fledged programming language support. Such programming support generally adds features such as syntax and semantic colouring, code folding, indentation and formatting, code completion and refactoring, error capture through pre-compilation, etc. The Scala NetBeans support used by the author in developing the AMDS is one such module.

One of the most critical features the Scala NetBeans module provides is code completion. This feature enables real time code suggestions based on code currently being typed. For example, in the AMDS context a part of the code that has been heavily utilised is the Network module. After instantiating a variable with the Network class (*val network = new NetworkModule(cfg)*), as can be seen in Figure 13, upon typing *network.*(dot) a list of available Network module methods pops up, which allows quick and easy code creation. This feature also pre-empts many programming bugs that could potentially be introduced due to human error at the time of code writing.



```
network.
```

CheckNetwork(): Unit – AMDS.NetworkModule
ContactMaster(): Unit – AMDS.NetworkModule
Server – AMDS.NetworkModule
util – AMDS.NetworkModule

Figure 13 - NetBeans Scala Code Completion

4.2 vSphere Java API Presentation

In order to effectively communicate with the VMWare infrastructure running on the author's test bed, a software package was selected to facilitate this need. Since vSphere is at the centre of all VMWare decisions, the author has chosen the vSphere Java API[19] for several reasons:

i. Fully supports the Object Oriented Programming (OOP) paradigm, which allows for extensible and maintainable software to be built.

ii. Being Java based, it can be easily integrated into the Scala project presented in this thesis.

iii. It has been specifically designed to support and work with vSphere, which helps reduce and, in some cases, entirely hide the complexity of interacting with the VMWare infrastructure.

iv. Allows for easy development of web interfaces capable of interfacing with the cloud network, while at the same time maintaining a very small system footprint.

```
package com.vmware.vim25.mo.samples;
```

[19] http://sourceforge.net/p/vijava/code/HEAD/tree/v5.1a/docs/Get%20started%20with%20VI%20Java%20API.pdf

```
import java.net.URL;
import com.vmware.vim25.*;
import com.vmware.vim25.mo.*;
import com.vmware.vim25.mo.util.*;
public class HelloVM {
     public static void main(String[] args) throws Exception {
          CommandLineParser clp = new CommandLineParser(new
               OptionSpec[]{}, args);
          String urlStr = “https://esx-server/sdk”;
          String username = "username";
          String password = "password";
          ServiceInstance si = new ServiceInstance
               (new URL(urlStr), username, password, true);

          InventoryNavigator(rootFolder).searchManagedEntities
               ("VirtualMachine")[0];
          Folder rootFolder = si.getRootFolder();
          VirtualMachine vm = (VirtualMachine) new
          VirtualMachineConfigInfo vminfo = vm.getConfig();
          VirtualMachineCapability vmc = vm.getCapability();
          System.out.println("Hello " + vm.getName());
          System.out.println("GuestOS: "
               + vminfo.getGuestFullName());
          System.out.println("Multiple snapshots supported: " +
          vmc.isMultipleSnapshotsSupported());
          si.getServerConnection().logout();
     }
}
```

Code Fragment 1 – Hello World VM interaction code[20]

As can be seen in Code Fragment 1, the vSphere API provides a straightforward method of interacting with the vSphere client. It allows for username and password authentication by supplying the *urlStr* (vCentre/ESXi HTTP address), *username* and *password* needed to reach and access vSphere. This particular example selects the first of all Virtual Machines present on the system and displays some basic information about it before terminating the connection.

The *ServiceInstance* seen in Code Fragment 1 is an object (OOP context). Once authenticated access to the vSphere instance has been established, it can:

i. Provide information about the server itself, such as configuration, properties and capabilities.
ii. Grant access to all information vSphere has direct access to, such as Virtual Machines, Virtual and physical Networks, system and network settings, events and alarms (*EventManager*), etc. through the *rootFolder* object, which contains information on all system entities.

[20] http://sourceforge.net/p/vijava/code/HEAD/tree/v5.1a/docs/Get%20started%20with%20VI%20Java%20API.pdf

iii. Allows for searching (*SearchIndex*) the vSphere controlled entities by different criteria and gain direct access and issue commands to and any one of them, such as a signalling a Virtual Machine to shut down:

Task task = ((VirtualMachine)me).powerOffVM_Task();

The VMWare environment is extensive, so in order to fully take advantage of the vSphere API the author made heavy use of the VI SDK API reference guide[21]. This provides information on and exact description of each available object and its properties and methods. Also, the vSphere API software package itself comes with several very useful code samples to help provide a starting point for any relevant project.

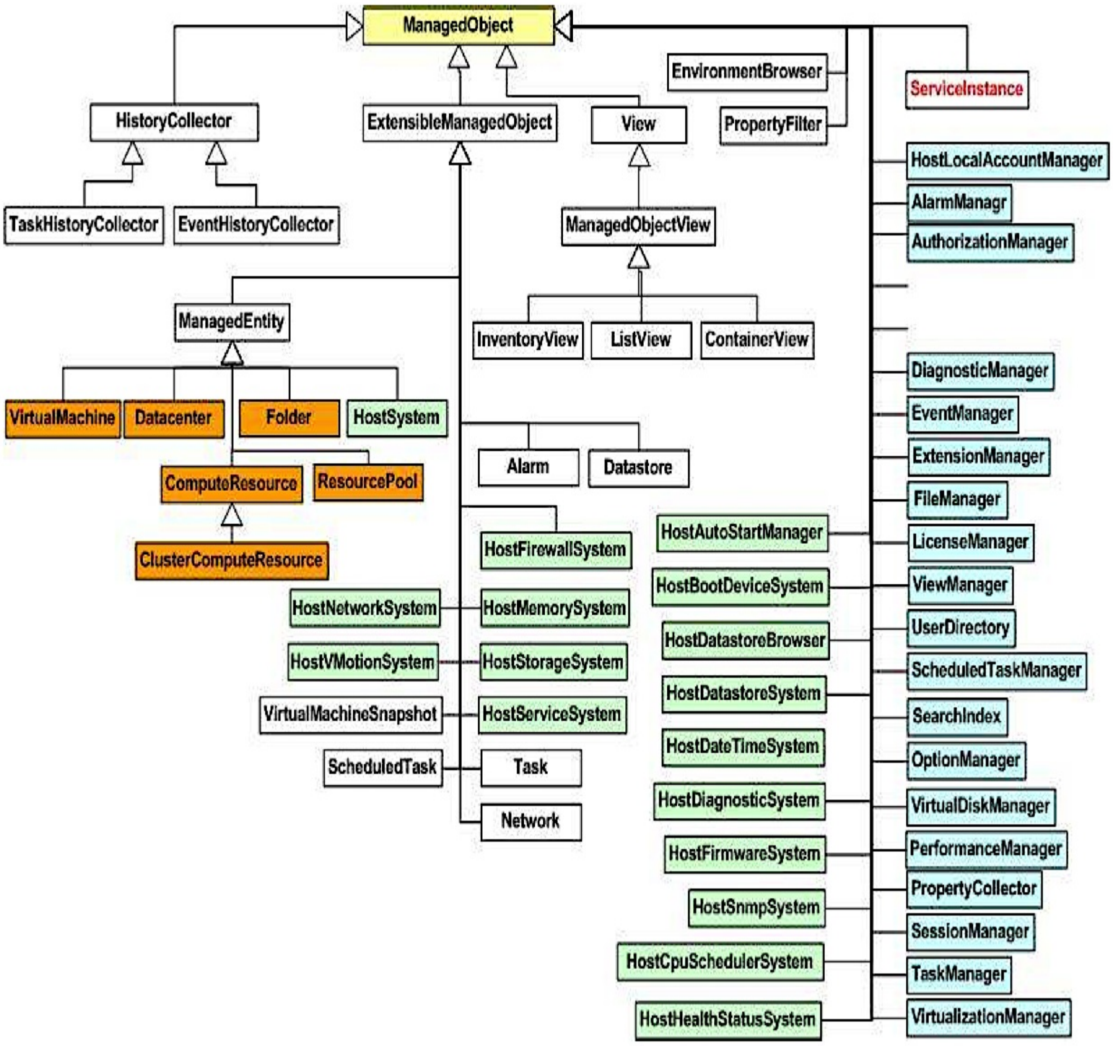


Figure 14 - Object model of VI Java API[22]

[21] https://www.vmware.com/support/developer/vc-sdk/visdk25pubs/ReferenceGuide/

[22] http://sourceforge.net/p/vijava/code/HEAD/tree/v5.1a/docs/Object%20model%20of%20VI%20Java%20API.pdf

The authors of the vSphere Java API package have provided an object model diagram of the VI Java API seen in Figure 14, which helps with understanding the VMWare infrastructure at an abstract software level. The *ServiceInstance*, which can also be seen in Code Fragment 1, is present on the far right in the diagram provides direct access to many different *Manager* objects that provide functionality for interacting with the different areas vSphere manages. The *ManagedEntity* class facilitates interaction with the different system entities found on the Virtual Interface client inside vSphere, such as Virtual Machines, the Host System, Resource Pool, Datacentre, etc. From here, the *HostSystem* class further provides support for accessing and managing many different key aspects of the host system, such as the Firewall, Memory, Storage, Network, etc.

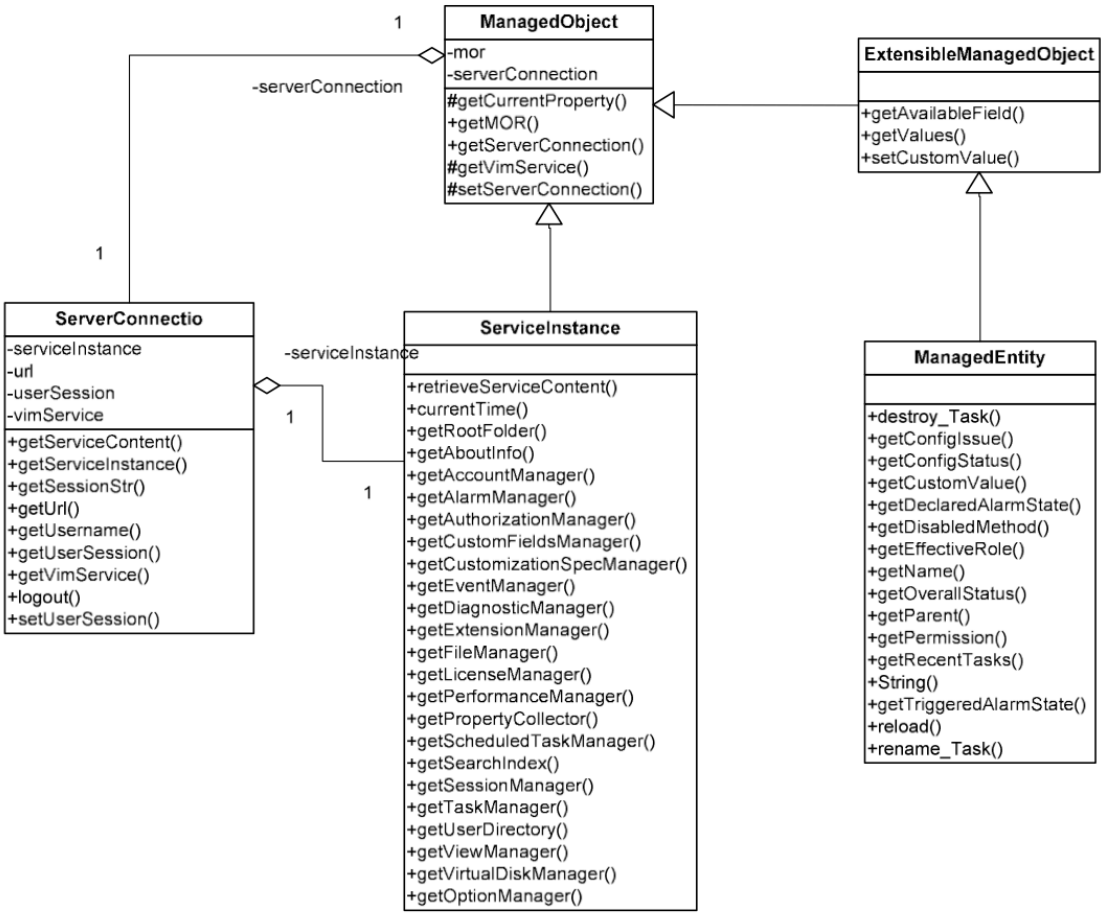


Figure 15 - Partial UML diagram of the VI Java API[23]

A UML diagram helps better understand the vSphere API software model (Figure 15). It provides detailed information on different functionality available as part of each major vSphere Object, seen in Figure 14.

[23] http://sourceforge.net/p/vijava/code/HEAD/tree/v5.1a/docs/Object%20model%20of%20VI%20Java%20API.pdf

The *ManagedObject* class makes use of the private properties *mor* (*ManagedObjectReference* – VI SDK managed object) and *serverConnection* (authenticated server connection) and several public methods to provide instant access to the mentioned entities.

The *ServerConnection* class contains information such as the *url* it used to contact the server, the *userSession* active session information that contains data such as the username used in the authentication process, as well as the *vimService* that provides functionality through over 300 methods.

The ServerConnection also contains a reference to the *ServiceInstance* object, which is the first of its type accessible to a developer. It facilitates the authentication process by taking in information such as the url, username, and password or a url and sessionID combination and creating a valid vSphere connection instance.

The ServiceInstance contains a *ServiceContent* object, which holds references to all manager type objects attached.

Finally, the *ManagedEntity* object is the most important of its type because it provides direct access to key vSphere entities such as the Virtual Machines, Host System, etc. It allows for searching the vSphere inventory and provides direct access to each search result entity.

4.3 Ubuntu Operating System Overview

The Ubuntu Server Operating System is a Linux system (also known as a 'distribution'), based on the Debian distribution, offering a secure environment for either production or development environments. It offers a wide range of software packages (tools, services, libraries, frameworks, systems) designed to be used to accomplish virtually any scenario. All packages are also generally open source, meaning they can be altered to suit any need, provided there is relevant programming knowledge, and also widely distributed without any legal implications.
Being based on the Debian distribution, an already highly efficient, stable and powerful Operating System, Ubuntu has been developed to take advantage of all the features and also incorporates numerous enhancements to help make system

development as streamlined as possible. This and the fact that it has datacentre support out of the box, makes it ideal for data processing intensive systems, such as the AMDS with its Control and Botnet modules, both running highly intensive operations. Ubuntu is very lean and optimised, so much so that its minimum system requirements are very low (192MB RAM, 300MHz single-core processor, 700MB storage), thus allowing more breathing space for other tasks running on it.

Ubuntu also supports a wide range of database engines and source code compilers. The ones that are of interest to the author are:

I. MongoDB, a NoSQL database engine capable of storing large amounts of data without any noticeable performance drop,

II. JavaVM, the service that powers all Java and Scala code and enables them to run across all existing platforms, and

III. Scala, a highly optimised programing language, with multi-thread management capabilities and simplified distributed application support out of the box.

Ubuntu is also very well supported, having certain Long Term Support editions (LTS) that receive updates for up to 5 years, thus making them very cost effective in the long run.

4.4 Adopted Software Engineering Methodology

An outline of the software methodology flow adopted by the author in designing, developing, deploying, testing, and maintaining the AMDS is now presented.

Figure 16 describes the workflow employed by this research in its approach to the Software component. It starts with designing, implementing and simulating components to achieve initial success confirmation. It then carries on with deployment and results collection and analysis. If this final process does not confirm initial simulation results, then the whole process starts over.

Conceptual Design is an objective of each and every activity, where issues relevant to the research are discovered. It describes desired features and operations in detail, including screen layouts, process diagrams, pseudocode and other documentation.

Software Development puts the Conceptual Design into code. Afterwards, Functional Analysis brings all the pieces together into a special testing environment, and then checks for errors, bugs and interoperability. Deploy to Hardware is the final stage of initial development, where the software is put into production. Testing is an important part of the software life cycle and it serves the purpose of discovering design flaws. Testing results are then analysed for design improvements and the whole process restarts.

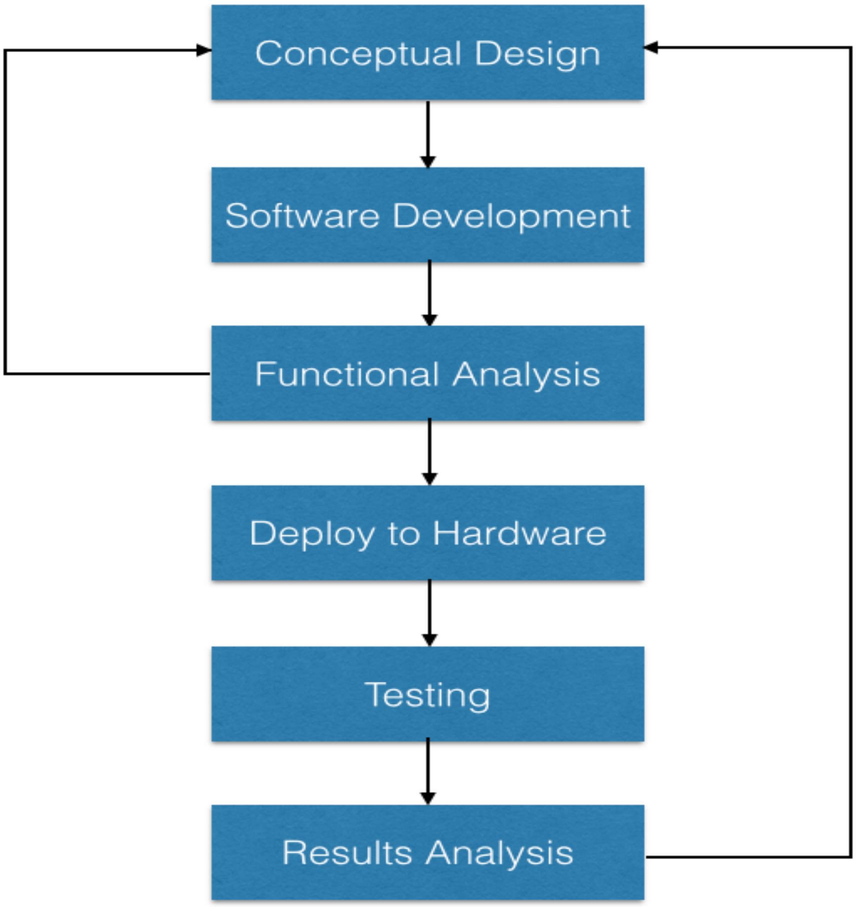


Figure 16 – AMDS Development Methodology Workflow

As part of *Objectives 1* and *4*, a novel 'Cloud-based' solution for teaching computer networks in an educational context has been configured. One key advantage of the system is its ability to commission and decommission virtual infrastructures comprised of routers, switches and virtual machines on demand. It makes use of hardware located in different physical locations, VMWare software to manage the virtual resources and NetLab+ to manage the configuration of multiple different virtual scenarios.

As part of *Objectives 2* and *3* an optimised and novel approach to an Autonomous Virtual Server Management System in a 'Cloud Computing' environment has been developed. One key advantage of this system is its ability to improve hardware power consumption through autonomously moving virtual servers around a network to balance out hardware loads. This has a potentially important impact on issues of sustainability with respect to both energy efficiency and economic viability. Another key advantage is the improvement of the overall end-user experience for services within the Cloud. This has been investigated through the configuration of the Cloud Computing test-bed rig.

4.5 Testing Principles

This section presents the underlying testing principles employed by the author in their work with the AMDS.

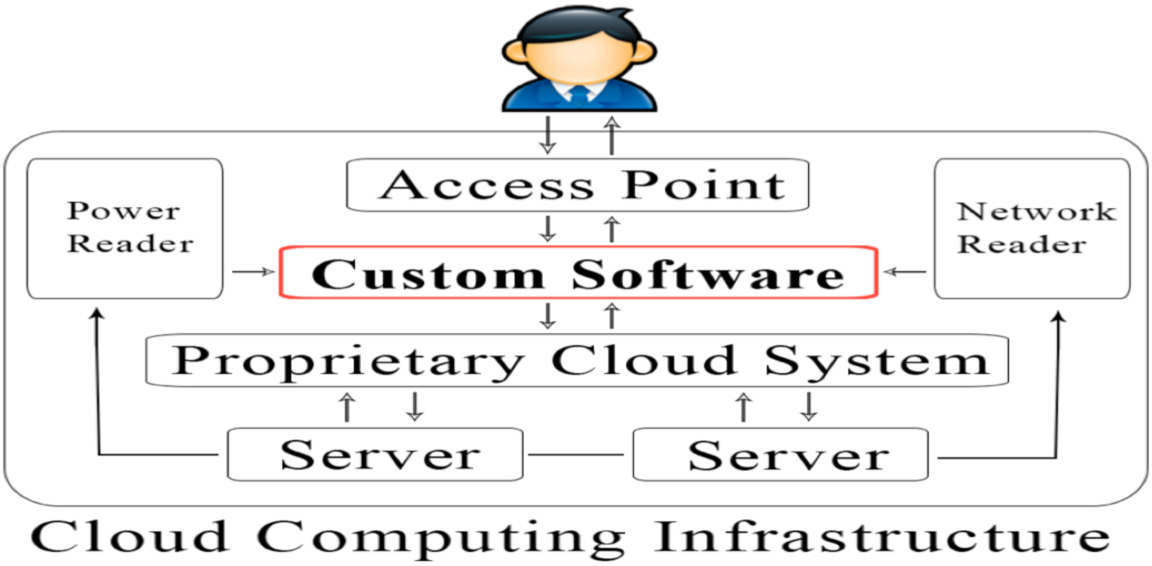


Figure 17 - AMDS Overview

Figure 17 presents an overview of the AMDS's position within a Cloud Computing environment. The AMDS connects to the four most important components of the cloud system: access point (connection to the outside world), power reading hardware (monitors power consumption), network reading hardware (monitors network flow), and the heart of the cloud system – the proprietary software (VMWare ESXi) that makes decisions regarding the server and storage management.

The AMDS is running from within a custom built Linux based Virtual Machine. The VM itself has direct access to the entire internal physical cloud network as well as a range of internal virtual networks that link up several other VMs together. This setup gives the AMDS the possibility of coordinating itself with other AMDS instances, allowing it to spread traffic load between them in time of high network usage.

The Autonomous Management Distributed System (AMDS) – The Software chapter will continue with presenting the design, development, and implementation approaches undertaken by the author in creating the AMDS.

Chapter 5: Autonomous Management Distributed System (AMDS) – The Software

Chapter 5: Autonomous Management Distributed System (AMDS) – The Software

This chapter focuses on the design, development, and the implementation of the AMDS.

In depth, it describes:

- the network optimization technique that serves as basis for the software implementation;
- the design of all system components and the relationship between them – analysis of the system design diagram;
- the development strategy, detailing the steps undertaken towards its implementation;
- the implementation process, detailing the choice in programming language and what specific features were utilised.

As part of *Objectives 3* and *5* presented in section 1.1, a novel modular design of an Autonomous Management Distributed System (AMDS) for Cloud Computing environments has been developed. It has been implemented using the Scala programming language because of its unique way of dealing with data structures, a topic more thoroughly discussed in the Programming Language Considerations subchapter.

The AMDS design considerations that underpinned the author's software approach will be presented.

5.1 AMDS Design Considerations

The AMDS has been designed from the ground up with distributed deployment, modularity and security in mind, using a full object oriented approach. A key feature of this system is the ability to gather and store information from various networking and monitoring devices from within the same computing cluster. Another key feature is the ability to intelligently control any number of Hypervisor instances (as long as an appropriate module is developed to enable communication) based on analysis of collected data and predefined parameters. The Hypervisor in turn, once it receives commands from the AMDS, proceeds to issue instructions to monitored servers in order to maximise energy efficiency, reduce the carbon footprint and minimize running costs.

5.2 AMDS Design Specifications

The AMDS was designed from the ground up with three main aims in mind: 1) security, 2) modularity, and 3) parallel processing.

To address security the author has based their design on existing work in order to introduce an appropriate authentication component. As such, each system component is required to authenticate each time it interacts with any other component.

Modularity has been achieved through implementing sound Object Oriented programming principles. Some core principles relevant to this work have been applied to the design. Every task the system is required to accomplish is split down into independent, fully reusable modules.

As can be seen in Figure 11, a typical NetFlow Cache database table contains many different columns generated based on parameters both taken from the packet itself and created by NetFlow in its flow generation. It keeps track of both new and old flows by separating them into two different tables: one for active flows, another for ended flows (either because they had exceeded their pre-set lifetime or the traffic had stopped).

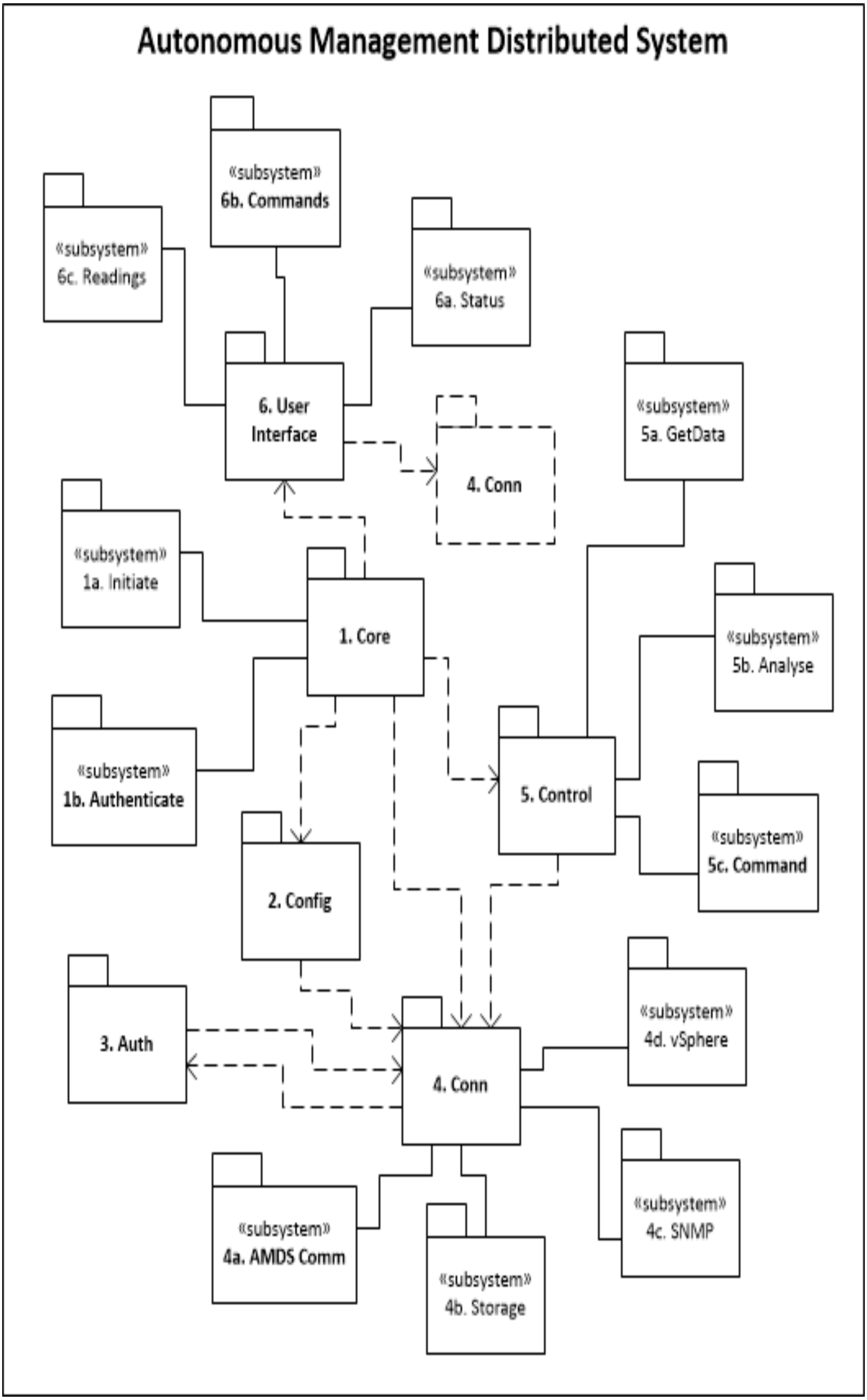


Figure 18 - UML Design Diagram of the AMDS

Figure 18 presents a UML diagram of the AMDS design. It is composed of 19 different parts, each designed for a specific task and fully reusable. The main features of these component parts will now be described:

1. ***Core:*** (Figure 18, 1.) Main system module. The entry point for the AMDS. From there the system accesses the configuration parameters (Figure 18, 2.) and starts its internal tasks (Figure 18, 5.) and the User Interface (Figure 18, 6.). It is responsible for facilitating communication between the different modules attached to it.
   a) ***Initiate:*** (Figure 18, 1a.) Module entry point. Achieves most of the module functionality. Initialises modules and establishes connections between them.
   b) ***Authenticate:*** (Figure 18, 1b.) Helper module. Undertakes the initial system authentication. This helps with detection of possible hijack attempts by making sure a current instance remains valid and genuine.

2. ***Config:*** (Figure 18, 2.) Helper module. Responsible for maintaining and providing access to the system configuration parameters. It interacts with the connection module (Figure 18, 4.) in order to gain access to the Storage component (Figure 18, 4b.), Config database. It is active throughout the lifespan of the system instance, facilitating on-the-fly parameter alteration.

3. ***Auth:*** (Figure 18, 3.) Key system module. Manages task and connection authentication. It performs checks against the initial determined instance validity and genuineness in an attempt to discover potential system hijack attempts and prevent them. It locks down any connection or task that does not pass the validation step and makes a log entry with relevant details on the security issue.

4. ***Conn:*** (Figure 18, 4.) Main system module. Facilitates all system connections between the modules themselves or between the modules and the computing cluster. It is the main access route to specialised mini-modules as well as attempt to authenticate each connection passing through it by calling upon the Auth module (Figure 18, 3.).
   a) ***AMDS Comm:*** (Figure 18, 4a.) Critical mini-module. Manages communication between instances of the AMDS including the passing of data between them. Acts as a load balancer by creating a bridge between current instance and one other instance. On each connection attempt it calls upon the Auth module to

verify the integrity of the outside instance before allowing any kind of information exchange.

b) ***Storage:*** (Figure 18, 4b.) Main mini-module. Keeps track of internal databases for each system module that deals with data. It stores information for the Config and Control modules.

c) ***SNMP:*** (Figure 18, 4c.) Specialised mini-module. Facilitates passing of information between current system instance and devices that understand the Simple Networking Management Protocol (SNMP). This protocol allows data to pass both ways, making it possible to issue commands and receive results between different devices that use it.

d) ***vSphere:*** (Figure 18, 4d.) Critical specialised mini-module. Interfaces the VMWare vSphere client to allow issuing commands and retrieving results. This module bridges the gap between the custom designed AMDS and the proprietary software solution provided by VMWare

5. ***Control:*** (Figure 18, 5.) Main module. Initiates data collection, storing and analysis tasks, as well as initiate commands to the vSphere Client through the Conn module (Figure 18, 4d.). This allows for data to be collected from monitoring devices across the computing cluster, stored, analysed and actions to be taken based on the results and the configuration parameters.

   a) ***GetData:*** (Figure 18, 5a.) Main mini-module. Deals with raw data retrieval. It initiates connections to the SNMP mini-module (Figure 18, 4c.), retrieves and stores collected information using the Storage mini-module (Figure 18, 4b.), Raw Data database.

   b) ***Analyse:*** (Figure 18, 5b.) Main mini-module. Retrieves chunks of raw data from the Storage mini-module (Figure 18, 4b.), Raw Data database, and come up with data capable of being compared to the configuration parameters. It then stores the analysis results using the same mini-module, but in the Results database.

   c) ***Command:*** (Figure 18, 5c.) Main mini-module. Compares analysis results with the configuration parameters and make intelligent decisions that maximise energy efficiency. After it stores issued commands in the Commands database, it then proceeds to send them to the correct interfacing mini-module from within the Conn module (Figure 18, 4.).

6. ***User Interface***: (Figure 18, 6.) Noncritical system module. Facilitates system

monitoring by presenting information stored on the system in human readable form.

a) ***Status:*** (Figure 18, 6a.) Main mini-module. Provides an overview of the current system state as well as global statistics, including access to security logs and top level information on database disk usage.

b) ***Readings:*** (Figure 18, 6b.) Main mini-module. Provides an in-depth view of each individual database currently utilised by the system instance. All databases maintained by the Conn module (Figure 18, 4.) are included. It makes use of data filtering and table display.

c) ***Commands:*** (Figure 18, 6c.) Main mini-module. Provides an in-depth view of all command decisions the current system instance has taken as well as the accompanying results received from all the commanded systems.

5.3 AMDS Implementation

This section discusses the necessary steps the author has undertaken towards implementing the AMDS. This process and its outcome both reflect the AMDS design presented in chapter 5.2.

5.3.1 Programming Language Considerations

In the implementation stage the author considered many programming languages capable of initiating remote connections, including Ruby, PHP, Java, Scala, C++, C#. The main criterion to be considered was portability i.e. make the system so that it can be deployed on as many different operating systems as possible. Only two of the considered languages met the required criteria: Java and Scala.

Java is an established programming language making its first appearance in 1995. It is able to function on any operating system running the Java Virtual Machine (JavaVM). All major systems, including Unix, Linux and Windows are currently capable of running the JavaVM. However whilst the language functionality continues to evolve it does not handle running multithreaded tasks well.

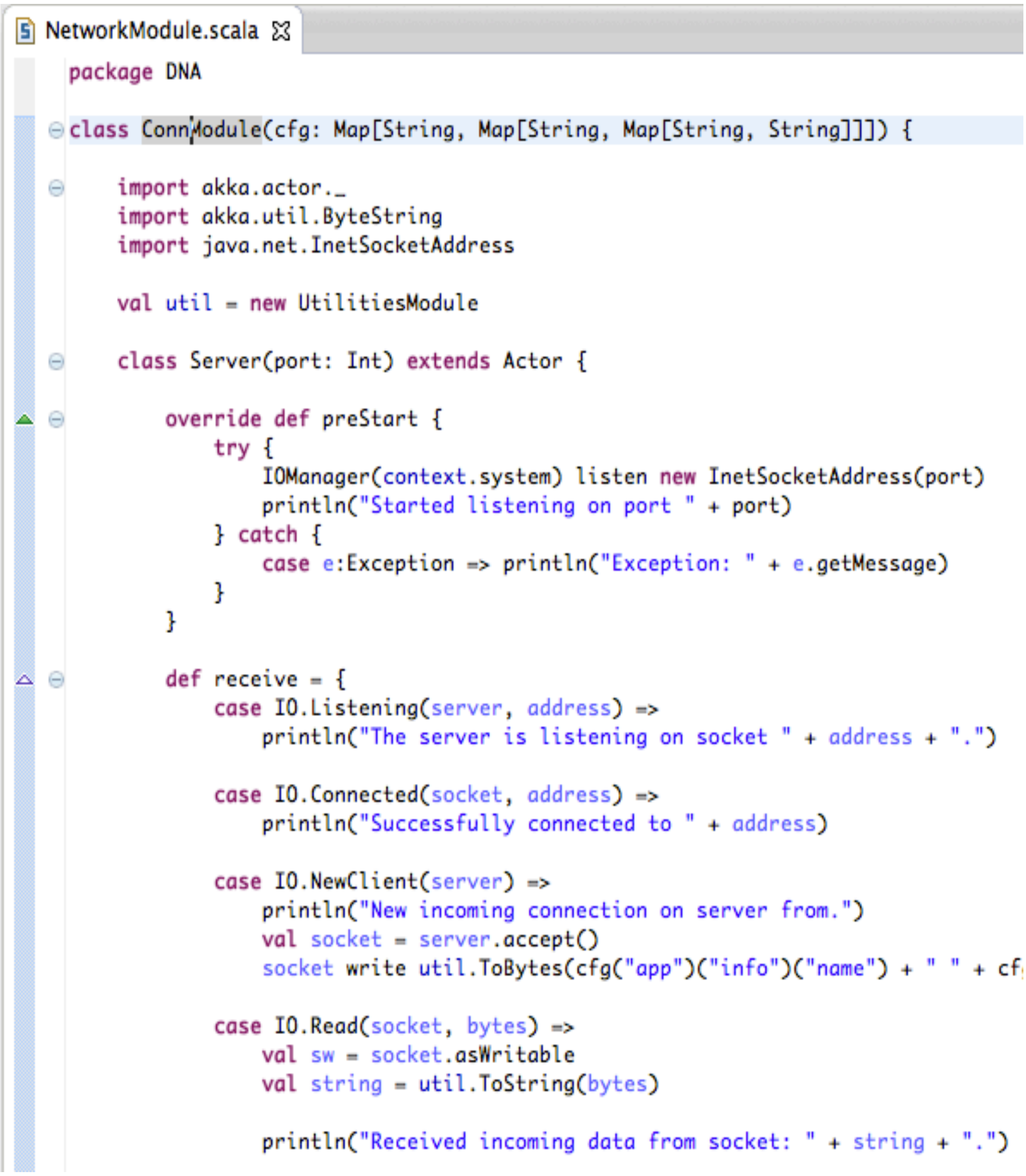
Scala is a relatively new language. In spite of only making its first appearance in 2003, it has grown in popularity very quickly due to its multithreading capabilities as well as its concise way of expressing common programming patterns. This has helped to drastically reduce development time on projects. One major advantage and the main reason why it has gained popularity so quickly is its seamless integration with Java. Java support in Scala, for example, can be provided by importing an appropriate library, and all Scala programs also run on the JavaVM, which means that these can be deployed on all the major systems.

5.3.2 Implementation Process

Scala has a unique way of dealing with data structures – there are mutable (can be changed – e.g. var) and immutable (cannot be changed – e.g. *val*) variable types. Scala creators recommend using the immutable types because this minimises the risk of random or unintentional data corruption during runtime.

Scala also allows for almost out-of-the-box distributed code implementations through the use of Actors. Scala Actors are capable of independent and asynchronous operation, operating by passing messages from one to another. They function under a command hierarchy and also allow for quick error recovery. Since each actor operates independently of each other, if one encounters a fatal error, the message is cascaded up the chain of command until it reaches an actor programmed to handle that type of issue. It can then proceed to take further actions as necessary e.g. restart the failed actor.

In the development process the author has used the Eclipse Integrated Development Environment (IDE) to assist with code completions and debugging as necessary. Each module has been implemented using inheritance based Object Oriented programming principles. Figure 19 shows a few lines of Scala code (part of the Conn module implementation). Java libraries have been used to facilitate remote connections and Actors have been used to operate as message transporters. The code in Figure 19 is set to receive remote connections and take different actions based on the type of result.



```
package DNA

class ConnModule(cfg: Map[String, Map[String, Map[String, String]]]) {

    import akka.actor._
    import akka.util.ByteString
    import java.net.InetSocketAddress

    val util = new UtilitiesModule

    class Server(port: Int) extends Actor {

        override def preStart {
            try {
                IOManager(context.system) listen new InetSocketAddress(port)
                println("Started listening on port " + port)
            } catch {
                case e:Exception => println("Exception: " + e.getMessage)
            }
        }

        def receive = {
            case IO.Listening(server, address) =>
                println("The server is listening on socket " + address + ".")

            case IO.Connected(socket, address) =>
                println("Successfully connected to " + address)

            case IO.NewClient(server) =>
                println("New incoming connection on server from.")
                val socket = server.accept()
                socket write util.ToBytes(cfg("app")("info")("name") + " " + cf

            case IO.Read(socket, bytes) =>
                val sw = socket.asWritable
                val string = util.ToString(bytes)

                println("Received incoming data from socket: " + string + ".")
```

Figure 19 - AMDS Part of Network Module written in Scala

```
{
      "config": {
            "self": {
                  "log": {
                        "decription": "Custom log function.",
                        "file": "logs/log_file.log"
                  },
                  "info": {
                        "is_master": true,
                        "address": "127.0.0.1",
                        "port": "23422"
                  }
            },
            "app": {
                  "info": {
                        "name":  "Autonomous  Management  Distributed
            System"
                  }
            },
            "master": {
                  "info": {
                        "address": "192.168.1.12",
                        "port": "23422"
                  }
            }
      }
}
```

Code Fragment 2 - AMDS configuration file JSON layout

The databases have been implemented using the MongoDB database storage engine. It is a NoSQL storage engine, which allows for big amounts of data to be stored without the need to account for or worry about data fragmentation or database design. As can be seen in Code Fragment 2, the MongoDB engine uses the standardised JSON data format to store information in different shapes and sizes. As such, one distinct advantage of this NoSQL engine is that data structure can change at any time and it will still store all information exactly the same, as long as there is at least a key reference found in each new piece of data added.

```
def connect() {

    import reactivemongo.api._
    import scala.concurrent.ExecutionContext.Implicits.global

    // gets an instance of the driver
    // (creates an actor system)
    val driver = new MongoDriver
    val connection = driver.connection(List("localhost"))

    // Gets a reference to the database "plugin"
    val db = connection("amds")

    // Gets a reference to the collection "acoll"
    // By default, you get a BSONCollection.
    val collection = db("config")

}
```

Code Fragment 3 - AMDS Scala MongoDB connection code[24]

In Code Fragment 3, the code used by the author to initiate a connection through Scala to the MongoDB database engine is presented. It makes use of the Scala Akka actor system to control the connection and, at the same time, provide fault tolerance at the highest level. The process is straightforward; all that is needed is the location (address) of the MongoDB server instance, along with the name of the database and tables of interest.

[24] http://reactivemongo.org/#step-by-step-example

```
import reactivemongo.api._
import reactivemongo.bson._
import scala.concurrent.ExecutionContext.Implicits.global

def listDocs() = {

      // Select only the documents that have an address of 127.0.0.1
      val query = BSONDocument("self.info.address" -> "127.0.0.1")
      // select only the field 'master'
      val filter = BSONDocument(
            "master" -> 1
      )

      /**
      * Let's run this query then enumerate the
      * response and print a readable
      * representation of each document in the response
      */
      collection.find(query, filter).cursor[BSONDocument]
            .enumerate().apply(
                  Iteratee.foreach {
                        doc =>
                              println("found document: " +
                              BSONDocument.pretty(doc))
                  }
            )

      // Or, the same with getting a list
      val futureList: Future[List[BSONDocument]] =
            collection.
            find(query, filter).
            cursor[BSONDocument].
            collect[List]()

      futureList.map { list =>
            list.foreach { doc =>
                  println("found document: " + BSONDocument.pretty(doc))
            }
      }

}
```

Code Fragment 4 - AMDS Scala MongoDB data retrieval code[25]

The author has chosen a MongoDB Scala framework named *ReactiveMongo* to help manage all database interactions. It provides scalable functionality capable of dealing with large data sets simultaneously, which keeps in tune with the author's AMDS system.

In Code Fragment 4, the method used by the author of this thesis to retrieve data from the MongoDB database is presented. The *find()* method takes a query object as a parameter, which it then uses to match all existing data against. This can return none, one or more results, depending on whether the sought after data already exists in the database table. In this case, the author is looking inside the *config* database for AMDS

[25] http://reactivemongo.org/#step-by-step-example

instances running on the IP address *127.0.0.1*. Next, a filter is used to only retrieve the *master* information of each row of data identified by the find() method. Finally, the query is then run against the collection (table) defined in Code Fragment 3 and all results, if any, are printed to screen for testing purposes.
In this particular case, this code is used by other AMDS instances to identify the AMDS master instance in order to establish a connection for load balancing purposes.

Another important component of the AMDS is the vSphere mini-module. In its development the author has made heavy use of the vSphere API for Java. This set of Application Programmable Interfaces (APIs) facilitates message exchanges between the Control module and the vSphere Clients currently in the computing cluster. Still in its preview stages, it allows for much interaction between third party programs and the vSphere client itself.

5.3.3 AMDS Virtual Machine Setup

The author has configured a Virtual Machine, going from design to implementation, using the Ubuntu Linux v12.10 Operating System distribution. It is configured using primarily default settings, however there are several customised software installations:

- Scala version 2.10.4.
    - It is used to execute the AMDS code and keep it running in the background.
- Customised system start-up scripts.
    - These are put in place to set up the needed environment for the AMDS. They are responsible for starting, maintaining, and checking the AMDS throughout its software lifetime.
    - They also check, maintain, and optimise the database and log files for the duration of the AMDS software lifetime.

The customised scripts have been created to ensure correct AMDS operation during its software lifetime, as follows:

```
#!/bin/bash

APPLICATION_PATH=/opt/amds
start() {
    echo -n "Starting"
    sudo start-stop-daemon --start --background --pidfile
           ${APPLICATION_PATH}/RUNNING_PID -d ${APPLICATION_PATH}
           --exec target/start -- -Dhttp.port=4200
    RETVAL=$?

    if [ $RETVAL -eq 0 ]; then
        echo " - Success"
    else
        echo " - Failure"
    fi
    echo
}
stop() {
    echo -n "Stopping"
    sudo start-stop-daemon --stop --pidfile
           ${APPLICATION_PATH}/RUNNING_PID
    RETVAL=$?
    if [ $RETVAL -eq 0 ]; then
        echo " - Success"
    else
        echo " - Failure"
    fi
    echo
}
  case "$1" in
    start)
      start
  ;;
    stop)
      stop
  ;;
    restart)
      stop
      start
  ;;
    *)
      echo "Usage: play-server {start|stop|restart}"
      exit 1
  ;;
esac
exit $RETVAL
```

Code Fragment 5 - AMDS Bash Management Script

Code Fragment 5, partly based on the script found here[26], presents the script **amds**, the AMDS instance management script. As can be seen, a typical bash process management script consists of two main functionality blocks:

- ***start()*** – deals with starting the process; and
- ***stop()*** – deals with gracefully stopping a process.

[26] http://www.whiteboardcoder.com/2013/06/creating-startup-script-for-your-play.html

The **start** function is in charge of making sure the AMDS has successfully started up and has begun its programmed tasks, such as reading configuration files, generating encryption certificate, initialising modules, etc. It then outputs a message according to whether the task has been a success or not. On the other hand, the **stop** function is in charge of making sure the AMDS has gracefully stopped its operations. It monitors the situation after initiating a shutdown and reports back on whether it has been successful or not.

The script shown in Code Fragment 5 also comes with two more functionality features as possible use cases:

- ***restart*** – runs the **stop** and **start** functions in sequence and reports back on these tasks' success; and
- *** - deals with incorrect user input when using the script, alerting the user when an invalid option has been requested.

```
#!/bin/bash
ps cax | grep amds > /dev/null
if [ $? -eq 0 ]; then
     echo "`date` - Process is running."
           >> /opt/amds/logs/operations.log
else
  service amds_script start
  echo "`date` - Process is was not running. It has been started."
     >> /opt/amds/logs/operations.log
fi
```

Code Fragment 6 - AMDS Bash status check script.

The script called **amds_check.sh** is used to check for the local AMDS instance and start the instance if it is not running. This script runs after the initial operating system boot-up sequence has completed. As can be seen in Code Fragment 6, the **amds_check.sh** script is in charge of checking whether the *amds* process is still running and performing its duties. In case it has unexpectedly stopped, a **start** option is attempted. Regardless of the process' state, this script saves current check status to an **operations.log** file for later viewing. This helps with debugging potential situations when the AMDS might inadvertently crash.

The scripts presented in Code Fragment 5 and Code Fragment 6 interact with each other in the case the AMDS crashes and it requires restarting. Then Code Fragment 6 makes use of Code Fragment 5 in an attempt to resume AMDS's operations.

5.3.4 AMDS Setup on the Virtual Machine

All AMDS Scala and Database files and folders are located in the folder **/opt/amds** on the Operating System. This is the location from where AMDS will carry out its operation throughout its software lifetime.

Initial AMDS configuration lies within the Database, containing operational constants as well default operation settings that can be internally changed throughout its software lifetime as more network and operation data is gathered.

5.3.5 Autonomous Virtual Machine Management

The author has created a Snapshot of the Virtual Machine once optimal operation was insured. It is stored directly on the Network Storage Drive and can be used at a later date as backup.

This Virtual Machine has then been converted into a Virtual Machine Template that is actively used by the VMWare vSphere Client to spawn and start up new Virtual Machines as network loads escalate.

The AMDS is responsible for monitoring network operations and relaying commands to the vSphere accordingly. In current context and based on network data flow, these commands can be one of the following:

- Use the AMDS VM Snapshot to spawn a new Virtual Machine and start it up.
- Graciously stop a particular AMDS Virtual Machine.

Stopping an AMDS instance is performed once the following steps have been taken:

1. Check that all the latest Database information has been synchronised successfully across all other currently running AMDS instances.
2. Check that the AMDS instance that is about to be shut down has no current standing operations. If it does, transfer all operational parameters relevant to those current operations over to the master AMDS instance. The master AMDS then allocates the jobs to other running AMDS instances or acts upon them itself if no other instances are available or other instances are currently busy.

This final stage ends the AMDS management procedure.

5.4 AMDS lifetime operational logic flow

The AMDS master instance is the first one to run within the network. It is responsible for checking and making sure its own internal operations are functioning at optimal efficiency as well as manage other slave instances.

As can be seen in Figure 20, upon start, the AMDS creates a security fingerprint that it will then use throughout its software lifetime to sign every decision it makes or action it takes. This design feature ensures that only authorised AMDS instances are able to issue commands. It helps prevent its own hijacking or interference from outside entities.

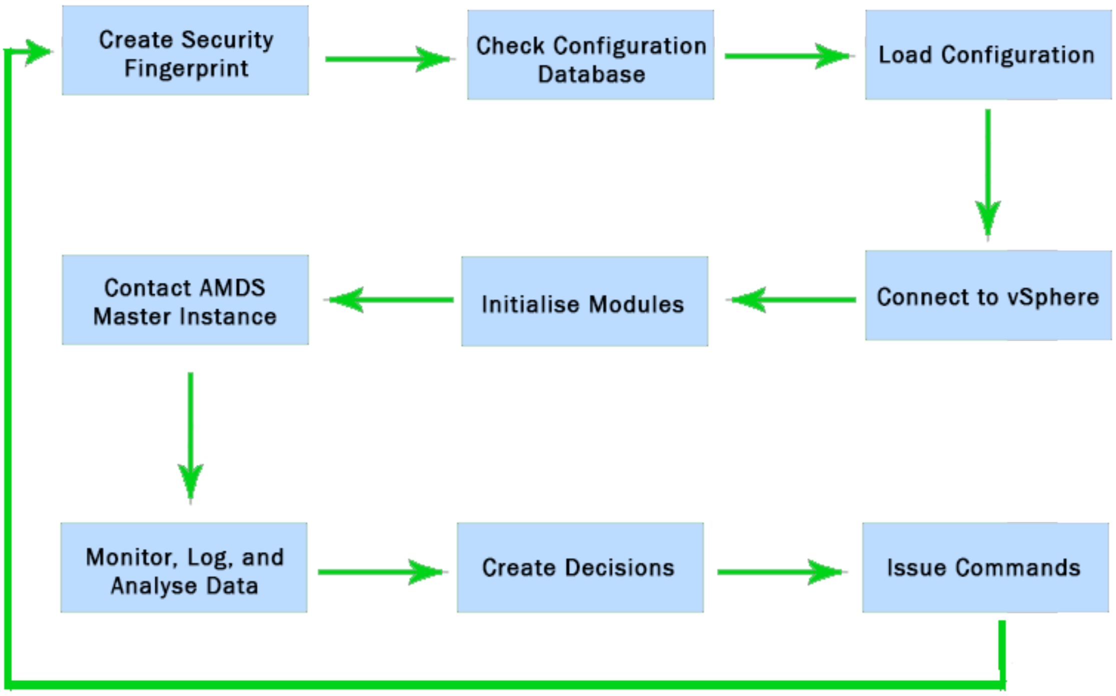

Create Security Fingerprint
Check Configuration Database
Load Configuration
Connect to vSphere
Initialise Modules
Contact AMDS Master Instance
Monitor, Log, and Analyse Data
Create Decisions
Issue Commands

Figure 20 - AMDS lifetime operational logic flow.

It then checks the configuration Database and loads its initial parameters from it. Once successful, it continues to initialise its internal modules. If no errors are encountered, it then proceeds with establishing connection to any other AMDS instances currently running on the network.

Once this start-up procedure has taken place without any issues, the AMDS then proceeds with establishing connections to any configured VMWare vSphere Client (or any other provider for which a connection module is present).

Finally, its internal modules begin their monitoring, logging, and analysing incoming network data and vSphere (or any other provider for which a connection module is present) status. Based on recorded data it proceeds with making optimisation decisions according to its internal algorithms and forwards them to the virtualisation clients.

Chapter 6: AMDS System Enhancement - Botnets

Chapter 6: AMDS System Enhancement – Botnets

This chapter makes use of the results analysis presented in Cloud Computing Test Bed – Software Deployment on Hardware by utilising them in the context of Network Hijacking. It presents a novel prevention strategy that has been implemented as a module of the AMDS, and discusses a proposed abstract software model for use in other computing environments.

6.1 Detection of Successful Network Hijacking Attempts – An Abstract Model

This section focuses on detailing the abstract hijacking detection software model developed by the author, in particular the botnet heuristic detection algorithm adaptation. The author has chosen to employ an anomaly-based heuristic algorithm based on the work carried out by (Binkley et al., 2006).

The algorithm consists of two main detection components: an Internet Relay Chat (IRC) mesh, and a Transmission Control Protocol (TCP) scan detection heuristic. The author has made use only of the TCP heuristic component and adapted it as necessary into the AMDS botnet detection module.

An algorithm requirement is that a scanner actively samples network flow and collects information on a percentage of observed data packets. This scanner would store, as a tuple set, the following data flow information: *IP source address*, *SYNS*, *SYNACKS*, *FINSSENT*, *FINSBACK*, *RESETS*, *PKTSSENT*, and *PKTSBACK*.

The IP address serves as a logical key for the entire stored packets waiting to be analysed. The SYNS are counts of SYN packets, while SYNACKS are SYN packets sent with the ACK flag set. FINS sent both are counted, and RESETS are only counted when sent back to the IP source. The PKTSSENT represents the total number of packets sent by the IP source, and the PKTSBACK is the total number of packets sent back to the IP source. This information helps in providing an approximate network-based indication of the kind of exploit in use at the time (Binkley et al., 2006).

$$w = (S_S + F_S + R_r)/T_{sr}$$

Equation 1 - TCP work weight (percentage) (Binkley et al., 2006)

The algorithm makes use of a metric defined by Equation 1, also called a work weight, expressed in percentage, which when calculated as being close to 100% generally indicates the presence of an anomaly. This percentage is calculated by dividing the sum of SYNS and SYNACKS (S_s), FINS (F_s), and RESETS (R_r) to the total number of packets (T_{sr}). This is intended to be used as a tool by the analysing entity to create a ranked list of possible botnet activities.

Another detection algorithm the author has based his work on is discussed by (Xiaobo et al., 2010), which presented a TCP Botnet detection algorithm through measuring the "quasi-periodicity degree and packet average size of IRC [type] conversations based on ukkonen algorithm".

The ukkonen algorithm traverses a given character string in steps, going through it from left to right, one step per character, with each step potentially being composed of several operations. Its end goal is to deconstruct the string into a series of character suffixes.

As presented by the authors, the algorithm only analyses a group of packets' size, defined as a "Conversation Content Sequence (CCS)" and viewed as a string of characters; it attempts to employ the ukkonen algorithm to search for re-duplicate substrings within the communication flows. Each CCS consists of 100 packets, not including the initial 20 packets used to establish communication between IRC client and server.

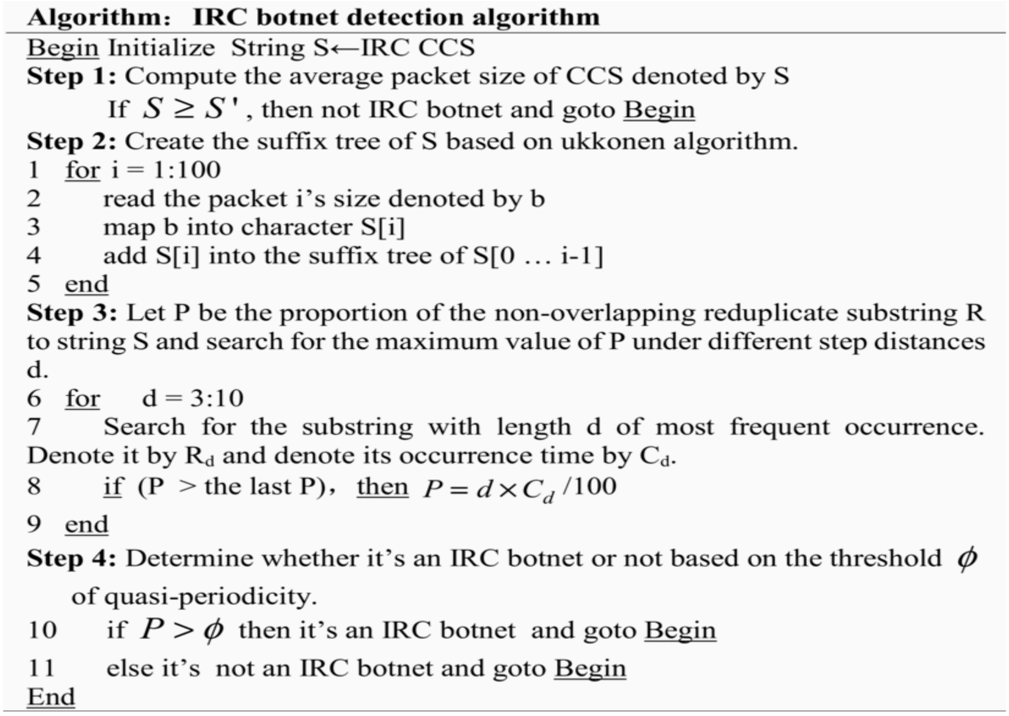
This algorithm's pseudo code can be seen in Figure 21. The variables that form its basis are as follows (Xiaobo et al., 2010):

- *S* = average packet size of CSS;
    - This is to be calculated repeatedly during the lifetime of each experiment by looking at the total size of captured flow sample, in bytes, and dividing that by the actual number of packets present within the sample.
- S' = threshold of average packet size;
    - This is to be set at the beginning of experiment based on readings

performed on legitimate network traffic outside of the experiment.

- P = quasi-periodicity degree of CCS;
- *Φ* = threshold of the quasi-periodicity degree.

Thus, the communication will end up being classified as IRC botnets only if the conditions $P > \Phi$ and $S < S'$ are simultaneously satisfied.



```
Algorithm: IRC botnet detection algorithm
Begin Initialize String S←IRC CCS
Step 1: Compute the average packet size of CCS denoted by S
        If $S \geq S'$, then not IRC botnet and goto Begin
Step 2: Create the suffix tree of S based on ukkonen algorithm.
for i = 1:100
    read the packet i's size denoted by b
    map b into character S[i]
    add S[i] into the suffix tree of S[0 … i-1]
end
Step 3: Let P be the proportion of the non-overlapping reduplicate substring R to string S and search for the maximum value of P under different step distances d.
for d = 3:10
    Search for the substring with length d of most frequent occurrence. Denote it by $R_d$ and denote its occurrence time by $C_d$.
    if (P > the last P), then $P = d \times C_d / 100$
end
Step 4: Determine whether it's an IRC botnet or not based on the threshold $\phi$ of quasi-periodicity.
    if $P > \phi$ then it's an IRC botnet and goto Begin
    else it's not an IRC botnet and goto Begin
End
```

Figure 21 - IRC Botnet Detection Algorithm (Xiaobo et al., 2010, Table 1)

6.2 AMDS Botnet Detection Module - Design

This section focuses on the design and implementation of the malicious activity detection module, based on the abstract model discussed in section 6.1.

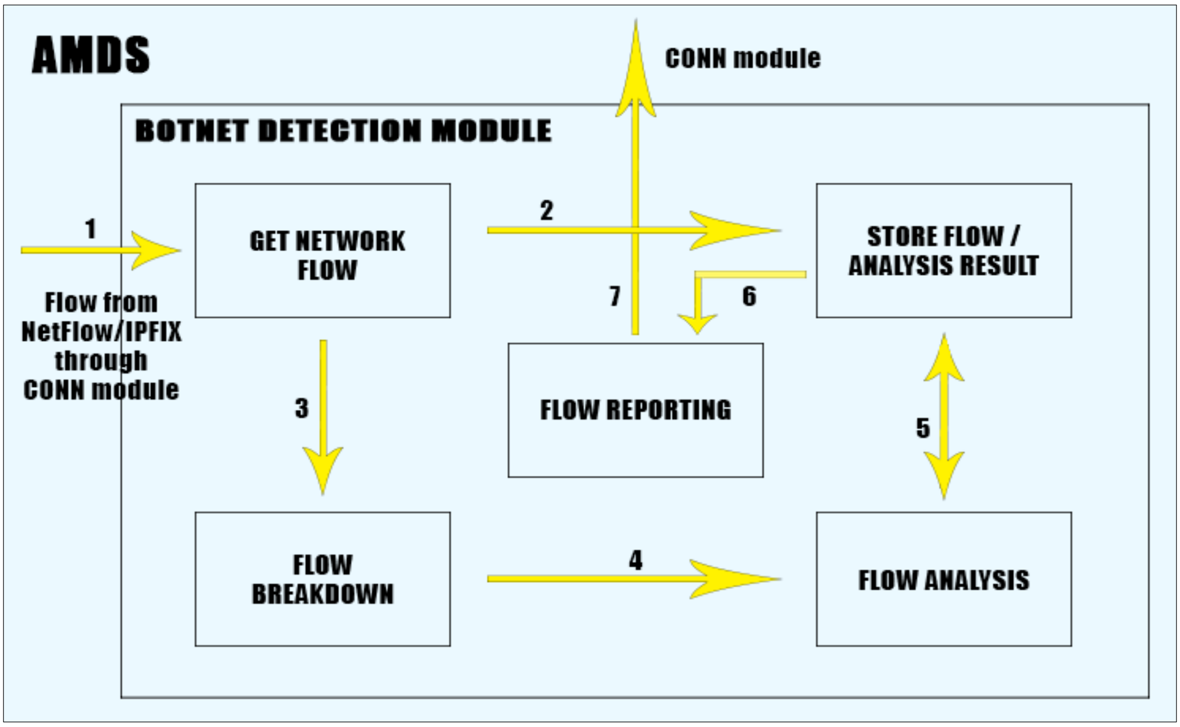


Figure 22 - Botnet Detection Module Design

As can be seen in *Figure 22*, the Botnet Detection software module design specifies the logical position of its five major components and the information flow between them, illustrated by the numbered yellow arrows. The data flow direction is given by the yellow arrows, while the sequence in which it takes place is indicated by the numbers 1 to 7, 1 being the data entry point and 7 being the data exit point, both of which being facilitated by the AMDS Connection module. For more information on the Connection module's design, please see Chapter 5.

This new module's major components are all designed to each fulfil a specific function, thus feeding into the AMDS modular design concept. They will now be described at an overview level, as follows:

I. *Get Network Flow*. This component, as its label suggests, is responsible for requesting and accepting network data flows from the NetFlow/IPFIX setup through the AMDS Connection module. It first initiates the request, which is then forwarded by the Conn module using an appropriate communication interface, and it then awaits a response in the form of a Network Flow. Once it receives this information, it passes it on to both the storage and breakdown components.

II. *Store Flow / Analysis Result*. This part of the Botnet module is responsible with storing, as a long-term solution, raw Network Flows as forwarded by the previously discussed component and analysis results as transmitted by the *Flow Analysis* component, described below. It uses an internal storage system rather than relying on the other AMDS modules for security considerations. Another responsibility it carries refers to passing on the analysis results to the *Flow Reporting* component.

III. *Flow Breakdown*. This element is charged with extracting key bits of information from the raw network flows, as required by the detection algorithm, and forwarding them to the *Flow Analysis* component. Generally, it looks for packet size, IP addresses and ports (for both packet source and destination), class of service, device interface, and protocol type.

IV. *Flow Analysis*. This block performs the most complex and key functional tasks of the module. It is the embodiment of the detection algorithm and, as such, is charged with utilising the information extracted from the raw network flows. The main purpose of this component is to attempt to detect malicious botnet activities by comparing current findings with past findings. This is achieved through querying the *Store Flow / Analysis Result* component for old records of previously analysed data and comparing healthy flows with current, yet unidentified flows. In the event of an inconclusive analysis result, it flags the current flow as such. Finally, it forwards its findings to the storage component.

V. *Flow Reporting*. This element's main responsibility is to retrieve analysis results and forward them to the User Interface AMDS module for review purposes. It also is charged with forwarding analysis statistics based on past results, which helps provide an overview of this module's activities.

In conclusion, this chapter has presented in detail the theory, design, and functionality of a Botnet detection module for the AMDS. Next, the hardware used by the author to create the test bed, as well as the tests the author ran and their results will be presented.

Chapter 7: Cloud Computing Test Bed – Software Deployment on Hardware

Chapter 7: Cloud Computing Test Bed – Software Deployment on Hardware

This chapter describes how the AMDS was installed on a Virtual Machine and what steps have been undertaken to achieve interconnectivity between the Virtual Network interfaces and the underlying physical hardware.

7.1 Cloud Test Bed Hardware

The author, as previously detailed in Dinita et al. (2012), has access to a small but powerful test bed comprising seven physical servers (mix of HP Proliant, Dell R710 and Viglen brands), three Storage Area Networks (HP) and multiple routers and switches (CISCO and HP), and HP ILOs (TABLE 1) located within the Department of Computing and Technology (FIGURE 23 and FIGURE 24 are illustrating the structure of these systems) across four rooms (Bry015, Mel205, Datacentre, Bry101).

Two of the servers are 'public'-facing (that is, they are utilised for teaching purposes), whilst the others are 'private'-facing (i.e. they are utilised for research – not yet implemented). The private-facing resource is isolated from the Internet and only accessible from departmental computers via an appropriate security protocol. All of the networking cards have Gigabit type interfaces (>=1Gbps, Gigabit per second, transfer rate) and the link between the Servers and the Filer (FIGURE 24) is one of 10Gbps, which facilitates rapid data movement to and from the Filer across the network (Dinita et al., 2012).

The described test bed allows for the possibility of testing unique and complex scenarios such as simulating a global infrastructure, where each room is considered to be a remote location in a different country/part of the world (Dinita et al., 2012).

The author also has access to several items of test hardware that allow simple power consumption readings to be taken through proprietary User Interfaces: HP Integrated Lights-Out (ILO), APC Power Distribution Unit (PDU) and APC Uninterrupted Power Supply (UPS) devices. These devices are directly connected to the physical servers and as such produce relevant power metrics.

All hardware presented is fully network enabled and as such has been provided with unique IP addresses to which connections can be made. The specialised hardware relevant to this point all support the SNMP protocol (communication protocol that allows two way data connections).

Table 1 - Test Bed Components (Dinita et al., 2012)

Make	**Specifications**	**Quantity**
HP Proliant	108GB RAM 2 x Intel Six-Core Xeon 2.4GHz 10 Gbit NICs	4
Dell R710	38GB RAM 2 x Intel Quad-Core Xeon 2.4GHz Gbit NICs	2
Viglen	16GB RAM 2x Intel Dual-Core Xeon 2.2GHz Gbit NICs	1
HP Filer	8TB HDD 10 Gbit NICs	1
Dell Storage Area Network	1.5TB	3
HP Integrated Lights Out	Gbit NICs	4
Cisco 4948 Switch	Gbit NICs	1
Cisco 3560 Switch	Gbit NICs	1
Cisco 5510 Adaptive Security Appliance (ASA)	Gbit NICs	1
NetLab Pod	N/A, proprietary appliance	4

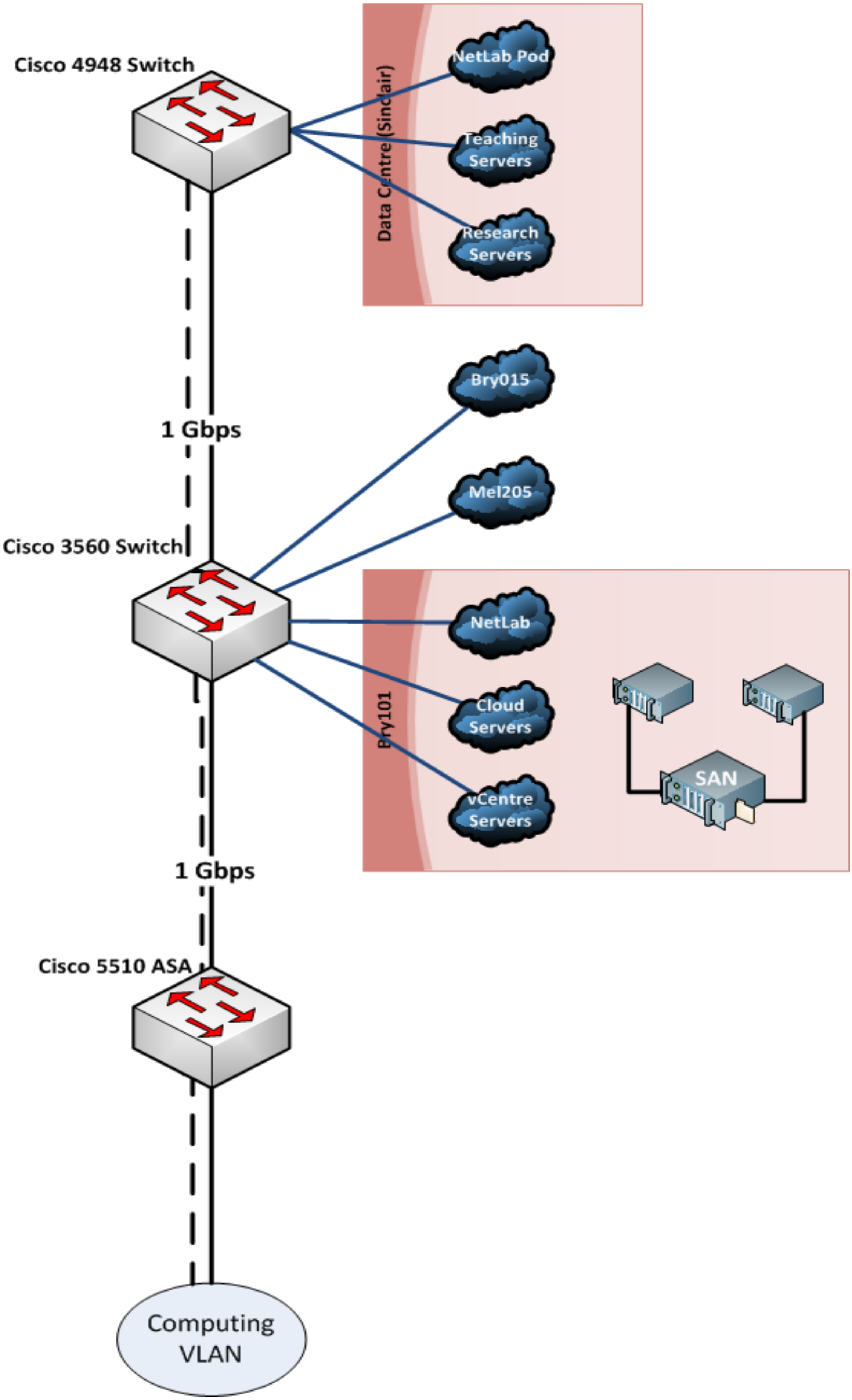


Figure 23 - Test Bed - Custom design infrastructure (Dinita et al., 2012)

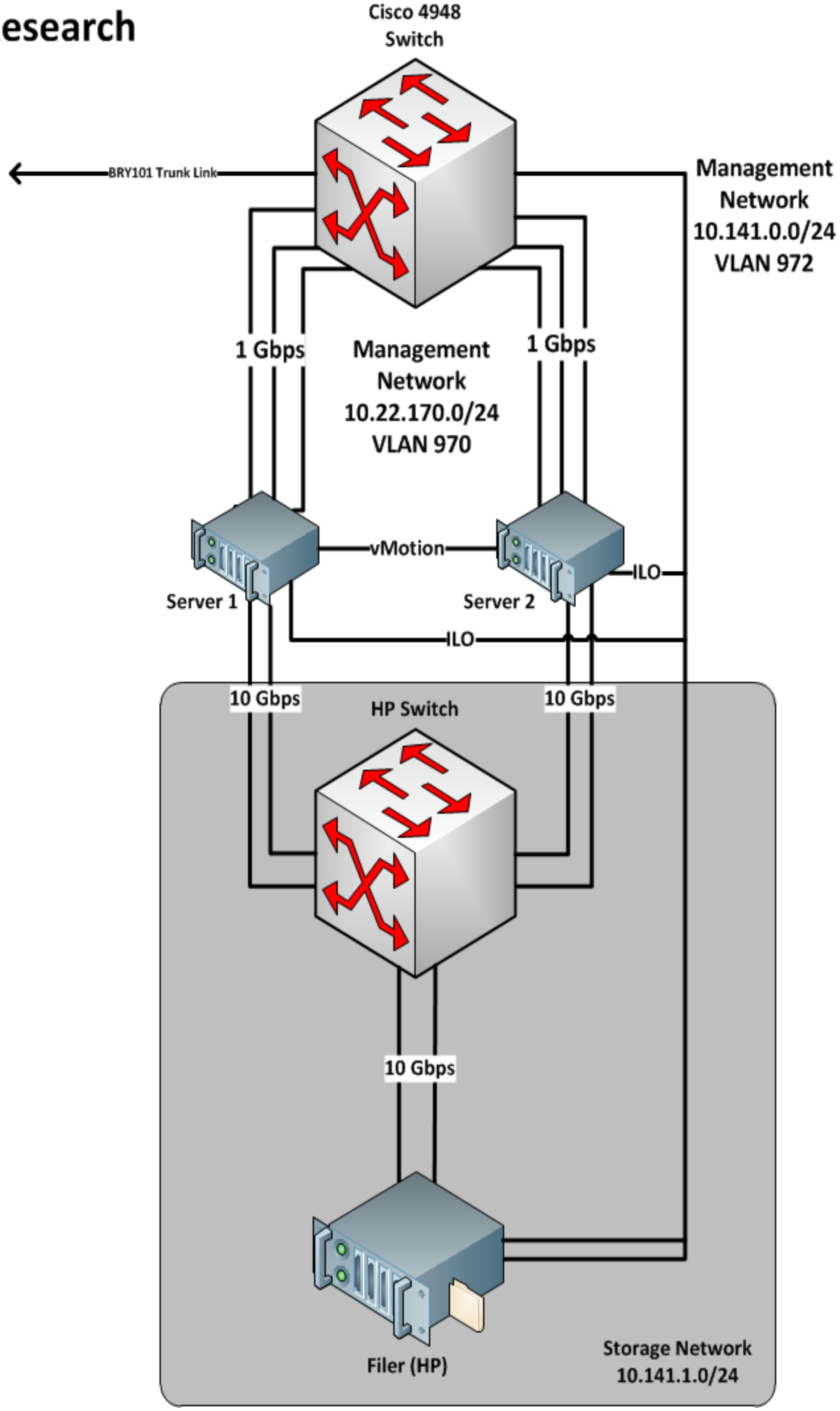
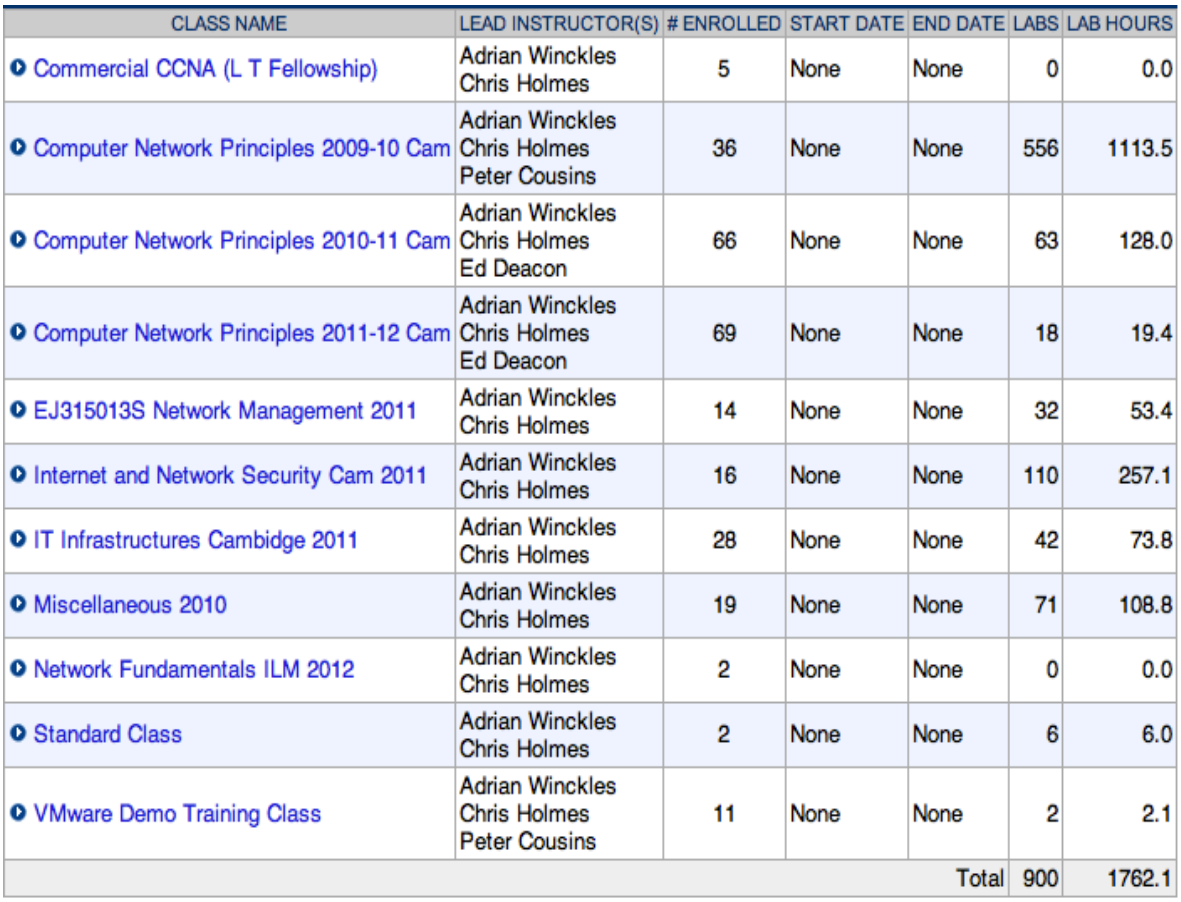


Figure 24 - Test Bed - Datacentre View (Dinita et al., 2012)



CLASS NAME	LEAD INSTRUCTOR(S)	# ENROLLED	START DATE	END DATE	LABS	LAB HOURS
Commercial CCNA (L T Fellowship)	Adrian Winckles Chris Holmes	5	None	None	0	0.0
Computer Network Principles 2009-10 Cam	Adrian Winckles Chris Holmes Peter Cousins	36	None	None	556	1113.5
Computer Network Principles 2010-11 Cam	Adrian Winckles Chris Holmes Ed Deacon	66	None	None	63	128.0
Computer Network Principles 2011-12 Cam	Adrian Winckles Chris Holmes Ed Deacon	69	None	None	18	19.4
EJ315013S Network Management 2011	Adrian Winckles Chris Holmes	14	None	None	32	53.4
Internet and Network Security Cam 2011	Adrian Winckles Chris Holmes	16	None	None	110	257.1
IT Infrastructures Cambidge 2011	Adrian Winckles Chris Holmes	28	None	None	42	73.8
Miscellaneous 2010	Adrian Winckles Chris Holmes	19	None	None	71	108.8
Network Fundamentals ILM 2012	Adrian Winckles Chris Holmes	2	None	None	0	0.0
Standard Class	Adrian Winckles Chris Holmes	2	None	None	6	6.0
VMware Demo Training Class	Adrian Winckles Chris Holmes Peter Cousins	11	None	None	2	2.1
				Total	900	1762.1

Figure 25 - NetLab+ Testing Results (Dinita et al., 2012)

As can be seen in Figure 25 (Dinita et al., 2012), there already have been 900 Labs and over 1700 hours of lab work put into testing the described test bed by over 260 students, all of which have run without any issues.

7.2 Deployment of the AMDS on the Cloud Test Bed

The AMDS is currently deployed and running on the author's test bed. An Ubuntu Linux based Virtual Machine was chosen to run the application due to the high reliability of the Operating System. The VM has is connected to the ESXi Servers via a closed Virtual Network. This ensures seamless connectivity between virtual and physical hardware, thus allowing the AMDS to receive information from the ILOs and the vSphere Client and to send commands back.

This section provides an in-depth description of three different experiments executed on the author's test bed.

7.2.1 Experiment 1: Ensure Correct AMDS Operation

This first experiment involves the AMDS running on the Cloud Test Bed as described in sections 5.3.3 and 5.3.4. Its scope is limited to verifying correct system operation within the cloud network.

This experiment has several key points it is trying to achieve:

I. Debug every AMDS module in an attempt to identify software flaws introduced at the point of creation.
II. Verify ASA-VLAN-AMDS network flow.
III. Verify AMDS-vSphere connectivity and command compatibility.
IV. Verify AMDS-ILO connectivity and command compatibility.
V. Test AMDS' analysis capabilities.

Details of the experiment are as follows.

The AMDS is deployed on a Virtual Machine Instance (VMI), which has been created based on the already defined VM Template through the VMWare vSphere Client. This VMI is connected to the internal Cloud Network as shown in Figure 26.

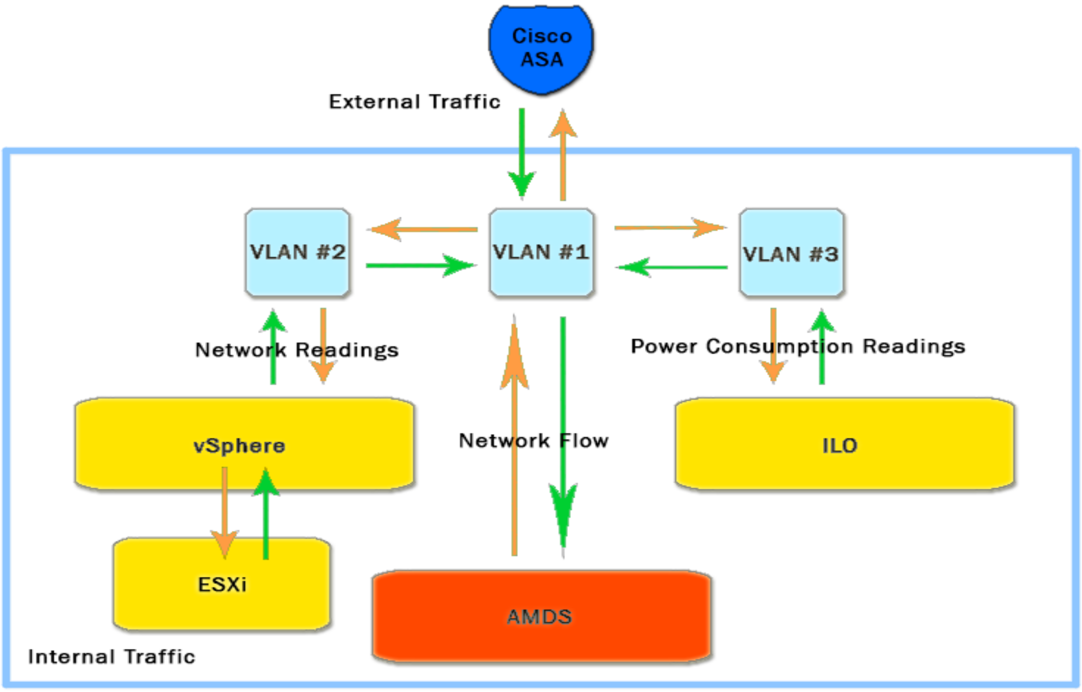


Figure 26 - AMDS Experiment Logical Layout - Ensure Correct AMDS Operation

The experimental parameters are the default ones set out in the configuration files. The impact and evolution of these parameters are out of the scope of this experiment and are presented in section 7.2.2.

At a logical overview level, the Cloud Infrastructure is composed of a Cisco ASA Router (Adaptive Security Appliance), three VLANs (Virtual Local Area Network), the vSphere Client, the ESXi Server, a group of ILOs (Integrated Lights Out), and the AMDS. Each of these components is underpinned by a series of physical network cables, switches, and routers that facilitate interconnectivity between them. The ESXi Server is comprised of multiple independent VMs interconnected by the three VLANs.

The network operational flow is expressed through two different arrow colours in Figure 26, both relevant to the AMDS:

- *GREEN* reflects network traffic flowing towards the AMDS, while
- *ORANGE* reflects network traffic flowing from the AMDS towards all other infrastructure components.

Green traffic is composed of data the AMDS requires when performing the infrastructure logical analysis from the points of view of processor loads, power consumption, and network flow, from the following sources:

- Processor load data is retrieved from the vSphere Client;
- Power consumption data is retrieved from the ILO group; and
- Network flow data is obtained from the various switches and routers spread across the infrastructure.

Orange traffic is composed of commands the AMDS issues post-analysis. This includes:

- VM moving commands to the ESXi Servers through vSphere;
- Physical server shut-down / start-up commands to various ESXi Servers through vSphere;
- Network flow restriction requests to the Cisco ASA router; and
- Load balancing requests across the infrastructure to other AMDS running instances.

Even though all infrastructure components are located within the same physical network, the VLANs allow splitting it up into smaller logical groups which can be more easily maintained and controlled. Each VLAN is created and maintained by the vSphere Client and only has direct access to the logical component in its immediate vicinity. This allows for the formation of highly compartmented and self-contained/-managed logical groups.

Since the AMDS has direct access to all VLANs, it is capable of interfering with regular network data flow based on its post-analysis results. This gives it the power to control every aspect of a physical infrastructure through controlling its logical groups. Once AMDS collects several hours worth of data, it is then capable of issuing commands to vSphere, which in turn will forward these, as needed, to other Components under its control.

Example commands that can be issued are among the following:

- If any servers are available, meaning underutilised (less than 75% processor load), attempt to split active VMs from one server among all the other ones in a similar position, and shut it down once this operation has completed.
- If all active servers are reaching peak efficiency (close to 100% processor load), start up a new server and add it to the active cluster.

In order to technically achieve all of this experiment's points, the author has put together a special Linux Virtual Machine template capable of performing server load stress tests. This is achieved by having the VM push the Virtual Processor to loads of up to 100%.

The VM comprises of the latest Ubuntu Linux operating system as well as the latest versions of the Apache Web Server and the Tsung open source multi-protocol distributed load-testing tool.

Apache is a web service that upon start it listens on a predefined port (usually 80) on the server. It is capable of delivering multiple web pages to millions of clients simultaneously. A simple web page has been created and put in place for the purpose of these tests.

Tsung[27] is a complex application that is capable of creating millions of simultaneous connections (also known as virtual clients) to any given web service. The

[27] http://tsung.erlang-projects.org/

configuration file allows fine control over the length of each connection time wise as well as how fast the number of simultaneous connections grows over a predefined time frame. This allows measurement of server power consumption while the processor load ranges from 0% to 100%. Other features of Tsung are beyond the scope of this research.

Upon launch the VM starts up both Apache and Tsung. Apache is set up to listen on port 80 for connections from anywhere on the network (in this case, from within the same location – localhost or 127.0.0.1). Tsung is set up to create virtual clients for Apache every 0.5 milliseconds over a time frame of 15 minutes. By the end of the given time frame the number of simultaneous connections will mount up to 1.8 million. This forces the Virtual Processor to slowly go through the needed loads.
For the purpose of these tests the authors have deployed 40 VMs based on the original template VM and have launched them all at the same time to achieve the desired effect.

The second AMDS experiment, designed to test AMDS network sampling capabilities, will now be presented.

7.2.2 Experiment 2: AMDS Network Sampling

This second experiment involves the AMDS running on the Cloud Test Bed as described in sections 5.3.3 and 5.3.4, as well as building upon the parameters of Experiment 1. The scope of this experiment is limited to AMDS network sampling through the use of NetFlow and IPFix open source software.

This experiment has several key points it is trying to achieve:

I. Sample 10% of all network data flow using IPFix / NetFlow. See below for an explanation of why this percentage was chosen.

II. Sort collected samples into logical groups based on parameters such as data packet Size, Source, Destination, and Commands.

III. Test the AMDS' ability to query stored data packets and analyse their respective groups in an attempt to discover unusual network behaviour.

IV. Construct an operational model based on packet analysis results capable of detecting suspicious client behaviour.

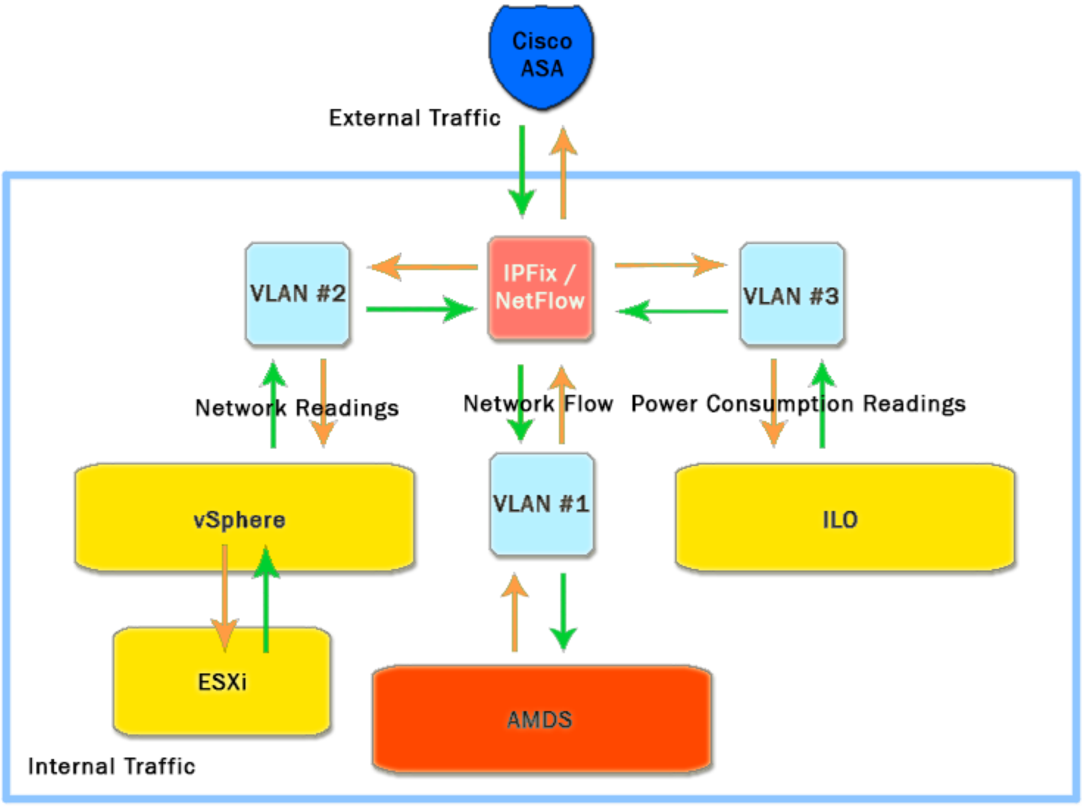


Figure 27 - AMDS Experiment Logical Layout - AMDS Network Sampling

In Figure 27 a new logical node has been created and inserted in-between existing logical network nodes (VLAN #1-3, and the ASA). This allows for potentially all data packets, the ones coming into the network as well as the ones going outside the network, to be stored in a local database and inspected at a very low level. Every packet has the option of being grouped up with other similar packets for easier comparison.

The IPFix / NetFlow logical node is capable, through outside (other locally networked devices) interference, to sample random data packets passing through them. Although it is capable of sampling 100% of the data, this is not recommended as it would slow down network flow as well as increase power consumption on the network hardware node it resides. For these reasons only 10% of all data is sampled, stored, and grouped up in the local IPFix/NetFlow database.

The local data packet database has the ability to be queried for small portions of data at a time for easier analysis. Also, the AMDS has the capability of achieving this task through one of its built for purpose modules. Once retrieved, the AMDS creates a graph of data packets by comparing their Size, Source, Destination, and Commands they carry. The more regular data it analyses, the higher the chance of it detecting unusual behaviour on the network, and the lower number of false positives.

After several data sets have been analysed, a model is then created which is used to test all future packet samples, while at the same time still keeping graph records used to continuously improve the data model.

7.2.3 Experiment 3: AMDS Performance Measurements

This third experiment involves the AMDS running on the Cloud Test Bed as described in sections 5.3.3 and 5.3.4, as well as building upon the parameters of the Experiment 1: Ensure Correct AMDS Operation chapter. The scope of this experiment is limited to AMDS performance measurements through self produced logs detailing each action and connection undertaken by the AMDS, together with the time in milliseconds each task has taken to complete.

The current setup also has the AMDS Botnet Module activated, which analyses NetFlow / IPFix network samples. This helps accurately determine the AMDS production performance on a live datacentre infrastructure.

This experiment has several key points it is trying to achieve:

I. Record execution time of each action and connection undertaken by the AMDS with the Botnet Module enabled.

II. Group all tasks into logical groups and calculate an average execution time for each task type.

In Figure 28 a new logical node (Botnet Module) has been created and attached to an existing logical node (AMDS). This allows for an accurate performance assessment in a fully operational datacentre environment.

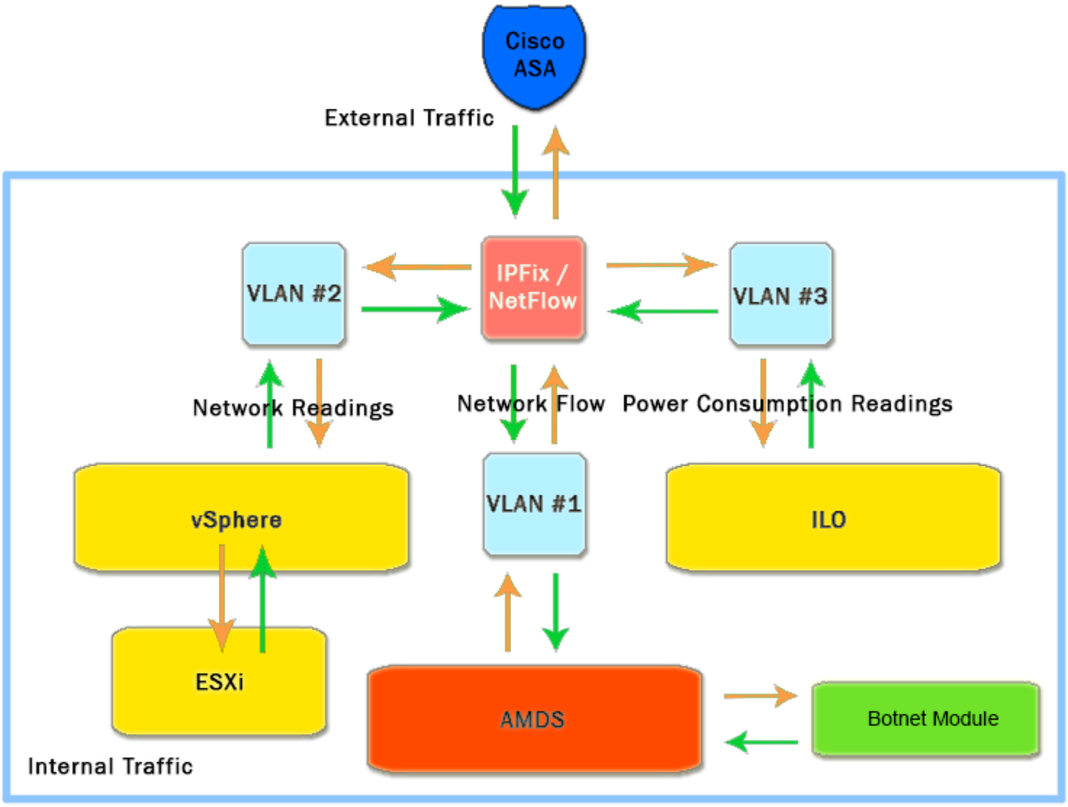


Figure 28 - AMDS Experiment Logical Layout - AMDS Performance Measurements

By default, the AMDS records all actions undertaken during its lifetime. These contain information on what triggered each action, the evaluated data, the result of the evaluation and the command issued to the action initiator, such as vSphere, an ILO, a switch or a router.

$$T_{total} = T_{end} - T_{start}$$

Equation 2 - Operation running time

In addition to this, for the purpose of this experiment the AMDS logging functionality has been extended to also record the start time of when an action has been triggered and the time of when said action has completed, in milliseconds. Then, a simple subtraction operation is applied to both recorded times according to Equation 2 and the final operation running time is attached to the current operation log entry.

After several log data sets have been analysed, a performance report is generated that provides an overall view on the AMDS operational timings.

7.2.4 Experiment 4: AMDS Botnet Module Detection Capabilities

This fourth experiment involves the AMDS running on the Cloud Test Bed as described in sections 5.3.3 and 5.3.4, as well as building upon the parameters of Experiments 1 and 2. The scope of this experiment is limited to further testing the AMDS Botnet Module's detection capabilities through a bigger, more standardised set of infected network packets.

In addition to the existing setup, the author has also added an external Botnet-like attacker using the botnet code written by Charles Leifer[28]. This is a simple program that allows a master machine to control other, infected machines through IRC commands. This experiment assumes said machines are already infected.

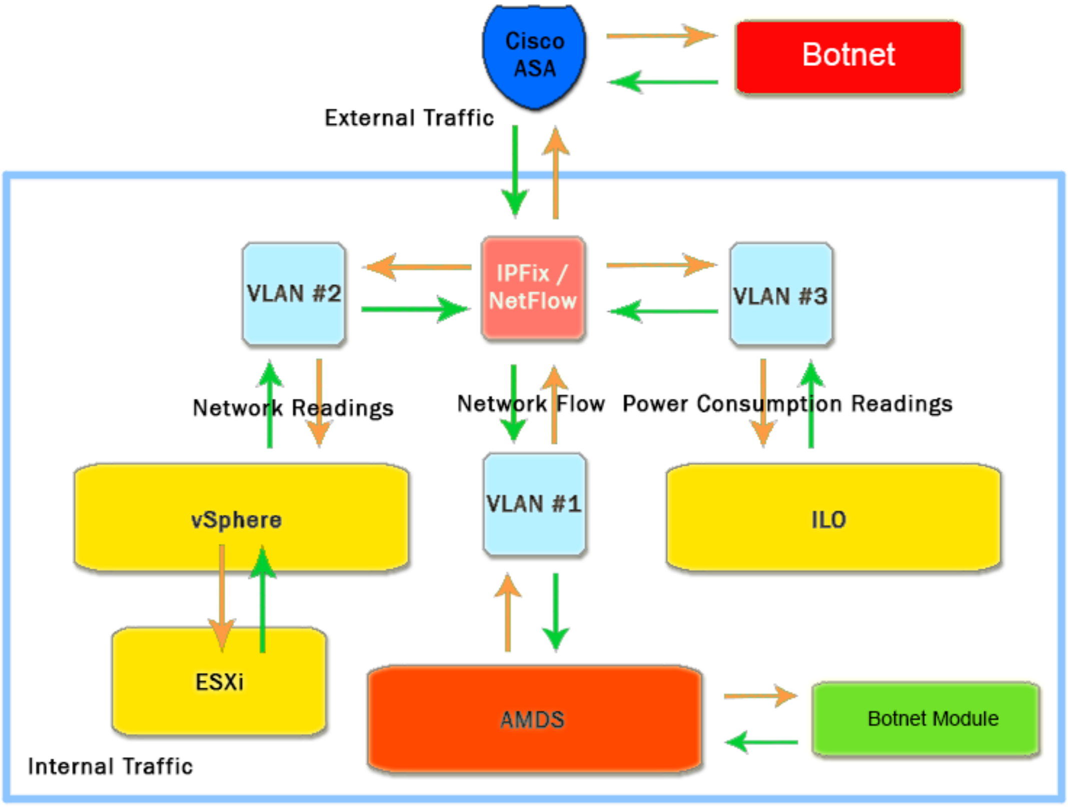


Figure 29 - AMDS Experiment Logical Layout - AMDS Botnet Module Detection Capabilities

[28] https://github.com/coleifer/irc/tree/master/botnet

This experiment has several key points it is trying to achieve:

I. Further test AMDS' ability to query stored data packets and analyse their respective groups in an attempt to discover unusual network behaviour.

II. Refine and test existing operational model based on packet analysis results capable of detecting suspicious client behaviour.

In Figure 29 a new logical node has been created and made available to the existing infrastructure by interfacing with the ASA and becoming part or regular network traffic. This new logical node is a very simple Botnet application, consisting of a master and several workers. This allows for infected data packets to be randomly introduced into the system alongside regular, healthy data packets.

The botnet master resides outside the system, while the botnet workers all reside somewhere inside running Virtual Machines inside the datacentre. The biggest implication of this is that all infected traffic will need to pass through the ASA, as well as the NetFlow/IPFix sampling node. All traffic going between these two parties is, for the purpose of this experiment, considered infected.

The packets used in this experiment vary in size, but are typically between 100 and 500 bytes. Regular user traffic is typically around the 500 byte mark, which should give the existing model ample leeway to adapt.

The previous detection rate found in Experiment 2 stands at 43% while using simple infected packets automatically generated through a simple program written by the author. The existing user activity model created based on said packets is now tested against near-real world botnet packets in an attempt to test and refine it.

7.3 Deployment Results Analysis

The AMDS has been designed to keep track of all incoming/outgoing data. As such, it keeps detailed logs of every activity that takes place within its boundaries. This section focuses on presenting the logs the AMDS has produced over a period of several months and on the analyses of the results.

7.3.1 Ensure Correct AMDS Operation Experiment Results

This section presents and discusses the results obtained from running Experiment 1, discussed in section Experiment 1: Ensure Correct AMDS Operation.

Since the AMDS has been deployed it has produced a great number of data stored within several databases. The data come from queries performed by the AMDS on the different networking hardware operating within the cloud environment (Switches, Routers, ILOs, ESXi).

The author has merged and analysed all generated data, the results of which have been expressed in Table 2 and Figure 30.

$$P_{rise} = P_{current} / P_{idle}$$

Equation 3 - Power Consumption Rise Percentage

$$E = L / P_{rise}$$

Equation 4 - Server Operation Efficiency

The system efficiency was calculated by using the formulas from Equation 3 and Equation 4. In Equation 3 P_{rise} is the power consumption rise percentage calculated by dividing $P_{current}$ (server power consumption at any other time – processor load > 0%) by P_{idle} (server power consumption when idle – 0% processor load). In Equation 4 E (server operating efficiency) is calculated by dividing L (server load) by P_{rise}.

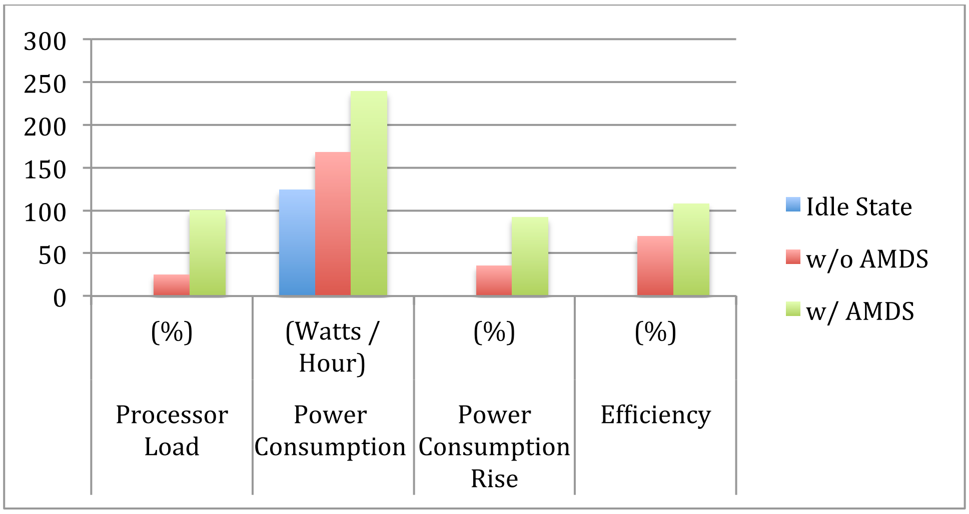


Figure 30 - AMDS Running Results – Graph View

Table 2 - AMDS Running Results – Table View

	Processor Load (%)	Power Consumption (Watts / Hour)	Power Consumption Rise (%)	Efficiency (%)
Idle state	0	124	N/A	N/A
w/o AMDS	25	168	35.5	70
w/ AMDS	100	239	92	108

In the second row of Table 2 labelled "Baseline" the data have been recorded on a machine in an idle state i.e. no other processes or applications running on it apart from the base system services. The Processor Load is at 0%, with a Power Consumption of 124 Watts / Hour. In this idle state, there are no Power Consumption Rise or Efficiency calculations to be done.

In the third row of Table 2 labelled "w/o AMDS" the data have been recorded before the AMDS has been enabled (decision making module was disabled). The system Efficiency stabilized at 70% due to the fact that several servers operating at 25% of their potential hardware load. The server power consumption at this stage was 168 Watts / Hour, an increase of 35.5% from system idle state.

In the fourth row of Table 2 labelled "w/ AMDS" the data have been recorded after the AMDS has been enabled (decision making module was enabled). Almost immediately all system traffic had been redirected towards one of the active servers while the others had been shut down to conserve power, thus bringing the system efficiency up to 108%. Power consumption at this stage was 239 Watts / Hour, an increase of 92% from system idle state.

The readings gathered from both the relevant hardware devices have been put side by side in Table 3 and Table 4, with the corresponding graphical representation shown in *Figure 31*, Figure 32, and Figure 33. Estimates of power consumption over longer periods of time have been generated in Table 5 with associated data graphed in Figure 34.

Table 3 - Processor Loads VS Watt Server Power Consumption

	Watts / Hour	Watts Power Consumption Rise (WPCR) (%)	Efficiency (PL / WPCR) (%)
0% Processor Load (PL) (Idle)	124	N/A	N/A
25% Processor Load (PL)	168	35.5	70
50% Processor Load (PL)	191	54	92
75% Processor Load (PL)	217	75	100
100% Processor Load (PL)	239	92	108

Table 4 - Amps Server Power Readings vs Processor Loads

	Amps	Amps Power Consumption Rise (APCR) (%)	Efficiency (PL / APCR) (%)
0% Processor Load (PL) (Idle)	1.2	N/A	N/A
25% Processor Load (PL)	1.55	29.1	86
50% Processor Load (PL)	1.8	50	100
75% Processor Load (PL)	1.95	62.5	120
100% Processor Load (PL)	2.2	83	120.4

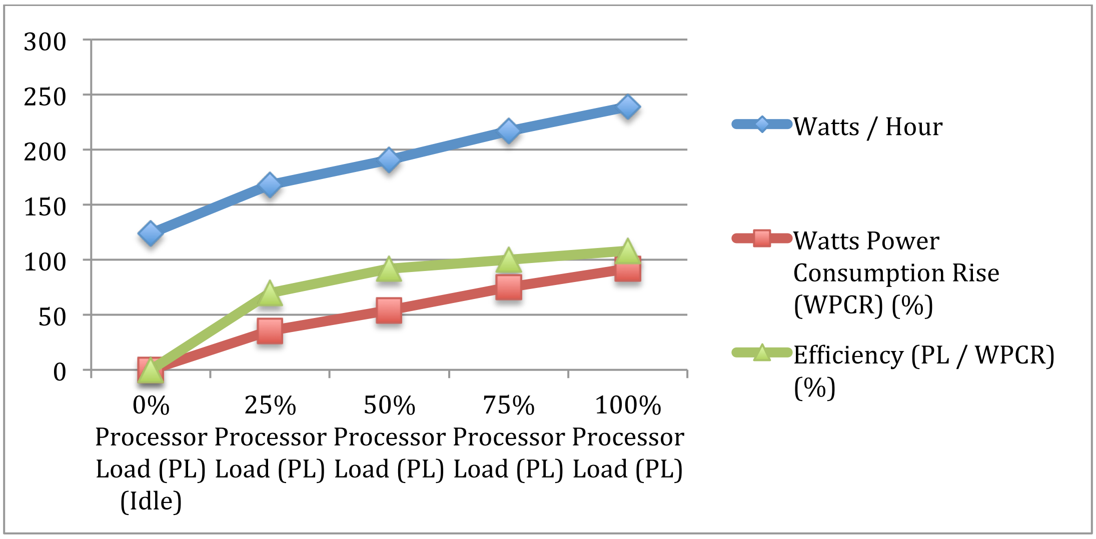


Figure 31 - Processor Loads VS Power Consumption - Graph View

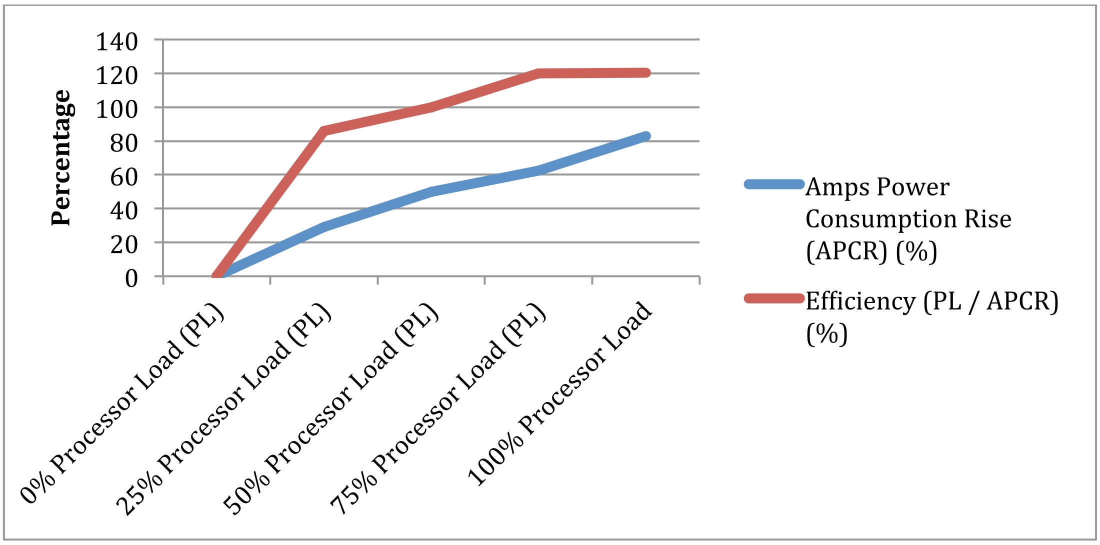


Figure 32 - Amps Server Power Readings vs. Processor Loads (PL) – Rise and Efficiency Percentages

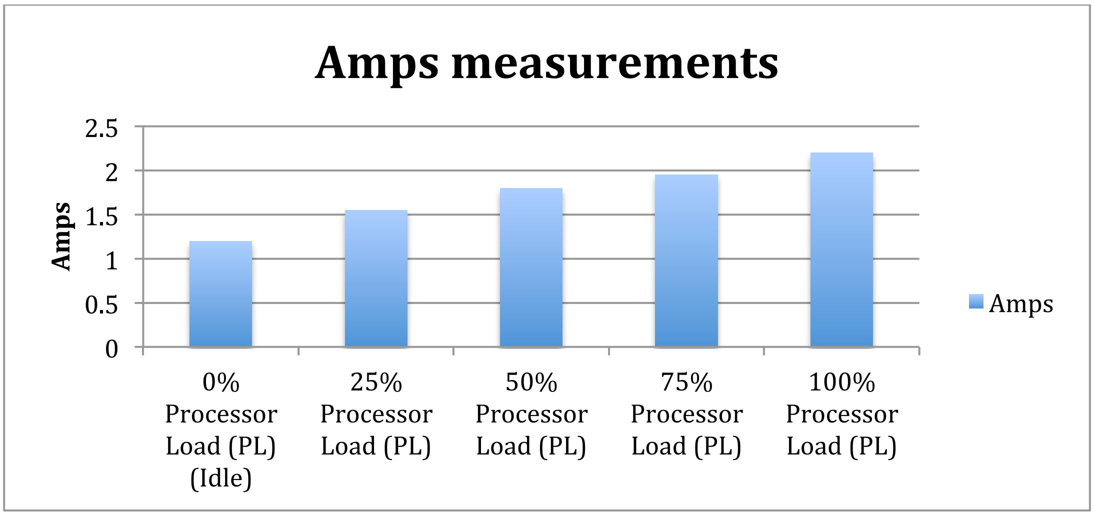


Figure 33 - Amps Server Power Readings vs. Processor Loads (PL) - Amps measurements

The results show that the higher the Processor Load is, the more efficient the Watt Consumption becomes. At 100% Processor Load there is only a 92% increase in Watt Consumption, resulting in an 8% power consumption reduction. The same applies for Processor Load percentages and Amps Consumption percentages. For 100% Processor Load there is only an 83% increase in Amps Consumption. For optimal efficiency it appears to be desirable that the Processor Load be kept over 75% for the best Power Consumption efficiency.

Table 5 – Over time Power Consumption estimates with variable Uptime

	1 Month (±732h) (Megawatts)	**3 Months (±2196h) (Megawatts)**	**6 Months (±4392h) (Megawatts)**	**12 Months (±8784h) (Megawatts)**
0% PL (124 Wh)	0.91	0.272	0.545	1.09
25% PL (168 Wh)	0.123	0.369	0.738	1.48
50% PL (191 Wh)	0.14	0.42	0.839	1.68
75% PL (217 Wh)	0.159	0.477	0.953	1.9
100% PL (239 Wh)	0.175	0.524	1.05	2.1

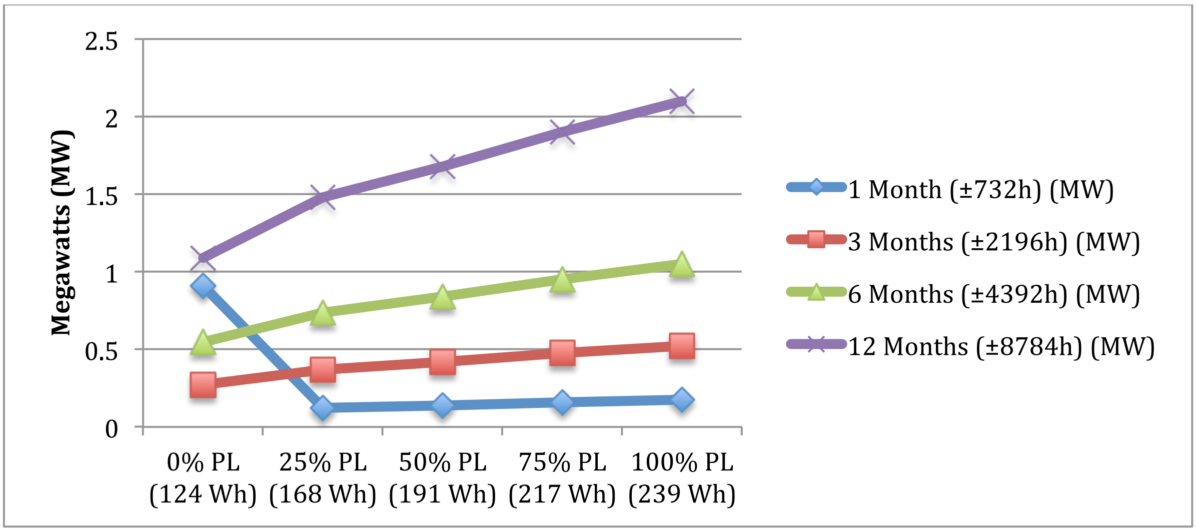


Figure 34 - Over time Megawatt (MW) Power Consumption estimates with variable Uptime

From the perspective of datacentres it is more cost effective to maximise the Processor Load on each server and running fewer servers rather than having a higher number of servers but with lower Processor Load percentages. Furthermore, using the results presented in Table 5 and Figure 34, the cost of operation from an energy efficiency point of view can be calculated for the following Datacentre use case scenarios:

- Case 1 (UC1): one server running at 100% PL (AMDS not running), operating cost estimation projected in Figure 35;

  vs.

- Case 2 (UC2): four servers running at 25% PL each (AMDS running), operating cost estimation projected in Figure 35.

Table 6 - Cost estimates for two Use Case scenarios

Use Case 1 (UC1) One Server @ 100% PL	2.1 Megawatts (See Table 5) * £88.45 / Megawatt = £186
Use Case 2 (UC2) Four Servers @ 25% PL	One Server: 1.48 Megawatts (See Table 5) * £88.45/ Megawatt = £131 Four Servers: £131 * 4 = £524
| UC1 – UC2 |	£338 UC2 shows a 181.72% cost increase compared to UC1

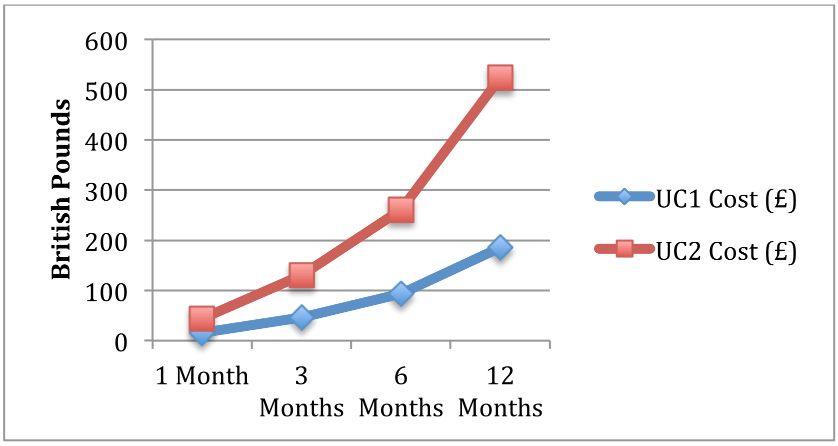


Figure 35 - Use Case 1 and 2 monthly operating costs over one year

Both use cases assume an operation time span of one year and the price used in the calculations is £88.45 / Megawatt [29]. As can be seen in Table 6, operating one server at 100% capacity for one year costs approximately £186 (UC1), while operating four servers at 25% capacity for the same period of time costs approximately £524 (UC2), £131 per server, resulting in a significant 181.72% increase over UC1.

[29] Price taken from http://www.businesselectricityprices.com/kwh.php on 27th November 2012. Multiplied by 10 to get to £ / MWh (Megawatt per hour).

The results presented above demonstrate how the AMDS is capable of minimising the cloud system power consumption by up to 8%, since at 100% Processor Load there is only a 92% increase in Watt Consumption, and at the same time generating an important operating cost reduction. They also show highly potential industrial applications in datacentre energy management and the lowering of datacentre operating costs, as follows:

i. *Green Datacentre.* The proposed system is capable of reducing overall energy consumption by intelligently turning physical servers on and off based on data collection from throughout the computing cluster.

ii. *Lower Datacentre operating costs.* This is a direct consequence of the previous statement. Overall lower energy consumption leads to reduced operating costs. This in turn allows for higher profits and more investments to be made.

iii. *Set-and-forget scenarios.* The AMDS, due to its autonomous nature and modular design, is capable of on-the-fly self-reconfiguration based on analysis results of gathered data. This flexibility makes it ideal for set-and-forget situations as well as low cost maintenance schedules.

7.3.2 AMDS Network Sampling Experiment Results

This section presents and discusses the results obtained from running Experiment 2, discussed in section 7.2.2.

The experiment has run over an extensive period of time and it has produced a great deal of log data. An overview of this data, along with some of the experiment parameters, can be seen in Table 7.

The experiment has been run using the following parameters:

A. 1Gbps network connection speed between the clients and the AMDS, which allows a maximum of 1 billion bytes per second.

B. Data packet sample size was set at 10% of all traffic at the point of collection. The reason for choosing this percentage is twofold:

   a. Although the IPFix / NetFlow logical node is capable of sampling 100% of the data, this is not recommended as it would slow down network flow as well as increase power consumption on the node they reside.

b. The heuristic algorithm only looks at average packet sizes in order to detect malicious network activity and, as such, 10% is enough to perform this task.

C. Average data packet size ranged between 500 and 1000 bytes of data. An average packet size of 500 bytes (S' from the heuristic algorithm presented in Chapter 6.1) has been derived from measurements performed prior to running the experiment on legitimate network traffic, generated with the help of the Botnet software[30] by configuring it as such. The Botnet software employed to generate the malicious traffic automatically set the difference in infected packet size between 1 and 500 bytes, resulting in a total packet size of between 500 and 1000 bytes (legitimate + malicious commands).

D. Infected (Botnet) packets have been used randomly starting with Sample #500. This threshold was chosen in order to give the detection heuristic algorithm a chance to create a healthy packet model to compare infected packets against.

Table 7 - Packet Analysis Results

	Sample #50	**Sample #250**	**Sample #500**	**Sample #1000**
# of Packets (1000s)	200	200	133	100
~ Packet Size (Bytes)	500	500	750	1000
# Infected (1000s)	0	0	18	28
# Detected (1000s)	0	0	5	12
Detection Rate (%)	0	0	28	43

For the first half of the experiment, as can be seen in Table 7, regular packets with an average size of 500 bytes have been filtered through the AMDS Botnet Detection Module. This has been used as a training mechanism for the heuristic algorithm exemplified the Botnets chapter, so it would later on have a healthy packet model to compare infected packets against.

[30] https://github.com/coleifer/irc/tree/master/botnet

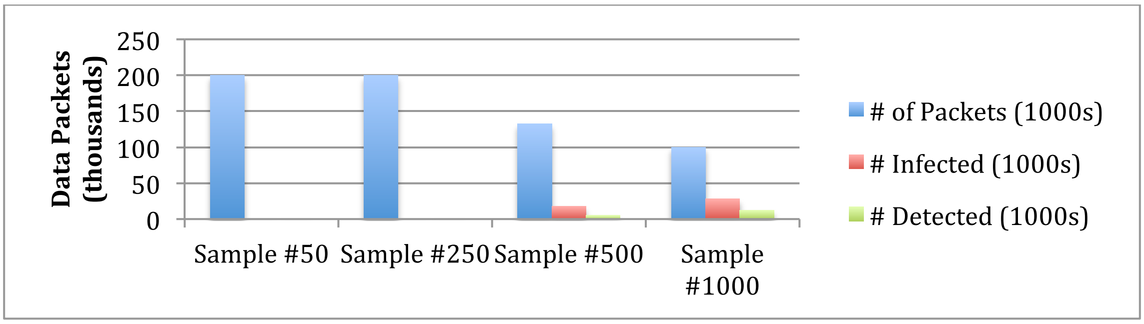


Figure 36 – Data Packet Distribution per 10% Sample

As can be seen in Figure 36, using a 1Gbps network link has translated into approximately 2 million data packets, 500 bytes each, as can be seen in Figure 37. Of those, 200 thousand packets (10% according to the experiment parameters) have been captured at each of the initial 500 sample readings and stored for analysis. From the point when Botnets packets were introduced into the Experiment (starting with sample #500), due to the Botnet packets increasing the overall size of each healthy packet (extra malicious commands accompany healthy packets), the total number of packets flowing each second through the network has been reduced. As such, these infected samples were made up of only between 100 thousand and 133 thousand packets on average.

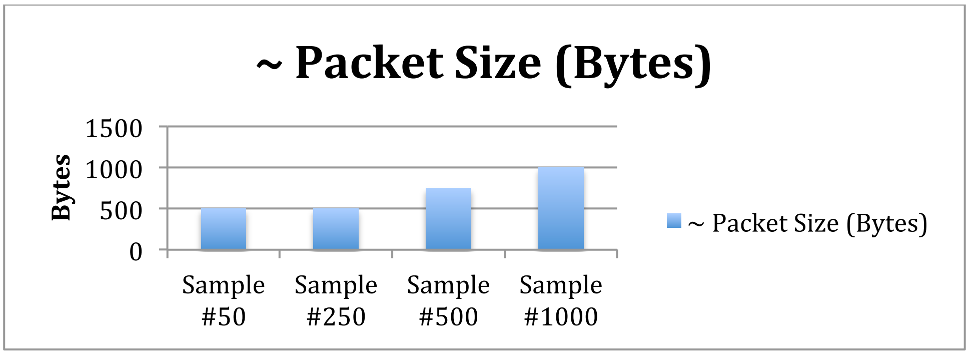


Figure 37 - Average Data Packet size in Bytes throughout Experiment lifetime

For the second half of the experiment, a random percentage of Botnet generated data packets have been introduced alongside the regular data packets used in the first half of the experiment. This had a direct impact on the data packet size as this has increased the average packet size of the samples by as much as 50%, from 500 to 750 bytes each, as can be seen in Figure 37. The Botnet packets have been created by combining botnet and client commands in one single packet, resulting in a data packet with an average size of 1000 bytes.

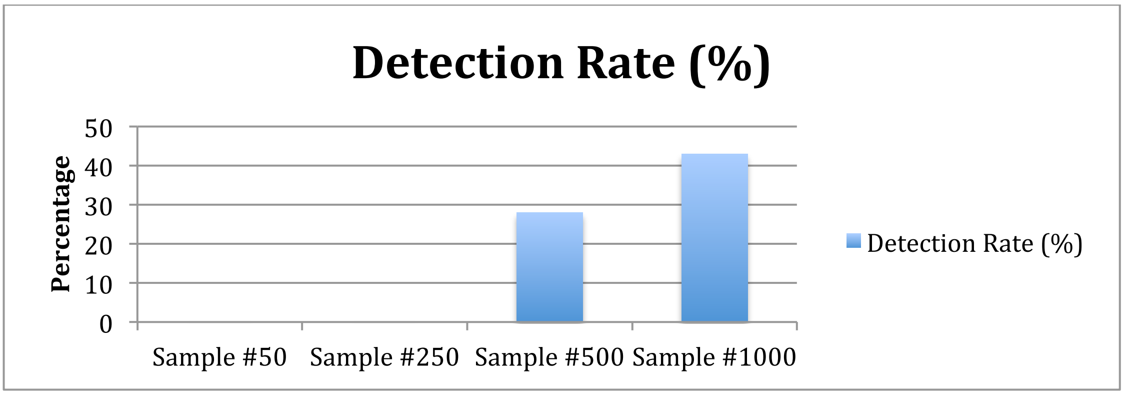


Figure 38 - Infected Data Packet Detection Rate

As can be seen in Figure 38, the AMDS, through the use of its Botnet Detection heuristic algorithm, has managed to detect approximately 28% of all infected packets at the start of the Botnet attack. This detection rate has steadily increased up until the end of the experiment to approximately 42% of all infected packets.

The implications of the results shown in Figure 38 go beyond just detecting a potential botnet attack in a datacentre. Although this fact alone gives these results meaning, it also allows for an abstract software model to be defined, discussed in the AMDS System Enhancement – Botnets chapter, which can then be implemented using different programming languages in a multitude of different computing environments.

The results presented above give a clear indication of the potential of the AMDS having its Botnet Detection Module activated. Applying the heuristic algorithm to more and more data packet samples allows the AMDS Botnet Detection module to better understand what Botnet data packets look like, and detect more similar packets or even unknown Botnet packet types in the future.

Furthermore, since a 1Gbps connection can only handle 1 billion bytes worth of data every second, whenever botnet communication is in progress it typically uses up part of that connection, which in turn slows down legitimate user's access to the datacentre's resources. Having the AMDS Botnet Module in place means that the ASA (Adaptive Security Appliance), responsible for potentially filtering out external traffic, can receive instructions to block all communication from the outside coming from the detected infected devices, thus allowing only trustworthy data packets to pass through, and as such improving the overall system accessibility for legitimate clients.

7.3.3 AMDS Performance Measurements Experiment Results

This section presents and discusses the results obtained from running Experiment 3, covered in section 7.2.3.

The data presented in Table 8 is an extract from the AMDS operational log produced throughout the AMDS lifetime. In order to remain within this experiment's boundaries, several irrelevant columns have been stripped out. The data represent several steps undertaken by two AMDS modules, the Botnet and the Control modules respectively, when processing both external and information.

The trigger column contains information on the source of the current request, the type of the operation, the operation start and end times, as well as the total execution time, also known as latency, expressed in milliseconds.

The AMDS Botnet module is responsible for determining whether incoming external network traffic is a botnet threat or not. In order to achieve this, it performs several steps in order to fully process a network flow. First, it retrieves the raw flow data. Next, it stores the data for historical and analytical purposes. Then, it proceeds with breaking it down into relevant chunks, such as IP source/destination, source/destination port/service, etc. Afterwards, the resulting data set is analysed by employing the algorithm described in section 6.1 of this thesis. Finally, the analysis results are stored in the local flow analysis database. An additional step would be taken based on whether a threat has been detected, not encountered during the lifetime of this experiment but evidenced by Experiment 4.

The other module present in Table 8 is the AMDS Control module. It is responsible for interfacing with the datacentre infrastructure from a power consumption efficiency point of view. To achieve this, it first retrieves raw power output from an ILO and stores it for historical and analytical purposes. Next, it also retrieves the CPU status of all servers under vSphere's command and stores this as well. Then, it performs an analysis on the recently stored data to determine if there is a need to send instructions to vSphere and stores the results. Finally, if required, it proceeds with contacting vSphere with relevant Virtual Machine commands, such as powering them down and shutting down the server they are hosted on, or powering up a server, depending on the case.

Table 8 - AMDS Operational Log Extract

Triggered by	Operation	Started at	Ended at	Total time (ms)
AMDS_botnet	networkFlow_retrieve	05/08/2014 13:06:11.433	05/08/2014 13:06:11.541	108
AMDS_botnet	networkFlow_storeRaw	05/08/2014 13:06:11.541	05/08/2014 13:06:11.585	44
AMDS_botnet	networkFlow_breakdown	05/08/2014 13:06:11.585	05/08/2014 13:06:12.011	26
AMDS_botnet	networkFlow_analyse	05/08/2014 13:06:12.011	05/08/2014 13:06:16.270	4259
AMDS_botnet	networkFlow_storeAnalysis	05/08/2014 13:06:16.270	05/08/2014 13:06:16.319	49
……………………	………………………………	……………………	……………………	………
AMDS_control	power_retrieve	05/08/2014 13:07:12.113	05/08/2014 13:07:12.145	32
AMDS_control	power_storeRaw	05/08/2014 13:07:12.145	05/08/2014 13:07:12.188	43
AMDS_control	cpu_retrieve	05/08/2014 13:07:12.188	05/08/2014 13:07:12.215	27
AMDS_control	cpu_storeRaw	05/08/2014 13:07:12.215	05/08/2014 13:07:12.262	47
AMDS_control	efficiency_analyse	05/08/2014 13:07:12.262	05/08/2014 13:07:14.183	1921
AMDS_control	efficiency_storeAnalysis	05/08/2014 13:07:14.183	05/08/2014 13:07:14.245	62
AMDS_control	vsphere_sendCommand	05/08/2014 13:07:14.245	05/08/2014 13:07:14.258	13
……………………	………………………………	……………………	……………………	………

Each operation latency is calculated by employing Equation 2 ($T_{total} = T_{end} - T_{start}$) on the recorded start and end times. Each operation represents a small part (method) of their parent module (Control, Botnet, etc.) and it is started internally through a chain of commands controlled by their parent. Depending on the flow at a given time, the operation may call upon another module's operation to perform some task for it. Each time this happens, the start and end times are recorded by each method, just before it begins and just after it finishes its intended task, and stores it in a log file. As can be seen in Table 8, the total execution time of one Botnet initiative, defined as the collection of operations needed to retrieve, assess and react to a network flow, is approximately 4,486 milliseconds. The flow analysis has taken the most time to complete due to the numerous comparisons it needs to make with the existing user activity model, as well as previous flows with ambiguous threat status.

Also from Table 8, the total execution time of one Control initiative, defined as the collection of operations needed to retrieve, assess and react to server CPU and Power readings, is approximately 2,145 milliseconds. The efficiency analysis has taken the longest to complete due to numerous comparisons it needs to make with the existing server activity model. The reason behind it taking less time to complete when compared with a Botnet initiative is the more complex nature of a network flow when compared with CPU and Power readings.

Table 9 - AMDS Average Module Operation Times (ms)

	Botnet Module (ms)	**Control Module (ms)**
Initial readings	4486	2145
End of Experiment Average Time	3855	2117
Change vs. Initial readings (%)	-14%	-1.3%
Period of time Observed (approx.)	05/08/2014 13:00 – 05/08/2014 16:00	

The results presented in Table 9 are average execution times of each of the more important AMDS modules: Botnet, Control, Auth, and Conn modules, respectively. These latency timings have been observed over approximately 3 hours of uninterrupted AMDS operational lifetime.

The *Initial Readings* table row represents the very first time each of the observed modules has completed one task. The *End of Experiment Average Times* table row have been calculated as averages of the time it has taken each module to start and complete subsequent tasks, until the end of the experiment.

The two main AMDS helper modules, Auth and Conn, both have very low average latencies (approximately 4ms, and 28ms respectively) due to the fact that they mainly deal with internal connections, both inside AMDS as well as the datacentre. Auth is slightly faster on average because all operations are handled in-house, while the Conn module is constantly dealing with external entities – vSphere, switches, routers, etc., and as such is dependant on network conditions as well as each device's programming features, current load or other factors. Due to their low latency times and the fact that they are constantly in use by all other AMDS modules, they were not included in the analysis.

As can be seen in Figure 39, the Botnet module average latency has declined by approximately 14% during the lifetime of this experiment, caused by the fact that with each new network flow the user activity model improves, and as such it is able to determine the flow threat level slightly faster each time. This is a significant latency reduction, which can only improve over time as the user activity model becomes better refined.

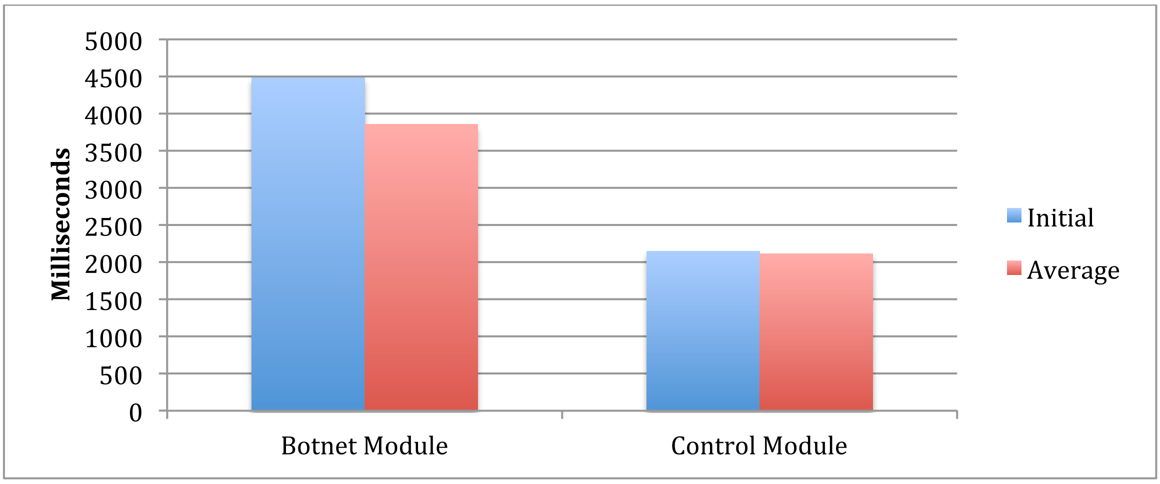


Figure 39 - Experiment 4 Results Latency Comparison

Also in Figure 39, the Control module average latency has also declined by approximately 1.3%. The most likely cause of this is the different latency expressed by the network devices it interfaces with, which are influenced mostly by external traffic fluctuations.

7.3.4 AMDS Botnet Module Detection Capabilities Experiment Results

This section presents and discusses the results obtained from running Experiment 4, covered by section 7.2.4.

The experiment has run over an extensive period of time and it has produced a large amount of log data. An overview of this data, along with some of the experiment parameters, can be seen in Table 10.

The experiment has been run using the following parameters:

A. 1Gbps network connection speed between the clients and the AMDS, which allows a maximum of 1 billion bytes per second.

B. Data packet sample size was set at 10% of all traffic at the point of collection. The reason for choosing this percentage is twofold:

    a. Although the IPFix / NetFlow logical node is capable of sampling 100% of the data, this is not recommended as it would slow down network flow as well as increase power consumption on the node they reside.

    a. The heuristic algorithm only looks at average packet sizes in order to detect malicious network activity and, as such, 10% is enough to perform this task.

C. Average data packet size ranged between 400 and 555 bytes of data. The Botnet application[31] used in this experiment was configured to generate healthy packets (calculated an average based on this - S' from the heuristic algorithm presented in Chapter 6.1) as well as infected packets throughout the lifetime of the experiment, in random amounts at random intervals, with the infected packets only appearing after sample #3000 was made.

D. Infected (Botnet) packets have been used randomly starting with Sample #3000.

[31] https://github.com/coleifer/irc/tree/master/botnet

Table 10 – Data Packet Analysis Results

	Sample #500	Sample #1500	Sample #3000	Sample #5000
# of Packets (1000s)	181	189	207	247
~ Packet Size (Bytes)	552	529	483	405
# Infected (1000s)	0	0	85	128
# Detected (1000s)	0	0	29	67
Detection Rate (%)	0	0	34.1	52.3

For the first half of the experiment, as can be seen in Table 10, regular packets with an average size of 529-552 bytes have been filtered through the AMDS Botnet Detection Module. This has been used as a training mechanism for the heuristic algorithm exemplified in the Botnets chapter, so it would later on have a healthy packet model to compare infected packets against.

As opposed to Experiment 2, the number of packet samples used has been increased in an attempt to give the heuristics algorithm leeway to adapt to real botnet conditions. Also, the average packet size of 483 bytes starting with Sample #3000, the sample when random botnet traffic has been introduced, is well below the equivalent Sample #500 from Experiment 2, where the average packet size was 750 bytes. The

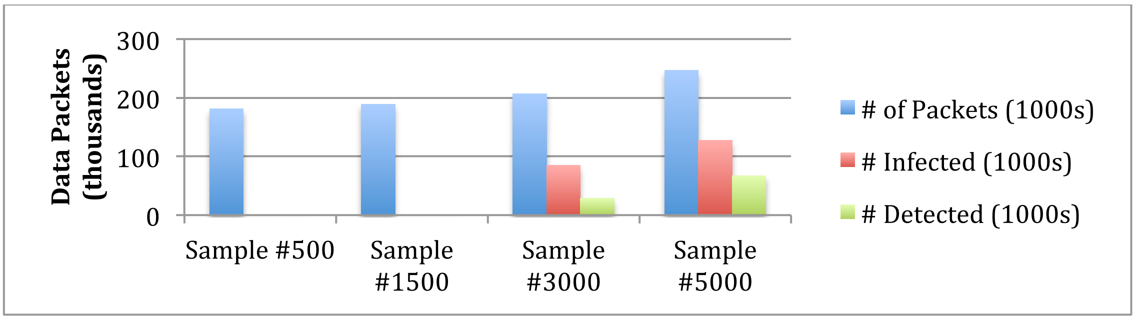


Figure 40 – Data Packet Distribution per 10% Sample

implication here is that the existing model was not tuned for detecting IRC Botnet communications, however, the detection rate has risen to just over 50%, which is better than in Experiment 2 where the detection rate was 43%.

As can be seen in Figure 40, using the 1Gbps network link has evaluated into approximately 1.81 million data packets, of which 181 thousand of the 500 initial readings for analysis have been sampled. The total number of data packets has slowly risen during the course of the experiment, at first due to random healthy traffic and towards the end due to random infected botnet packets. Having a reduced average packet size has impacted the samples containing infected Botnet packets in addition to the regular packets by having a reduced total number. These samples were made up of between 207 thousand and 247 thousand packets on average.

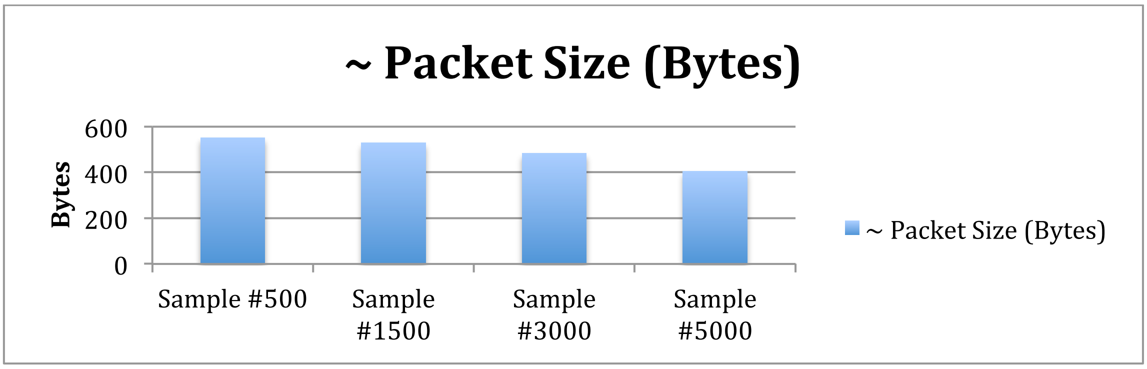


Figure 41 - Average Data Packet size in Bytes throughout Experiment lifetime

For the second half of the experiment, a random percentage of Botnet generated data packets have been introduced through the use of actual botnet code alongside the regular data packets used in the first half of the experiment. This had a direct impact on the data packet size as this has decreased the average packet size of the samples by as much as 24%, from 529 to 405 bytes each, as can be seen in Figure 41. The Botnet packets generated by the master and workers are a combination of general IRC commands as well as custom built attack commands, resulting in a data packet with an average size of 330 bytes. This value comes from measurements performed on the Botnet application outside the experiment for the purpose of calibrating the experiment parameters.

The heuristic algorithm attempts to detect malicious packets by comparing potentially infected packets to the healthy network traffic model it has built prior to the introduction of infected packets starting with packet #3000.

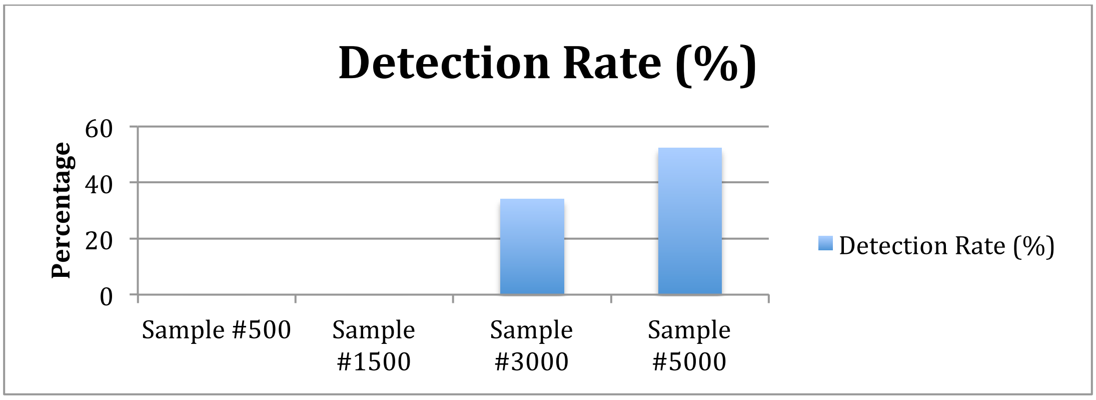


Figure 42 - Botnet Packet Detection Rate

As can be seen in Figure 42, the AMDS, through the use of its Botnet Detection heuristic algorithm, has managed to detect approximately 34.1% of all infected packets at the start of the Botnet attack. This detection rate has steadily increased up until the end of the experiment to approximately 52.3% of all infected packets.

Since this experiment has made use of a real world botnet system, the implications of the results shown in Figure 42 go beyond just detecting a potential botnet attack in a datacentre. Although this fact alone gives meaning to these results, it also allows for the abstract software model to be refined, as discussed in Chapter 6: AMDS System Enhancement – Botnets, which can then be implemented using different programming languages in a multitude of different computing environments.

The results presented above reinforce the results obtained in Experiment 3 (AMDS Performance Measurements Experiment Results) and again give a clear indication of the potential of the AMDS having its Botnet Detection Module activated. Applying the heuristic algorithm to more and more data packet samples allows the AMDS Botnet Detection module to better understand what real world Botnet data packets look like, and detect more similar packets or even unknown Botnet packet types in the future.

Furthermore, since a 1Gbps connection can only handle 1 billion bytes worth of data every second, whenever botnet communication is in progress it typically uses up part of that connection, which in turn slows down legitimate user's access to the datacentre's resources. Having the AMDS Botnet Module in place means that the ASA, responsible for potentially filtering out external traffic, can receive instructions to block all communications from the outside coming from the detected infected devices, thus allowing only trustworthy data packets to pass through, and as such improving the overall system accessibility for legitimate clients.

Chapter 8: AMDS in Comparison with Alternative Solutions

Chapter 8: AMDS – Operational Performance and Efficiency Evaluation

This chapter presents the summary of the test results from all AMDS experiments presented in chapters 7.3.1, 7.3.2, 7.3.3 and 7.3.4.

8.1 Experiment 1 Results Summary

In chapter 7.3.1, entitled Ensure Correct AMDS Operation Experiment Results, the main focus is on the main AMDS software package. This experiment has been designed to test the network and hardware loads sampling features built into the AMDS, as well as provide insight into how energy efficient a datacentre can become while employing its capabilities.

Initially, manual power consumption readings were taken by the author of the datacentre servers in order to establish a point of reference for the experimental results. The author found that while idling (0% utilisation), the servers were using 124 Watts / Hour. Once the datacentre had reached a stable operations status, servers were operating at approximately 25% capacity by default, thus only achieving 70% efficiency (168 Watts / Hour). From a power consumption point of view, this represents a fairly low efficiency within a datacentre.

Next, the AMDS was installed on a new Virtual Machine on one of the active servers and was left running for an extended period of time, while at the same time producing operation logs. Towards the end of the experiment, the logs were inspected and the findings proved the immediate positive impact of the AMDS on the datacentre: some of the servers that were active at the start of the experiment were shut down, while the Virtual Machines operating on them were moved to the other servers left running, thus increasing their load to close to 100% capacity. This allowed them to achieve 108% efficiency (239 Watts / Hour) when compared to an idle server, this also representing 38% increased efficiency when compared to a server operating at only 25% capacity, presented in the previous paragraph.

Furthermore, it also generated a significant operations cost reduction of 262%.

8.2 Experiment 2 Results Summary

In chapter 7.3.2, entitled AMDS Network Sampling Experiment Results, the main focus is on the data flow sampling and Botnet module features of the AMDS. This experiment has been designed to test the correctness of network sampling as well as measure the success rate of the Botnet module.

The experiment involved the NetFlow/IPFIX setup sampling network data approximately 1000 times, extracting 10% of the active network flow each time. The Botnet module has successfully passed the initial stage of retrieving, storing and breaking down the raw flow data.

The second stage, also named the training stage, involved creating a regular user activity model by observing normal network activity. To achieve this first activity model, 500 healthy samples were used with an average packet size of 500 bytes.
In the third and final stage, the model was used to compare another 500 samples of network data also containing malicious communications, generated by a simple custom script written by the author and containing approximately 750-1000 bytes of data per packet.

Initial results revealed a steady increase in the detection rate, rising from 28% after the first 250 infected samples to 43% after another 250 infected samples, thus validating the Botnet detection design proof of concept.

The implications of these results go beyond just detecting a potential botnet attack in a datacentre. Although this fact alone give these results meaning, it also allows for an abstract software model to be defined, as discussed in the AMDS System Enhancement – Botnets chapter, which can then be implemented using different programming languages in a multitude of different computing environments.

8.3 Experiment 3 Results Summary

In chapter 7.3.3, entitled AMDS Performance Measurements Experiment Results, the main focus is the performance efficiency, measured in latency times in milliseconds, of the AMDS software package. This experiment has been designed to record and provide an overview of the speed with which the AMDS performs under live datacentre conditions.

The findings of this experiment were positive, with the two main components that were tested, the Control and Botnet modules, operating well within expected parameters. The Control module achieved an initial operation latency average of 2,145 milliseconds, dropping to approximately 2,117 milliseconds by the end of the experiment, while the Botnet module achieved an initial operation latency average of 4,486 milliseconds, dropping to approximately 3,855 milliseconds. The results reflect the design and implementation approach taken by the author in developing the AMDS, which have allowed it to reduce its operational latency times by up to 14% during its lifetime through self-reconfiguration. Due to its asynchronous nature, this has virtually no impact on day-to-day datacentre operations, while at the same time having a positive impact on the energy management and security aspects of the infrastructure.

8.4 Experiment 4 Results Summary

In chapter 7.3.4, entitled AMDS Botnet Module Detection Capabilities Experiment Results, the main focus is the AMDS Botnet module. This experiment has been designed to further test the module's capabilities by making use of a simple, real-world Botnet software to generate malicious communication.

This experiment has been set up in a similar way to the experiment presented in chapter 7.3.2. However, this time, instead of a simple custom script, a real-world botnet software was used. To achieve the experimental parameters, a Botnet master has been set up outside of the datacentre and several Botnet slaves on Virtual Machines inside the network.

For the purpose of this experiment, the previous user activity model had been erased and the AMDS allowed to retrain itself with a new model. The reason behind this is the fact that the Botnet software used in this experiment generated smaller network data packets as opposed to the much larger custom generated packets from the other mentioned experiment. The initial healthy packets all had on average 552 bytes of data, while the infected ones had dropped the average to around 405 bytes per packet towards the end of the experiment. These numbers were derived from calculating the average size of healthy data packets while only legitimate traffic was used, while the average size of infected packets was calculated from packets containing both legitimate as well as malicious commands. These calculations were done for every network flow sample recorded throughout the lifetime of the experiment.

Once infected packets had started flowing through the network, after approximately 3000 flow samples, the initial results revealed an average detection rate of 34.1%; this is much higher than in the previous experiment. Towards the end of this test, after approximately 5000 flow samples, the detection rate had risen to 52.3%, which again is much higher than the previous results.

These final test results have once again confirmed the effectiveness of the Botnet design and validated the proof of concept software.

In conclusion, this chapter has reiterated the results of the four experiments performed by the author and discussed the impact of each one. All four sets of results confirm the importance of the work as well as validate the impact on datacentre day-to-day operations that AMDS would have.

Chapter 9: Conclusions and Further Work

Chapter 9: Conclusions and Further Work

This thesis shows how energy costs can be driven down, operational efficiency improved and security enhanced by deploying the AMDS on any Cloud environment. The proof of concept was developed on a VMWare backed test bed. This chapter reiterates the initial Research Objectives by relating them to the findings of this research and then concludes the work by identifying and formulating the significant original contributions to knowledge resulting from this work.

9.1 Reflection on aims and objectives

Looking at Objective 1 (*Critically evaluate pattern of disruption across a Cloud infrastructure as a result of an overloaded service request*), the overall accessibility for legitimate clients has been evaluated through the experiments carried out by the author.

Since typically datacentre's access points have limited traffic capacity (1Gbps = 1 billion bytes, 10Gbps = 10 billion bytes), there can only ever be approximately 2 million concurrent connections (assuming an average of 500 bytes per connection), be it either incoming or outgoing. As such, whenever illegitimate (disruption) traffic takes place, it uses up part of the overall available bandwidth, thus limiting access for legitimate clients. For a DDoS (Distributed Denial of Service) attack, the datacentre access point is typically flooded with many more connections than it can handle, thus preventing authorised access to the entire infrastructure.

As for Objectives 2 (*Conceptually develop a software optimization technique by which a Cloud could autonomously manage the workloads placed on that infrastructure*) and 3 (*Implement and test a software application to achieve Objective 2 for a specific Cloud scenario (VMWare Hypervisors)*), the AMDS software has been designed and developed from the ground up to be capable of interfacing with any existing datacentre infrastructure (VMWare in the author's case) for optimisation purposes.

The AMDS has been designed with modularity, security, and scalability in mind, allowing for any number of features to be added as needed, while at the same time keeping itself entirely secure by authenticating every single communication request in real time and dropping any unauthorised operations.

The built in functionality allows the AMDS to interface with network appliances (ILO, Switch, Router) via its SNMP module, VMWare vSphere appliance via its vSphere submodule within the Connection module, and other instances of itself for load balancing purposes, for times sustained high network traffic. Another important feature is its ability to reconfigure itself on the fly, thus making it highly adaptable to new situations. Also, its Control and Botnet modules enable the analysis of current network and server conditions, as well as incoming network traffic, which allows it to cover all parts of the datacentre that are susceptible to optimisation and security threats. The AMDS is written using Scala, a highly optimised and feature-filled programming language, running on the Java Virtual Machine (JVM). Since JVM is capable of being run on all existing operating systems (Windows, Linux, Unix, and their variations), this makes AMDS a highly portable and versatile software.

In terms of Objective 4 (*Innovatively develop metrics that quantify Cloud vs. centralized service provision in terms of environmental sustainability*), Experiment 1 presents two fundamental equations used to calculate an active server's efficiency, the latter being the quantifying metric. The author discovered that the higher a server's current processor load is, the more efficient it becomes to keep it active, so much that running near 100% load increases a server's power consumption efficiency by as much as 8% compared to a server operating at 75% load.

Objective 5 (*Conceptually develop an application that will identify virtualised system hijacking and undertake a range of appropriate activities from simple notification to service suspension*) has been tackled through the AMDS Botnet Module. This allows the AMDS to also assume networking monitoring responsibilities and help existing firewall filters detect malicious threats. The AMDS is capable of interfacing with flow capturing and sampling systems like NetFlow and IPFix to look at incoming network traffic and classify each data packet into either malicious or not malicious categories. It makes use of both a heuristic algorithm and a TCP weight equation to analyse said packets and assign a rating for each one. Higher ratings generally mean some kind of unusual activity taking place. Being an AMDS module, it has access to all of AMDS' features and capabilities, such as interfacing with any network or virtualisation appliance, which gives it real-time threat response abilities such as lock down the entire datacentre, or just move an infected Virtual Machine to a secured, sandboxed area inside the datacentre.

Looking at Objective 6 (*Test the method/software and compare against other alternatives, e.g. FPGA/hardware and other software systems*), the AMDS results have been discussed alongside existing datacentre optimisation techniques.
First of all, existing solutions are difficult to configure and extend with new functionality, due to either not being modularly designed or being embedded on hardware. That makes them rigid and makes them become rapidly out of date.
Second of all, none of the evaluated solutions are capable of reconfiguring themselves on the fly as the environment changes. This also makes them go rapidly out of date unless they receive new programming, thus making them expensive solutions.
Finally, none of the evaluated solutions have true real-time network and datacentre monitoring techniques. This makes them incomplete solutions and, as such, less relevant to entire datacentre infrastructures.

9.2 Original contributions to knowledge

The overall contribution to knowledge resulting from this research work resides in the formulation of a method to increase the efficiency of energy management and botnet detection and the conceptual design of a relevant prototype software package validated through experimental testing.

9.2.1 A novel method of optimising cloud networks in terms of energy consumption and system operation (AMDS)

The novelty consists of the software's ability to reconfigure itself on the fly based on live network readings. This ability is evidenced in chapter 7.3.1, where the datacentre power consumption has been monitored before and after the AMDS was enabled.

Initially, the datacentre stabilised at 168 Watts / Hour with several servers operating at 25% capacity, which resulted in 70% operational efficiency when compared with power usage of an idle server. Once the AMDS was activated, it had an immediate impact on the datacentre configuration through consolidating active Virtual Machines on only a few servers, allowing them to run at 100% capacity, while shutting the other ones down in order to conserve power. This translated into the remaining running servers operating at 108% efficiency, consuming 239 Watts / Hour, resulting in only a 92% increase in power consumption when compared to an idle server. As such, the AMDS had improved the datacentre energy efficiency through reducing power consumption by as much as 8% and, based on several use cases used to calculate server operating costs, helped reduce operating costs by as much as 262%.

A key outcome of these results is that it is more cost effective to maximise a few server loads than it would be to maintain several servers at less than peak loads.

These results also show highly potential industrial applications in datacentre set-and-forget scenarios, energy management and the lowering of operating costs.

9.2.2 A novel method to prevent, detect and stop network intrusions and malicious behaviour in a cloud infrastructure

This developed method applies, in particular, to detecting Botnet behaviour and stopping it from propagating throughout the datacentre, discussed in chapters 6.1.

The method employed uses an anomaly-based heuristic algorithm employing a Transmission Control Protocol (TCP) scan detection heuristic. The main algorithm requirement is that a scanner actively samples network flow and collects information on a percentage of observed data packets. This scanner would store, as a tuple set, the following data flow information: *IP source address*, *SYNS*, *SYNACKS*, *FINSSENT*, *FINSBACK*, *RESETS*, *PKTSSENT*, and *PKTSBACK*. The TCP part of the algorithm translates this flow information into a percentage. The closer this is to 100%, the probability of indication/detection of network anomalies increases.

This algorithm has taken the form of a Botnet Detection module bolted on the existing AMDS software, discussed in detail in chapter 6.2. At overview level, this module retrieves network information from a NetFlow/IPFix sampling setup, breaks it down into its basic components as required by the detection algorithm and compares it against a regular user activity model constructed through historic network flow data.

As can be seen in chapters 7.3.2 and 7.3.4, the malicious activity detection success rate has grown steadily over time up to 52.3% of all infected network communications. The main driver of this increase over time is the system's ability to reconfigure itself on the fly based on changes happening within and around the virtual environment it resides in. These initial results serve as a proof of concept of what can be achieved using a purely software approach to an existing datacentre hardware infrastructure.

Furthermore, the most significant advantage of this approach resides in this Botnet module's ability of directly interfacing with existing datacentre management utilities through the AMDS, thus having near real-time threat response capabilities. An example of a response is: upon the detection of malicious behaviour, the suspected Virtual Machines could be moved to a secure location within the datacentre until the threat has been neutralised, or even the entire datacentre is put on lock down until the situation has been resolved.

Finally, this novel approach also allows for an abstract software model to be defined, as discussed in Chapter 6: AMDS System Enhancement – Botnets, which can then be implemented using different programming languages in a multitude of different computing environments. This serves as an initial platform upon which more malicious behaviour detection algorithms could be developed in the future.

9.2.3 A flexible solution to a general communications/networking problem

The AMDS' software design approach has traditionally has been tackled through rigid hardware solutions. This design offers a completely modular software approach to enhancing existing datacentre systems for the purpose of acquiring and analysing network traffic, hardware loads and power consumption of a cloud infrastructure and redistributing them for efficient energy management and optimal data communication parameters (security, data transfer speed, access wait times, power consumption).

Although hardware based solutions tend to offer the best performance, they are often rigid and require manual intervention to help them tackle new problems. The AMDS, even as software running within a Virtual Machine, offers very low operational latency, as evidenced in chapter 7.3.3.

This experiment focuses on the two main AMDS components, the Control and Botnet modules. They have been designed to operate asynchronously in order to provide maximum efficiency, and as such have managed an average operational task latency of 4,486 milliseconds for the Botnet module and 2,145 milliseconds for the Control module. Compared with initial readings, as expected, both modules have gradually reduced latency times over the experiment lifetime by up to 14%, mainly due to the reconfiguration capabilities of the AMDS during its lifetime.

These results reinforce the potential of this software design as a datacentre enhancement and security utility, with virtually no impact on day-to-day operations due to the asynchronous nature of the implementation.

Bringing this thesis to a close, potential future research projects that could be based on the author's work are discussed.

9.3 Future Work

This chapter presents several ideas that could be part of future research projects. Such projects would utilise and expand the AMDS to further benefit a Cloud Computing environment.

One such idea is the creation of a module capable of monitoring individual Virtual Machines in an attempt to provide an even finer control over the optimisation process. This module would be capable of remotely accessing active Virtual Machines and retrieving information on processor loads, memory and storage usage, running processes, etc. This would allow the master AMDS Control and Botnet modules to make even better decisions regarding their individual tasks of optimising energy efficiency and detecting and blocking malicious threats.

Another idea is expanding the Botnet module to include more detection algorithms to help with analysing network traffic, as well as gain other monitoring capabilities, such as accessing active individual Virtual Machines and looking at running processes and processor load and perhaps comparing them to the VMs profile (eg. VM only used for static web pages, high processor load may indicate unusual activities).

List of Figures

List of Tables

List of Code Fragments

List of Equations

References

Anderson, J. Q.; Rainie, L. (2010) “The future of Cloud Computing - Pew Internet & American Life Project”. Schubert, L., Jeffery, K. and Neidecker-Lutz, B. (Eds.) Analysis, 1, 1-26, European Commission. Available at: http://pewinternet.org/Reports/2010/ the-future-of-CloudComputing.aspx

Andrews, G. R. (2000) “Foundations of Multithreaded, Parallel, and Distributed Programming”, Published by Addison–Wesley, ISBN 0-201-35752-6.

Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A. D. (2009) “Above the Clouds: A Berkeley View of Cloud Computing”. Science, 53 (UCB/EECS-2009-28), 07-013, Citeseer.

Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A.D.; Katz, R.; Konwinski, A.; Lee, G.; Pattersonm D.; Rabkin, A.; Stoica, I.; Zaharia, M. (2010) “Above the Clouds: A Berkeley View of Cloud Computing”. Communications of the ACM, 53(4), pp. 50-58.

Arora, P.; Wadhawan, R. C.; Ahuja, Er. S. P. (2012) “Cloud Computing Security Issues in Infrastructure as a Service”. International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 2, Iss. 1, Jan. 2012, ISSN: 2277 128X.

B. Bloom (1956) “Taxonomy of Educational Objectives”. Longmans Green, New York.

Bagci, F. (2014) "*Towards Performance and Power Management of Cloud Servers*". Information Technology: New Generations (ITNG), 2014 11th International Conference on, pp. 599-604, 7-9 April 2014, doi: 10.1109/ITNG.2014.70

Barnatt, C. (2012) “Cloud Computing”. Available at: http://explainingcomputers. com/cloud.html

Baun, C.; Kunze, M.; Nimis, J.; Tai, S. (2011) “Cloud Computing”. Berlin: Springer-Verlag.

Berl, A.; Gelenbe, E.; Di Girolamo, M.; Giuliani, G.; De Meer, H.; Quan Dang, M.; Pentikousis, K. (2009) “Energy-Efficient Cloud Computing”. The Computer Journal, 53(7): 1045-1051 first published online August 19 2009.

Binkley, J. R.; Singh, S. (2006) “An algorithm for anomaly-based botnet detection”. Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI), pp. 43-48.

Bornico, L.; Walden, I. (2011) “Ensuring competition in the clouds: the role of competition law?”. ERA Forum, 12(2), pp. 265-285.

Breeding, M. (2012) “Cloud Computing for Libraries”. Chicago: ALA TechSource.

Burns, R. C.; Khan, O.; Plank, J. S.; Pierce, W.; Huang, C. (2012) “Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads". FAST, p. 20.

Caragiozidis, M.; Mouratidis, N.; Kavadias, C.; Loupis, M.; Berger, M. (2008) "Design Methodology for a Modular Component Based Software Architecture". Computer Software and Applications, COMPSAC '08. 32nd Annual IEEE International on, pp. 1122-1127, July 28 2008-Aug. 1 2008.

Carroll, M., Kotzé, P.; Van Der Merwe, A. (2012) "Securing virtual and cloud environments". Cloud Computing and Services Science, pp. 73-90, Springer New York.

Chandrashekar, J., Orrin, S., Livadas, C., Schooler, E. M. (2009) "The Dark Cloud: Understanding and Defending Against Botnets and Stealthy Malware". Intel RTechnology Journal, 13(2).

Chen, X.; Wills, G. B.; Gilbert, L.; Bacigalupo, D. (2010). "Using cloud for research: a technical review". TeciRes Project, University of Southampton. Available at: http://www.jisc.ac.uk/media/documents/programmes/research_infrastructure/tecires_technical_report%20100608.pdf

Chou, D. C.; Chou, A. Y. (2011) "Seeking Sustainable Computing: The role of Cloud Computing". Southwest Decision Sciences Institute Conference.

Cirstea, M.N. (2003) "Problem Based Learning in Microelectronics". Int. Journal of Eng. Education, Vol. 19, No.5, 2003, pp.738-741, ISSN:0949-149X.

Cisco IOS NetFlow. Available at: http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html

Cloud Security Alliance (2010) "Top Threats to Cloud Computing V1.0". Available at: https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf.

CORDIS (2013) "Providing a platform for a coordinated response to cloud cybercrime". Available at: http://cordis.europa.eu/news/rcn/36249_en.html

Corrado, E.M.; Moulaison, H.L. (2011) "Getting Started with Cloud Computing". London: Facet Publishing.

Dinita, R. I., Wilson, G., Winckles, A., Cirstea, M., Jones, A. (2012) "A cloud-based virtual computing laboratory for teaching computer networks". Optimization of Electrical and Electronic Equipment (OPTIM), 2012 13th International Conference on, pp. 1314-1318, doi: 10.1109/OPTIM.2012.6231992.

DMTF (Distributed Management Task Force) (2011) "Open Virtualization Format OVF". Available at: http://www.dmtf.org/standards/ovf

Feller, E.; Simonin, M.; J´egou, Y.; Orgerie, A.-C.; Margery, D; et al. (2014) "*Snooze: A Scalable and Autonomic Cloud Management System*". [Research Report] RR-8649, Inria Rennes, 2014, pp.31.

Geada, D.; Dave, D. (2011) "The case for the heterogeneous cloud". Cloud Comput J, 11(3), 521-525.

Hamdaqa, M.; Tahvildari, L. (2012) "Cloud computing uncovered: a research landscape". Advances in Computers, 86, 41-85.

Hinkle, M. (2010) "Eleven Open-Source Cloud Computing Projects to Watch". SocializedSoftare. com, January, 10.

IETF (2009) "Architecture for IP Flow Information Export". Available online: https://tools.ietf.org/html/draft-ietf-ipfix-architecture-12

Irani, G.N.H.; Tawosi, V. (2011) "AAMA: A new Authentication and Authorization architecture for modular information systems, a robust object oriented approach". Application of Information and Communication Technologies (AICT), 2011 5th International Conference on, pp. 1-5, 12-14 Oct. 2011.

JISC (2010) "JISC Strategy for 2010-2012". Available at: http://www.jisc.ac.uk/ media/documents/aboutus/strategy/strategy1012.pdf

Joint Task Force for Computing Curricula (JTFCC) (2005) "Computing Curricula 2005 – The Overview Report". ACM, ISBN 1-59593-359-X, Available at: http://www.acm.org/education/curric_vols/CC2005-March06Final.pdf

Junyoung, H.; Jiman, H.; Yookun, C. (2009) "EARQ: Energy Aware Routing for Real-Time and Reliable Communication in Wireless Industrial Sensor Networks". 2009 IEEE Transactions on Industrial Informatics, vol. 5, no. 1, pp. 3-11, Feb. 2009.

Karim, A.; Salleh, R. B.; Shiraz, M.; Shah, S. A. A.; Awan, I.; Anuar, N. B. (2014) "*Botnet detection techniques: review, future trends, and issues*". Journal of Zhejiang University SCIENCE C, 15(11), pp. 943-983.

Ke, A.; Yu, Y.; Chen, Y.; Zhao, E.; Xie, Y.; Yu, F.; Gillum, Q. (2009) "BotGraph: large scale spamming botnet detection". Proceedings of the 6th USENIX symposium on Networked systems design and implementation (NSDI'09), USENIX Association, Berkeley, CA, USA, p321-334.

Kolb, D. A. (1984) "Experiential Learning: experience as the source of learning and development". Prentice-Hall, New-Jersey.

Kumar, K.; Weiqing, S.; Pratik, R.; Tianning, L.; Sekar, R. (2005) "V- NetLab: A Cost-Effective Platform to Support Course Projects in Computer Security". 9th Annual Colloquium for Information Systems Security Education (CISSE 05), June 2005.

Lee, Y.; Kang, W.; Lee, Y. (2011) “A hadoop-based packet trace processing tool”. Springer Berlin Heidelberg.

Liu, Z., Chen, F., Chen, Z., Xiang, L.; Yuan, Z. (2008) “Decentralized formation control of mobile agents: a unified framework”. Physica A: Statistical Mechanics and its Applications, 387(19), 4917-4926.

Livenson, I.; Laure, E. (2011) “Towards transparent integration of heterogeneous cloud storage platforms”. Proceedings of the fourth international workshop on Data-intensive distributed computing (pp. 27-34), ACM.

Maffei, A.; Hofmann, A. (2010) “From flexibility to true Evolvability: An introduction to the basic requirements”. Industrial Electronics (ISIE), 2010 IEEE International Symposium on, vol., no., pp.2658,2663, 4-7 July 2010.

Mahmood (2013) “Software Engineering Frameworks for the Cloud Computing Paradigm”. Springer London, ISBN 978-1-4471-5031-2, Chapter 13, pp. 283-301.

Matalon, S.; Klein, R.; Walls, C. (2011) “Embedded System Power Consumption: A Software or a Hardware Issue?”. Mentor Graphics, Available at: http://www.mentor.com/resources/techpubs/upload/mentorpaper_68962.pdf.

McKendrick, J. (2011) “Cloud computing’s vendor lock-in problem: Why the industry is taking a step backward”. Forbes, November.

Mell, P.; Grance, T. (2011) “The NIST Definition of Cloud Computing”. NIST Special Publication 800-145.

Minas, L.; Ellison, B. (2009) “The Problem of Power Consumption in Servers”. Dr. Dobb’s Journal, May 2009.

Mirashe, S. P.; Kalyankar, N. V. (2010) “Cloud Computing”. Communications of the ACM, 51 (7), pp. 9.

Mohammad, H. (2012) “Cloud Computing Uncovered: A Research Landscape”. Elsevier Press. pp. 41–85., ISBN 0-12-396535-7.

Mora, D.; Taisch, M.; Colombo; A. W. (2012) “Towards an energy management system of systems: An industrial case study”. IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society, pp. 5811-5816, 25-28 Oct. 2012.

Moretti, C.; Bulosan, J.; Thain, D.; Flynn, P.J. (2008) “All-pairs: An abstraction for data-intensive Cloud Computing”. Parallel and Distributed Processing, 2008, IPDPS 2008, IEEE International Symposium, vol., no., pp.1-11, 14-18 April 2008.

Murakami, J. (2008) “A hypervisor IPS based on hardware assisted virtualisation technology". Black Hat USA 2008.

OGF (Open Grid Forum) (2011) "Open Cloud Computing Interface OCCI". Available at: http://occi-wg.org/

Oh, J. H., Jeong, H. C.; Im, C. T. (2010) "Malware auto-analysis system and method using kernel callback mechanism". U.S. Patent Application 12/942,700.

Popek, G. J.; Goldberg, R. P. (1974) "Formal Requirements for Virtualisable Third Generation Architectures". Communications of the ACM, 17 (7), pp. 412–421.

Prieto-Blázquez, J.; Arnedo-Moreno, J.; Herrera-Joancomartí, J. (2008) "An Integrated Structure for a Virtual Networking Laboratory". IEEE Transactions on Industrial Electronics, Vol. 55, No. 6, June 2008.

Ramos-Paja, C.A.; Scarpetta, J. M. R.; Martinez-Salamero, L. (2010) "Integrated Learning Platform for Internet-Based Control-Engineering Education". IEEE Trans. on Industrial Electronics, vol. 57, no. 10, pp. 3284 - 3296, Oct 2010.

Renesse, R. van; Birman, K.P (2006) "Autonomic computing – a system-wide perspective", Autonomic Computing: Concepts, Infrastructure, and Applications, pp. 1–11.

Rutkowska, J.; Tereshkin, A. (2008) "Bluepilling the Xen Hypervisor". Black Hat USA 2008.

Scala Programming Language, http://www.scala-lang.org.

Sotomayor, B.; Montero, R.S.; Llorente, I.M.; Foster, I. (2009) "An open source solution for Virtual Infrastructure Management in Private and Hybrid Clouds". IEEE Internet Computing, 13(5), pp. 1-11.

Staten, J., Alvarez, V.; McKee, J. (2012) "Assess Your Cloud Maturity: The Cloud Computing Playbook". Forrester Research, Inc, Cambridge.

Tsutomu, N.; Yoshihiro, O.; Hideki, E.; Takahiro, S.; Kazuhiko, K. (2010) "Using a Hypervisor to Migrate Running Operating Systems to Secure Virtual Machines". IEEE 34th Annual Computer Software and Applications Conference (Proceedings), p37- 46.

Unified Modelling Language. Available at: http://www.uml.org.

Vada, E. T. (2012) "Creating flexible heterogeneous cloud environments". Available at: https://www.duo.uio.no/bitstream/handle/10852/34153/thesis.pdf

Vicente, A. G.; Muñ, Oz; I. B., Galilea; J. L. L.; del Toro, P. A. R. (2010) "Remote Automation Laboratory Using a Cluster of Virtual Machines". IEEE Trans. on Industrial Electronics, vol. 57, no. 10, pp. 3276 - 3283, Oct 2010.

VMWare ESXi. Available at: http://www.vmware.com/uk/products/esxi-and-esx/overview.html

VMWare vCenter Operations Management. Available at: http://www.vmware.com/uk/products/vrealize-operations/features.html

VMWare vSphere with Operations Management. Available at: http://www.vmware.com/uk/products/vsphere-operations-management/

VMware vSphere SDK for Java. Available at: http://communities.vmware.com/community/vmtn/developer/forums/java_toolkit.

Vouk, M.; et al.; (2009) "Using VCL Technology to implement distributed reconfigurable datacentres and computational service for educational institutes". ACM Digital.

Wang, L.; Laszewski, G.V.; Kunze, M.; Tao, J. (2008) "Cloud Computing: a perspective study". New Generation Computing, 28(2), pp.137-147.

Warkozek, G.; Drayer, E.; Debusschere, V.; Bacha, S. (2012) "A new approach to model energy consumption of servers in datacenters". 2012 IEEE International Conference on Industrial Technology (ICIT), pp.211-216, 19-21 March 2012.

Weiqing, S.; Varun, K.; Kumar, K.; Sekar, R. (2008) "V-NetLab: an approach for realizing logically isolated networks for security experiments". Proceedings of the conference on Cyber security experimentation and test (CSET'08), USENIX Association, Berkeley, CA, USA, Article 5, 6 pages.

Williams, D.; Weatherspoon, H.; Jamjoom, H.; Liu, Y. (2011) "Overdriver: Handling memory Overload in an Oversubscribed Cloud". VEE '11 Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments.

Winckles, A.; Spasova, K.; Rowsell, T. (2011) "Remote Laboratories and Reusable Learning Objects in a Distance Learning Context". Networks, Issue 14, January 2011.

Xen Project Hypervisor. Available at: http://wiki.xenproject.org/wiki/Xen_Overview#What_is_the_Xen_Project_Hypervisor.3F

Xiaobo, M.; Xiaohong, G.; Jing, T.; Qinghua, Z.; Yun, G.; Lu, L.; Shuang, Z. (2010) "A Novel IRC Botnet Detection Method Based on Packet Size Sequence". Communications (ICC), 2010 IEEE International Conference on, vol., no., pp.1,5, 23-27 May 2010, doi: 10.1109/ICC.2010.5502092.

Zeidanloo, H.; Shooshtari, M.; Amoli, P.; Safari, M.; Zamani, M. (2010) "A taxonomy of Botnet detection techniques". Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference v2, p158–162. IEEE.

Zhang, Q.; Cheng, L.; Boutaba, R. (2010) “Cloud Computing: state-of-the-art and research challenges”. Journal of Internet Services and Applications, 1(1), pp. 7-18.

Publications Based on this Work (see Appendix B)

1) Dinita, R. I., Winckles, A., Wilson, G., (2014) "**Use of NetFlow/IPFIX Botnet Detection Tools to Determine Placement for Autonomous VMs**". Cybercrime Forensics Education and Training (CFET), 2014 7th International Conference on, ISBN 97801909067158

2) Dinita, R. I., Wilson, G., Winckles, A., Cirstea, M., Rowsell, T. (2013) "**A Novel Autonomous Management Distributed System for Cloud Computing Environments**". Industrial Electronics Conference (IECON), 2013 39th Annual Conference of

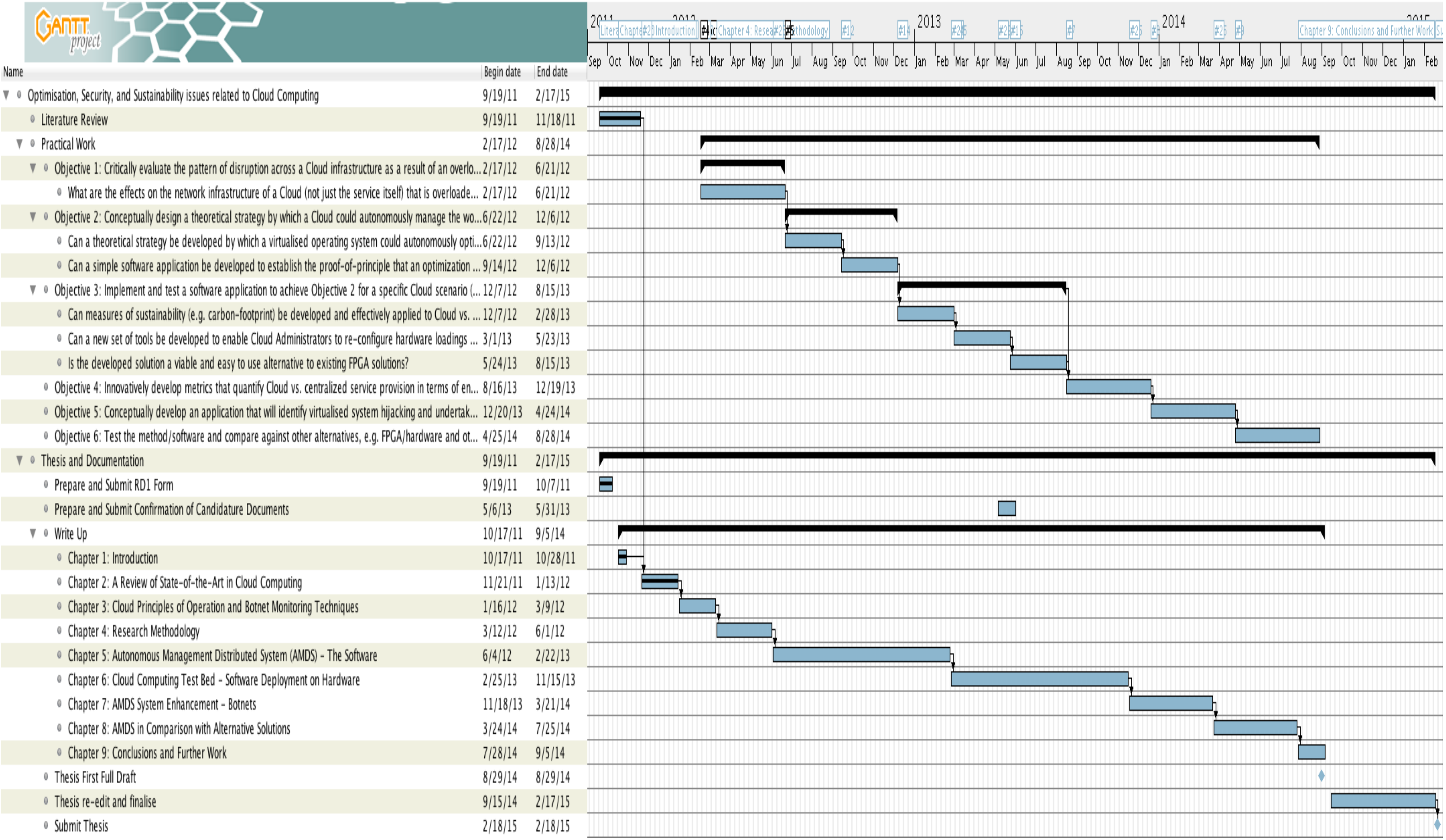
3) Dinita, R. I., Wilson, G., Winckles, A., Cirstea, M., Jones, A. (2013) "**Hardware Loads and Power Consumption in Cloud Computing Environments**". International Conference on Industrial Technology (ICIT), 2013, pp. 1291-1296, ISBN 978-1-4673-4568-2

4) Dinita, R. I., Wilson, G., Winckles, A., Cirstea, M., Jones, A. (2012) "**A cloud-based virtual computing laboratory for teaching computer networks**". Optimization of Electrical and Electronic Equipment (OPTIM), 2012 13th International Conference on, pp. 1314-1318, doi: 10.1109/OPTIM.2012.6231992

5) Dinita, R. I., Wilson, G., Winckles, A., Jones, A. (2012) "**Cloud Computing and Hardware Loads**". Anglia Ruskin University Research Student Conference, 6th Annual

List of Appendices

Appendix A – Research Management Gantt Chart



Name	Begin date	End date
Optimisation, Security, and Sustainability issues related to Cloud Computing	9/19/11	2/17/15
Literature Review	9/19/11	11/18/11
Practical Work	2/17/12	8/28/14
Objective 1: Critically evaluate the pattern of disruption across a Cloud infrastructure as a result of an overlo...	2/17/12	6/21/12
What are the effects on the network infrastructure of a Cloud (not just the service itself) that is overloade...	2/17/12	6/21/12
Objective 2: Conceptually design a theoretical strategy by which a Cloud could autonomously manage the wo...	6/22/12	12/6/12
Can a theoretical strategy be developed by which a virtualised operating system could autonomously opti...	6/22/12	9/13/12
Can a simple software application be developed to establish the proof-of-principle that an optimization ...	9/14/12	12/6/12
Objective 3: Implement and test a software application to achieve Objective 2 for a specific Cloud scenario (...	12/7/12	8/15/13
Can measures of sustainability (e.g. carbon-footprint) be developed and effectively applied to Cloud vs. ...	12/7/12	2/28/13
Can a new set of tools be developed to enable Cloud Administrators to re-configure hardware loadings ...	3/1/13	5/23/13
Is the developed solution a viable and easy to use alternative to existing FPGA solutions?	5/24/13	8/15/13
Objective 4: Innovatively develop metrics that quantify Cloud vs. centralized service provision in terms of en...	8/16/13	12/19/13
Objective 5: Conceptually develop an application that will identify virtualised system hijacking and undertak...	12/20/13	4/24/14
Objective 6: Test the method/software and compare against other alternatives, e.g. FPGA/hardware and ot...	4/25/14	8/28/14
Thesis and Documentation	9/19/11	2/17/15
Prepare and Submit RD1 Form	9/19/11	10/7/11
Prepare and Submit Confirmation of Candidature Documents	5/6/13	5/31/13
Write Up	10/17/11	9/5/14
Chapter 1: Introduction	10/17/11	10/28/11
Chapter 2: A Review of State-of-the-Art in Cloud Computing	11/21/11	1/13/12
Chapter 3: Cloud Principles of Operation and Botnet Monitoring Techniques	1/16/12	3/9/12
Chapter 4: Research Methodology	3/12/12	6/1/12
Chapter 5: Autonomous Management Distributed System (AMDS) - The Software	6/4/12	2/22/13
Chapter 6: Cloud Computing Test Bed - Software Deployment on Hardware	2/25/13	11/15/13
Chapter 7: AMDS System Enhancement - Botnets	11/18/13	3/21/14
Chapter 8: AMDS in Comparison with Alternative Solutions	3/24/14	7/25/14
Chapter 9: Conclusions and Further Work	7/28/14	9/5/14
Thesis First Full Draft	8/29/14	8/29/14
Thesis re-edit and finalise	9/15/14	2/17/15
Submit Thesis	2/18/15	2/18/15

Appendix B – Research Publications based on this Work

A Cloud-based Virtual Computing Laboratory for Teaching Computer Networks

Razvan I. Dinita, *Member, IEEE,* George Wilson, Adrian Winckles, Marcian Cirstea, *Senior Member, IEEE*, Aled Jones
Anglia Ruskin University, Cambridge, UK
razvan.dinita@anglia.ac.uk, george.wilson@anglia.ac.uk, adrian.winckles@anglia.ac.uk, marcian.cirstea@anglia.ac.uk, aled.jones@anglia.ac.uk

***Abstract*-This paper presents a novel 'Cloud-based' solution for teaching computer networks in an educational context. One key advantage of the system is its ability to comission and decomission virtual infrastructures comprised of routers, switches and virtual machines on demand. It makes use of hardware located in different physical locations, VMWare software to manage the virtual resources and NetLab+ to manage the configuration of multiple different virtual scenarios. The key features of the cloud infrastructure are described and evaluated.**

I. INTRODUCTION

Nowadays, students are increasingly attracted to computers, the internet and all networking involved in fully exploiting the increasingly larger computing resources available through the internet. Students are coming with a wide variety of technical and cultural background, motivation, age, experience and learning styles and all of these must be taken into account either by course extension – in order to establish a common foundation, or by lecturers in higher education adapting their expectations and the programmes they offer.

As the first method normally requires extra resources, such as extra teaching staff hours, availability of rooms, technical support, etc., it is the last one which is normally preferred because it is cost and time effective [1]. Therefore, the necessity arises to reconsider the teaching and learning strategies, and to adapt them to the learning styles and constraints of modern university environments.

Cloud computing is an emerging technology that devolves computing resources to the Internet [2]. The traditional teaching of computer networking in a higher educational institution is very much dependent on local hardware resources. The student experience of different computer networks will often be limited to the local hardware infrastructure currently available. That architecture cannot be easily changed because of the resource issues involved.

At the university the authors are affiliated with, there are multiple IT infrastructures and software solutions put in place that aim to assist and provide study platforms for students. A few examples are the VLE (Virtual Learning Environment) built using Microsoft products, Moodle (Modular Object-Oriented Dynamic Learning Environment), which is an open source platform, WebCT (Course Tools) etc. However, all these systems only provide basic means of interaction such as file sharing, discussion boards, questionnaires, blogs and access to course materials. These systems lack the means to allow students to conduct experiments, test taught theories and create innovative applications in the context of their chosen modules.

This paper focuses on presenting a cloud-computing solution for teaching computer networks, in which the network teaching resource is itself virtualised. A similar approach has already been researched [3], having a proven track record with over 80% positive student feedback. This paper aims to pick up where that solution left off and take it one step further through development of the underlying infrastructure and expansion of the overlying software platform.

This kind of solution is not meant to replace the traditional hands-on student experience, but rather provide a useful and cost-effective supplementary educational tool to support the teaching of computer networks (including by distance learning).

There have been other similar attempts at using NetLab+ in an Academic context as illustrated and described by [4], [5] and [6]. Although the software solutions used are similar, the test bed it has been deployed on makes the solution proposed by the authors of this paper a novel approach, which allows for implementation and testing of complex networking and security scenarios.

There have even been other similar attempts using clusters of Virtual Machines (VMs) [7] to virtualise automation learning topics based on the Moodle platform. The issue with that approach was that it was put together using multiple bits of technologies which could potentially allow a number of security breaches. It is also quite difficult to maintain, when it comes to expanding the infrastructure by adding more VMs or updating the Moodle platform.

Another virtual learning implementation was attempted by the authors of [8] through the implementation of four different virtual environments, each specialised in a different area of control-engineering. It makes use of a mix of open source Linux software and Matlab licensed software to provide the students with the means to implement and test different scenarios. Again, this kind of solution is difficult to maintain since it makes use of completely different software packages which have not been specifically designed to work together.

TABLE I
TEST BED COMPRISING HARDWARE

Make	Specifications	Quantity
HP Proliant	108GB RAM 2 x Intel Six-Core Xeon 2.4GHz 10 Gbit NICs	4
Dell R710	38GB RAM 2 x Intel Quad-Core Xeon 2.4GHz Gbit NICs	2
Viglen	16GB RAM 2x Intel Dual-Core Xeon 2.2GHz Gbit NICs	1
HP Filer	8TB HDD 10 Gbit NICs	1
Dell Storage Area Network	1.5TB	3
HP Integrated Lights Out	Gbit NICs	4
Cisco 4948 Switch	Gbit NICs	1
Cisco 3560 Switch	Gbit NICs	1
Cisco 5510 Adaptive Security Appliance (ASA)	Gbit NICs	1
NetLab Pod	N/A, proprietary appliance	4

II. OBJECTIVES IN A PEDAGOGICAL THEORY CONTEXT

From a business perspective there are three well-known technologies associated with Cloud Computing, namely Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS). The current research aims to promote another cloud service technology in providing an educational resource that will focus on delivery of simulated network hardware resources (the concept of Laboratory-as-as-Service (LaaS)). This distinguishes the work from traditional virtual learning environments in common use in educational establishments today.

The development of this lab setup is taking into account Kolb's well established descriptive model of the Adult learning process, consisting of four stages [9]: Concrete Experience is followed by Reflective Observation which generates Abstract Conceptualisation; this leads to Active Experimentation which will generate a new Concrete Experience. Abstract Conceptualisation can be stimulated better in lecture sessions, addressing the cognitive type of objectives (such as memory, interpretation) in accordance with Bloom's taxonomy [10].

On the other hand, the Active Experimentation, Concrete Experience and Reflective Observation are better addressed in interactive laboratory sessions, which concentrate more on the other cognitive components identified by Bloom (Translation, Application, Analysis, Evaluation) and the Affective aspects of learning (Responding, Valuing and Organization in particular).

The paper is presenting research work which led to a new teaching and learning method that addresses this second aspect, by the development of a novel style Laboratory setup, based on cloud computing, providing a modern educational resource.

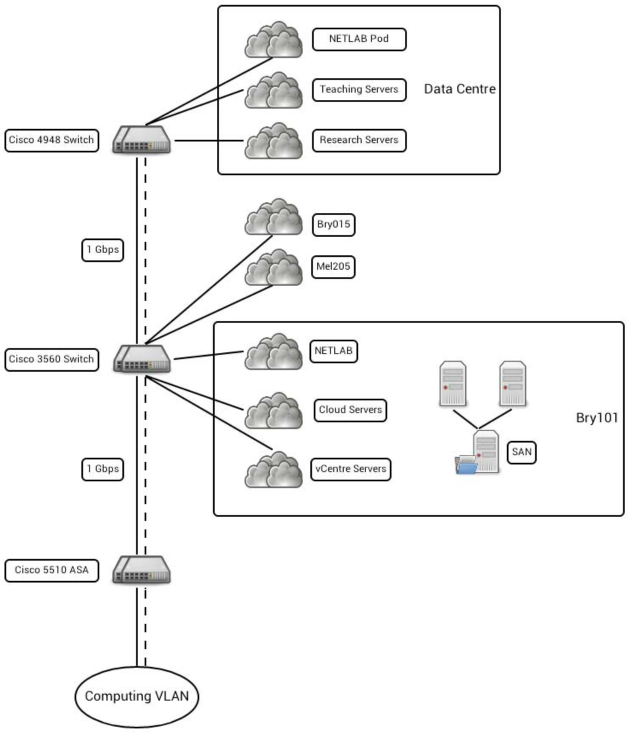


Fig. 1. Cloud Computing Test Bed – custom design and built infrastructure.

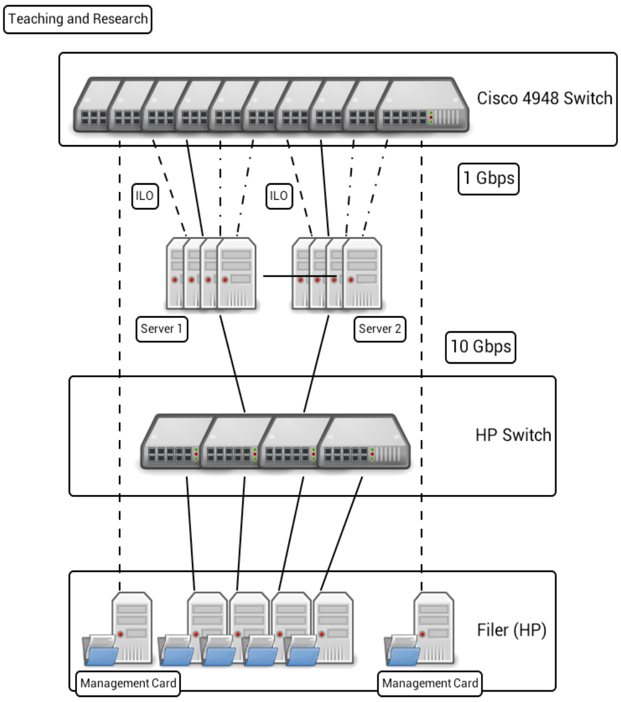


Fig. 2. Cloud Computing Test Bed – Datacentre view.

III. Methodology

A. *Hardware Infrastructure*

The authors currently have a small but powerful test bed comprising seven physical servers (mix of HP Proliant, Dell R710 and Viglen brands), three Storage Area Networks (HP) and multiple routers and switches (CISCO and HP), and HP ILOs (Table I) located within the Department of Computing and Technology (Figs. 1 and 2 are illustrating the structure of these systems) across four rooms (Bry015, Mel205, Data Center, Bry101).

Two of the servers are 'public'-facing (that is, they are utilised for teaching purposes), whilst the others are 'private'-facing (i.e. they are utilised for research – not yet implemented). The private-facing resource is isolated from the Internet and only accessible from departmental computers via an appropriate security protocol. All of the networking cards have Gigabit type interfaces (>=1Gbps, Gigabit per second, transfer rate) and the link between the Servers and the Filer (Fig. 2) is one of 10Gbps, which facilitates rapid data movement to and from the Filer across the network.

The described test bed allows for the possibility of testing unique and complex scenarios such as simulating a global infrastructure, where each room is considered to be a remote location in a different country/part of the world.

B. *Software Administration*

Both commercially available and Open Source software is used to manage both the underlying physical network infrastructure and the virtualized appliances running on that physical network.

The Test Bed resource is currently being developed in two ways:

i) Proprietary off-the-shelf remote laboratory systems such as NDG's NetLab are being used to offer complex IT systems (such as VMware vSphere/vCentre/ESXi courses) which can be easily deployed as virtual based solutions on demand (solution currently fully implemented).

ii) Configuration of an Open Source Apache Virtual Computing Laboratory (VCL) in a network distributed environment to support commissioning of reusable operating resources, as described by Vouk [11]. This provides a cloud computing based solution for network security laboratory teaching scenarios (solution still in development).

VMWare products such as the ESXi Hypervisor and vCentre are used to manage the physical infrastructure and run the VMs, while NDG's NetLab+ software is used to manage virtual appliances ranging from simple VMs to CISCO Routers and Switches, commissioning them on demand.

The reasoning behind choosing VMWare and NDG's proprietary software is that they have a proven track record of performing well and exceeding expectations in academic environments. According to [3] the majority of students responded very positive to the virtual labs they had the opportunity of installing, configuring and managing. They also believed that it has enhanced their learning experience through hand-on virtual sessions using said software solutions.

C. *NDG's NetLab+*

NetLab+ is able to commission and decommission entire virtual laboratories containing any number and combination of simple VMs, Routers and Switches (Fig. 3, 4, 5). This makes it an ideal environment for testing a great range of scenarios from simple networking to complex security configurations.

NetLab+ gives access to the virtual resources based on username/password combinations assigned by the Administrator account. These are made available to students on different network and security modules as needed.

One key advantage of this solution is instant access and high availability at any hour of the day, account holders being able to connect to NetLab+ via the public Computing VLAN (Fig. 1).

The Computing VLAN provides public access to numerous internal services and other networks, including the proposed virtual infrastructure. The front-end server makes use of several NICs to interconnect the various internal networks. The internal networks have the following IP classes: 169.x.x.x for NetLab+, 10.141.x.x for the VMWare solution and 10.x.x.x for the rest of the university services (VLE, Moodle, eVision etc.). Each individual network makes use of its own authentication framework, almost all of which allow login using a single university set of credentials handed out to every student and staff member.

Once authenticated, the students can proceed to reserve a Lab session (Fig. 6), choosing one of the many available virtual configurations (Fig. 3, 4, 5).

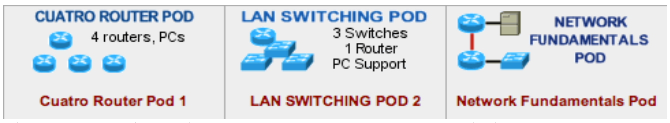


Fig. 3. NetLab+ Lab setups: 4 Routers, PCs; 3 Switches, 1 Router, PCs; Network Fundamentals setup (2 Routers, 1 Switch, PCs).

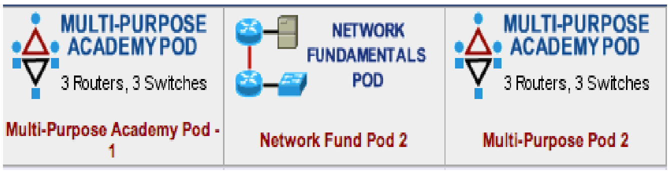
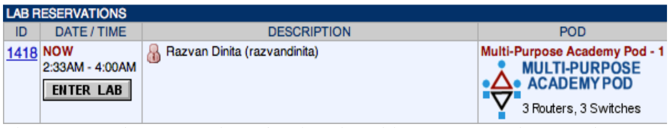


Fig. 4. NetLab+ Lab setups: 3 Routers, 3 Switches; Network Fundamentals setup (2 Routers, 1 Switch, PCs).

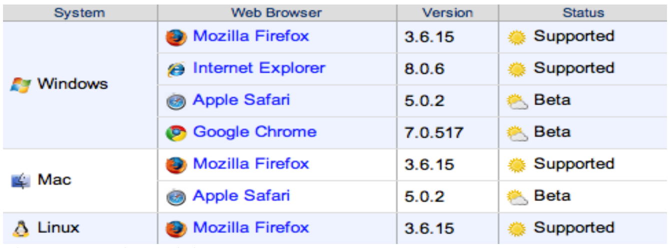
Fig. 5.NetLab+ Lab setups: ICM (Install, Configure, Manage) Master and Standard Pods.



LAB RESERVATIONS

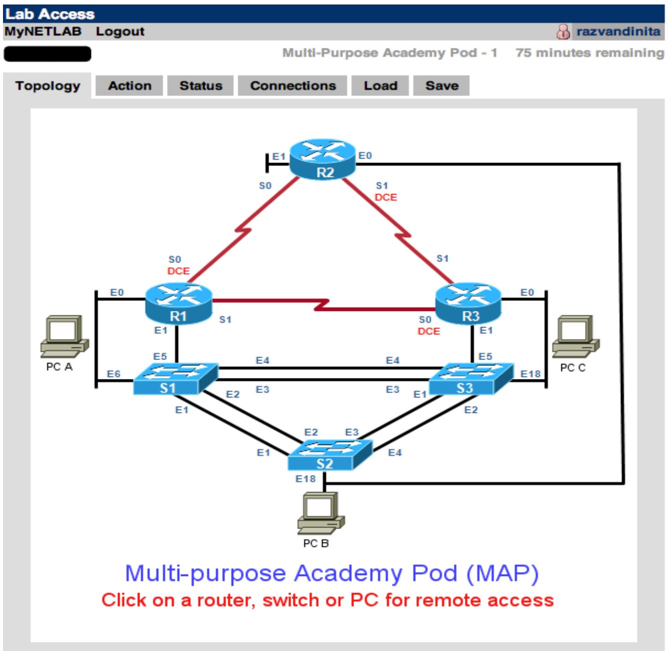
ID	DATE / TIME	DESCRIPTION	POD
1418	NOW 2:33AM - 4:00AM ENTER LAB	Razvan Dinita (razvandinita)	Multi-Purpose Academy Pod - 1 MULTI-PURPOSE ACADEMY POD 3 Routers, 3 Switches

Fig. 6. NetLab+ reservation of a virtual Multi-purpose Academy Pod.



System	Web Browser	Version	Status
Windows	Mozilla Firefox	3.6.15	Supported
	Internet Explorer	8.0.6	Supported
	Apple Safari	5.0.2	Beta
	Google Chrome	7.0.517	Beta
Mac	Mozilla Firefox	3.6.15	Supported
	Apple Safari	5.0.2	Beta
Linux	Mozilla Firefox	3.6.15	Supported

Fig. 7. NetLab+ web browser support.



Lab Access

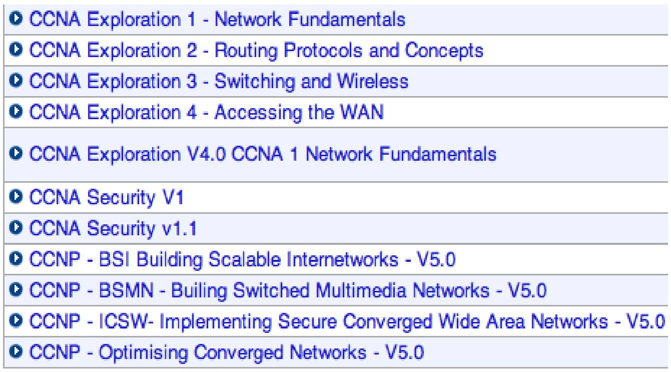
MyNETLAB Logout razvandinita

[redacted] Multi-Purpose Academy Pod - 1 75 minutes remaining

Fig. 8. Layout of a NetLab+ virtual Multi-purpose Academy Pod.

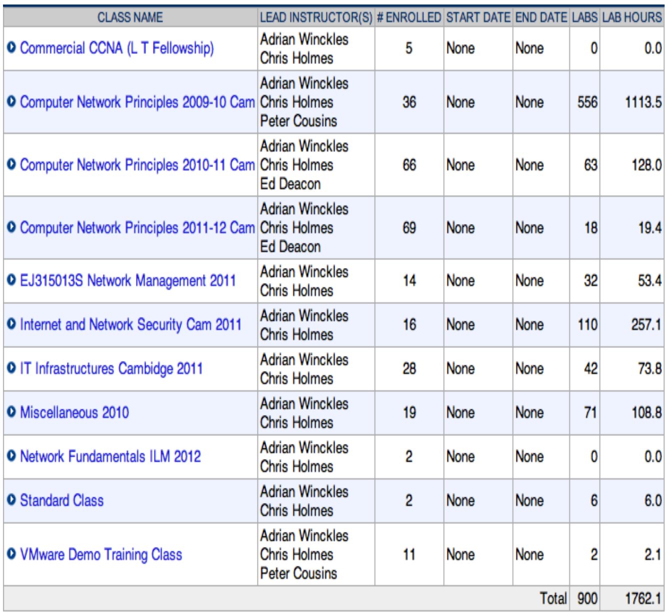
Many of the popular browsers (Fig. 7) can be used to fully interact with any part of the commissioned Virtual Lab setup. It makes heavy use of JavaVM to offer a rich GUI interface which allows easy management and configuration of any of the virtual appliances. For example the user can simply click R1/2/3 or S1/2/3 or PC A/B/C (Fig. 8) to bring up a completely interactive Console Window.

Every Lab configuration also offers the possibility of either remembering changes made each session or running fresh configurations every time. It even allows easy export of configuration progress at any time during its operation.



- CCNA Exploration 1 - Network Fundamentals
- CCNA Exploration 2 - Routing Protocols and Concepts
- CCNA Exploration 3 - Switching and Wireless
- CCNA Exploration 4 - Accessing the WAN
- CCNA Exploration V4.0 CCNA 1 Network Fundamentals
- CCNA Security V1
- CCNA Security v1.1
- CCNP - BSI Building Scalable Internetworks - V5.0
- CCNP - BSMN - Builing Switched Multimedia Networks - V5.0
- CCNP - ICSW- Implementing Secure Converged Wide Area Networks - V5.0
- CCNP - Optimising Converged Networks - V5.0

Fig. 9. NetLab+ available Labs.



CLASS NAME	LEAD INSTRUCTOR(S)	# ENROLLED	START DATE	END DATE	LABS	LAB HOURS
Commercial CCNA (L T Fellowship)	Adrian Winckles Chris Holmes	5	None	None	0	0.0
Computer Network Principles 2009-10 Cam	Adrian Winckles Chris Holmes Peter Cousins	36	None	None	556	1113.5
Computer Network Principles 2010-11 Cam	Adrian Winckles Chris Holmes Ed Deacon	66	None	None	63	128.0
Computer Network Principles 2011-12 Cam	Adrian Winckles Chris Holmes Ed Deacon	69	None	None	18	19.4
EJ315013S Network Management 2011	Adrian Winckles Chris Holmes	14	None	None	32	53.4
Internet and Network Security Cam 2011	Adrian Winckles Chris Holmes	16	None	None	110	257.1
IT Infrastructures Cambidge 2011	Adrian Winckles Chris Holmes	28	None	None	42	73.8
Miscellaneous 2010	Adrian Winckles Chris Holmes	19	None	None	71	108.8
Network Fundamentals ILM 2012	Adrian Winckles Chris Holmes	2	None	None	0	0.0
Standard Class	Adrian Winckles Chris Holmes	2	None	None	6	6.0
VMware Demo Training Class	Adrian Winckles Chris Holmes Peter Cousins	11	None	None	2	2.1
				Total	900	1762.1

Fig. 10. NetLab+ module classes with instructors, number of enrolled students and number of Labs and Lab Hours completed.

IV. RESULTS

Since there are a large number of Labs (training courses and materials) available as part of the NetLab+ solution, students have been granted access to the NetLab+ setup from the very beginning of its first deployment on the previously described test bed. Students enrolled on a wide variety of Network and Security modules have been testing the setup and at the same time learning how to install, configure and manage a wide range of Cisco appliances.

As it can be seen in Fig. 10, there already have been 900 Labs and over 1700 hours of lab work put into testing the described NetLab+ solution by over 260 students, all of which have ran without any issues.

At university level, there is an academic process called Module Evaluation, which is a simple survey presented to enrolled students. This gives students the opportunity to comment anonymously on various aspects of their modules. Student feedback plays an important role in quality enhancement. The findings of these surveys informs of both successful and unsuccessful teaching and learning practices.

In current context, the students who took part in testing of the proposed virtual infrastructure were enlisted on the following modules: Computer Network Principles, Network Technologies and Network Management.

Recently, they have been presented with a Module Evaluation survey for each of the mentioned modules. The survey results have exceeded all expectations, as almost all students believe that they have had their study experience enhanced by the proposed virtual infrastructure.

The students, however, have expressed several ideas for improvement:

- Better mobile access. Currently, since the NetLab+ solution is written almost entirely in Java there is almost no mobile support as of this writing. Of course, having better mobile access would increase student satisfaction levels as this would allow them to quickly login and check on projects running on commissioned VLabs, or simply continue a previously started Lab.

- Access to more physical networking hardware. Currently, there is only one rack containing networking hardware available to students to interact with. Having access to physical hardware allows students to better understand the Labs they have access to through much needed hand-on experience with cabling and configuring real-world networking equipment. This would ultimately improve their chances of getting employed once completing their course.

V. Future Work

Future work will involve development of the system administration of the Cloud Computing Test Bed, including:

- Expansion of the private-facing hardware (addition of more PCs – some are already available).
- Development of new software modules to manage disparate hardware and operating systems across a network.

Currently, as already mentioned, only the teaching side solution of this proposed solution has been fully implemented. The authors will implement the research solution in due course, which will provide even more flexibility by supporting fully customised virtualised appliances to be deployed and tested.

One proposed use of the Test Bed will be to develop optimization software which would improve energy sustainable cloud based solutions. Such software would be capable of autonomous decision making based on hardware loads, ultimately perhaps resulting in the movement of VMs across the network, be it to another server in the local cluster or one located in a different geographical location.

VI. Conclusions

Both VCL and NetLab+ solutions are capable of delivering an automated and self-maintained virtualised remote computing environment to cater for students' need, with very little ongoing administration.

Whilst VCL provides a highly scalable, flexible and very cost effective solution, it is limited in terms of the complexity of the solutions potentially offered. NetLab+ provides a more managed solution, better able to provide the complexity and flexibility that more advanced computer science courses may require, a premise supported by the setup shown in Fig. 10.

Also, in the long run, a NetLab+ setup will be easy to maintain and expand as NetLab+ has been specifically been designed to work with the VMWare vCentre/ESXi solutions, as opposed to using other virtualisation platforms alongside different open source software [7], [8].

References

[1] Cirstea, M.N., (2003), "Problem Based Learning in Microelectronics", *Int. Journal of Eng. Education*, Vol. 19, No.5, 2003, pp.738-741, ISSN:0949-149X.

[2] Mirashe, S. P., and Kalyankar, N. V. (2010), "Cloud Computing," [*Communications of the ACM*], *51*(7), 9.

[3] Winckles, A., Spasova, K., Rowsell, T. (2011), "Remote Laboratories and Reusable Learning Objects in a Distance Learning Context", *Networks*, Issue 14, January 2011.

[4] Prieto-Blázquez, J., Arnedo-Moreno, J., Herrera-Joancomartí, J. (2008), "An Integrated Structure for a Virtual Networking Laboratory", *IEEE Transactions on Industrial Electronics*, Vol. 55, No. 6, June 2008.

[5] Kumar, K., Weiqing, S., Pratik, R., Tianning, L., Sekar, R. (2005), "V-NetLab: A Cost-Effective Platform to Support Course Projects in Computer Security", *9th Annual Colloquium for Information Systems Security Education* (CISSE 05), June 2005.

[6] Weiqing, S., Varun, K., Kumar, K., Sekar, R. (2008), "V-NetLab: an approach for realizing logically isolated networks for security experiments", *Proceedings of the conference on Cyber security experimentation and test* (CSET'08), USENIX Association, Berkeley, CA, USA, , Article 5 , 6 pages.

[7] Vicente, A.G., Muñ, Oz, I.B., Galilea, J.L.L., del Toro, P.A.R. (2010), "Remote Automation Laboratory Using a Cluster of Virtual Machines", *IEEE Trans. on Industrial Electronics*, vol. 57, no. 10, pp. 3276 - 3283, Oct 2010.

[8] Ramos-Paja, C.A., Scarpetta, J.M.R., Martinez-Salamero, L. (2010), "Integrated Learning Platform for Internet-Based Control-Engineering Education", *IEEE Trans. on Industrial Electronics*, vol. 57, no. 10, pp. 3284 - 3296, Oct 2010.

[9] Kolb, D.A. (1984), "Experiential Learning: experience as the source of learning and development", Prentice-Hall, New-Jersey.

[10] B. Bloom, (1956), "Taxonomy of Educational Objectives", Longmans Green, New York.

[11] Vouk, M. et al. (2009). "Using VCL Technology to implement distributed reconfigurable data centres and computational service for educational institutes", *ACM Digital*, [Online].

Cloud Computing and Hardware Loads

Razvan-Ioan Dinita, Dr. George Wilson, Adrian Winckles, Dr. Aled Jones
Department of Computing and Technology, Faculty of Science and Technology

Abstract – *This paper describes an optimised and novel approach to an Autonomous Virtual Server Management System in a 'Cloud Computing' environment. One key advantage of this system is its ability to improve hardware power consumption through autonomously moving virtual servers around a network to balance out hardware loads. This has a potentially important impact on issues of sustainability with respect to both energy efficiency and economic viability. Another key advantage is the improvement of the overall end-user experience for services within the Cloud. This is currently being investigated through configuration of a cloud-computing test-bed rig. The key features of this and some predictions of what may be achieved are described and evaluated.*

Keywords: *Cloud Computing, Hardware Loads, Optimisation*

I. Introduction

Cloud computing is an emerging technology that devolves computing resources to the Internet [6]. This paper focuses on conceptually presenting an innovative approach to the resilience and optimization of Internet/network usage via mobility of virtualized servers within a cloud. The optimised running of a cloud will be investigated through the use of a virtual networking laboratory to develop new metrics that can be tested to quantify the processing efficiency (and therefore carbon footprint) of an optimised cloud network compared to a non-optimised cloud network and to a non-cloud infrastructure [1][2].

The background to the objectives of this work will now be discussed. The protocols that enable current network topologies to interface in such a way as to support Cloud functionality are well established, however the effect of an unanticipated amount of people trying to access the same file/service is poorly understood [11]. Whilst management tools are available to enable Cloud Administrators to re-configure hardware loadings according to service demands the approach is largely by-trial-and-error [7]. One consequence of the adoption of Cloud Computing in the commercial sector is that companies do not need to invest in their own hardware infrastructure. This has environmental consequences and their quantification is important for developing strategies for a sustainable environment [4]. A final area of interest concerns security. A botnet is a group of compromised computers connected to the Internet. Each compromised computer is called a bot, and could include individual virtual bots within a virtualised system. Whilst there are tools to protect a Cloud from such malware attacks, a very poorly researched area is how a successful hijacking of a Cloud's virtual operating system could be identified [3][5][8][9][12].

II. Objectives

This research will focus on the optimization and security of Internet/network usage via mobility of virtualized servers within a cloud. At least one cloud-resident application-specific prototype utility such as a virtual networking laboratory will be developed. Potential applications of such a virtual laboratory will allow suitable metrics to be proposed and tested that can quantify the reduction in the carbon footprint of the IT sector compared to a non-cloud infrastructure.

Objective 1: To critically evaluate the pattern of disruption across a Cloud infrastructure as a result of an overloaded service request. The effects on the network infrastructure of a Cloud (not just the service itself) that is overloaded by a service request will be studied. The results will be logged and analysed.

Objective 2: To conceptually design a theoretical strategy by which a Cloud could autonomously manage the workloads placed on that infrastructure, and implement and test a software application to achieve this aim for a specific Cloud scenario. Ideally a theoretical strategy will be developed by which a virtualised operating system can autonomously optimize the location of virtualized servers within a network. A simple software application to establish this proof-of-principle will be built.

Objective 3: To innovatively develop metrics that quantify Cloud vs. centralized service provision in terms of environmental sustainability. Measures of sustainability (e.g. carbon-footprint) will be developed and effectively applied to Cloud vs. non-Cloud scenarios.

Objective 4: To develop an application that will identify virtualized system hijacking and undertake a range of appropriate activities from simple notification to service suspension. Parameters indicative of the successful hijacking of a Cloud will be identified. A software tool will be developed to monitor the activities of a Cloud in such a way as to be able to identify the successful hijacking of the virtualized servers, and to mitigate the presence of a compromised Cloud.

III. Methodology

The authors currently have a small test bed comprising seven physical servers (mix of HP Proliant, Dell R710 and Viglen brands), three Storage Area Networks (HP) and multiple routers and switches (CISCO and HP), and HP ILOs located within the Department of Computing and Technology[1] (Fig. 1 and Fig. 2). Two of the servers are 'public'-facing (that is, they are utilised for teaching purposes) whilst the others are 'private'-facing (i.e. they are utilised for research). The private-facing resource is isolated from the Internet and only accessible from departmental computers via an appropriate security protocol. All of the networking cards have Gigabit type interfaces (>=1Gbps, Gigabit per second, transfer rate).

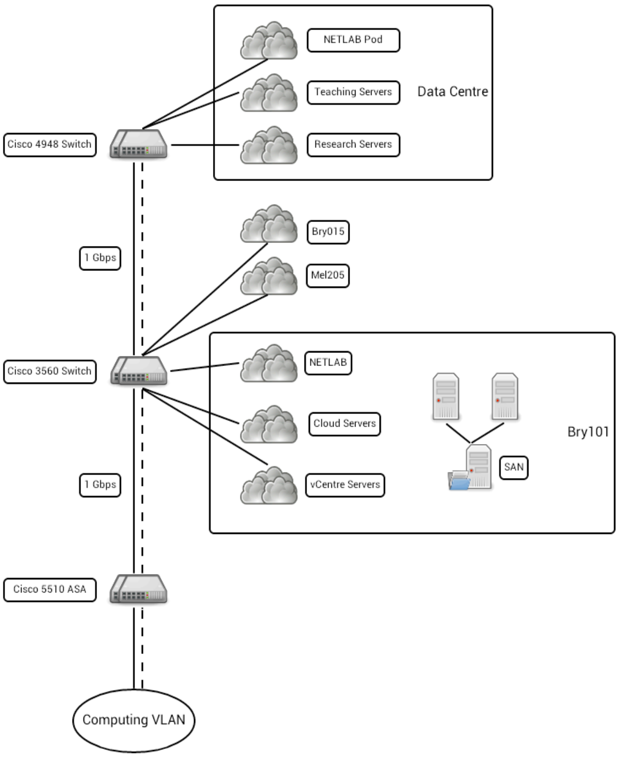


Fig. 1 Cloud Computing Test Bed – custom designed and built infrastructure.

The indicative research methods will focus on development of this resource, including:

- Expansion of the private-facing hardware (addition of more PCs – some are already available).
- Development of new software modules to manage disparate hardware and operating systems across a network.
- Building of cloud optimization strategies via a top-down approach using a proprietary software virtualization product (VMWARE) [10].
- Building of cloud optimization techniques via a bottom-up approach using an appropriate mainstream programming language (such as Java due to its library support tools for building bespoke network applications).
- Development of a software tool to detect and manage malware attacks on the virtualized servers.

Objective 1: To be achieved through heavy use of the test bed. Multiple virtual Machines (VMs) will be created on all four servers. Simultaneous requests will be made to all VMs to simulate a memory overload. Both hardware and network loads will be monitored, logged and analysed.

Objective 2: To be achieved through socket programming using the Java programming language. Sockets locally bind to specific port numbers, which allows direct communication to other (virtual) machines over a (virtual) network. The resulting software application will be installed on each server included in the test bed. It will be capable of directly communicating with other instances of itself and relaying information across the (virtual) network. The software application will have a modular design. As such, each task it will be capable of performing will represent one module. This design will allow easy expansion of its capabilities to allow it to perform more complex operations within a cloud network. It will also allow easy maintenance of each module and of the software application as a whole. The software application will also be capable of autonomously making decisions based on current hardware and network loads to move VMs around from one physical location to another within the same virtual network.

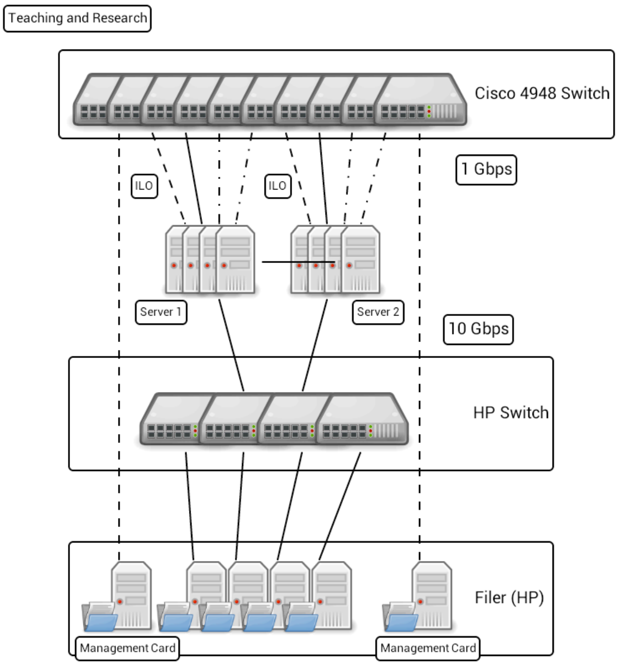


Fig. 2 Cloud Computing Test Bed – Data Centre view.

[1] Anglia Ruskin University, East Rd, Cambridge CB1 1PT, UK

Another way of implementing objective 2 is by looking at currently available software such as the VMWare vMotion technology to achieve the same end result - either by using a pure VMWare solution or by creating a hybrid solution comprised of VMWare technologies and a set of custom built technologies.

Moving a VM across a network to a different physical location raises the following concerns that will be addressed:

1) Maintaining and Updating the IP address tables across the network so that loss of service is avoided.

2) Ensuring the MAC addresses of the VMs are unique at the new physical location, refreshing them if clashes occur.

3) Refreshing the VM's Domain Name so that it matches the one at the new physical location.

Objective 3: To be achieved by measuring server and network hardware power consumption and logging the results. The analysis of these results will enable the authors to take steps towards developing the required metrics. An attempt will also be made towards identifying what levels of efficiency savings can be achieved through optimization of the cloud network. This will be a natural consequence of the aforementioned analysis.

Objective 4: To be achieved through testing of different approaches to service hijacking as presented here [14]. An approach to overcoming these security issues is presented here [13]. This will be investigated and, if relevant, a solution based on the Public Key Infrastructure will be built. Otherwise, alternative approaches to the aforementioned security issues will be researched and developed.

IV. Practical view

Power consumption is directly influenced by a great deal of factors [15][16]. The factors that this paper will focus on are the Processor, RAM and Hard Disk. These are the most relevant in the context of cloud computing because the Processor does calculations, stores the results in the RAM and it finally, if necessary, commits them to the Hard Disk. According to [15] the Processor ("*CPU Quadcore*") and the RAM ("*Memory*") have the highest power consumption of all the hardware types in an actively used server system. There is also a strong relation between hardware loads and power consumption [15][16]. The less hardware load on a system, the more efficiently the server operates and the less power it consumes.

Thus, the approach this work takes will help reduce the hardware loads by attempting to move 'hot' data (data actively used at any given time) from a highly active server (many user service requests) to a less active server (few user service requests). This approach aims to balance out the hardware loads across the same virtual network and at the same reduce the overall power required to achieve the same results as before the move.

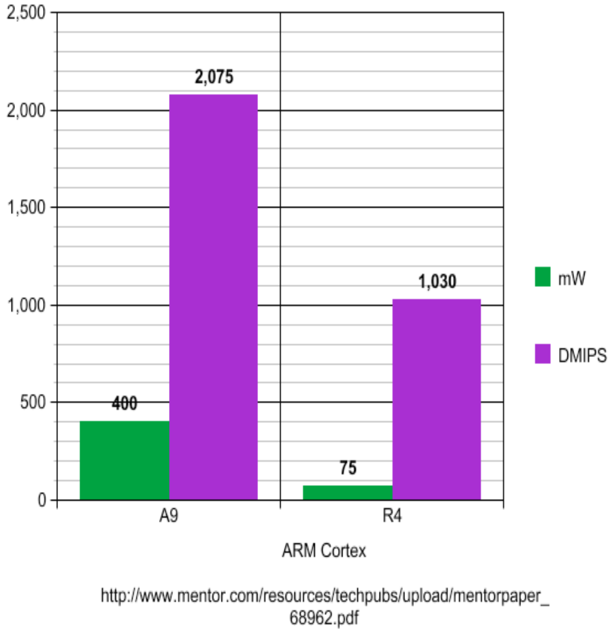


Fig. 3 Relation between miliWatts (mW) and the Millions of Instructions/Second (DMIPS). Data taken from [16].

V. Future work

Since the research is in its early stages, future work is mainly based on the conceptual ideas presented in this paper. Causes of disruption will be examined to investigate whether solutions can be implemented to ensure the resilient operation of a cloud network.

According to Fig. 3, processors consume four times as much power running at full speed (GHz) as running at half speed. Thus, another approach might be to attempt to reduce the Processor clock speed in addition to moving virtual Machines across the network. This however would depend on whether the VMWare Hypervisor (operating system running on the servers) gives direct access to this type of setting. Another issue is that reducing the processor clock speed in order to reduce the power consumption by a factor of at least 3x is only practical in an environment where speed is not mission-critical.

VI. Conclusions

This paper has described conceptual means and ideas of optimising and improving the resilience of cloud computing environments. The authors have discussed how cloud computing networks can be improved to help reduce the carbon footprint of the associated hardware through implementing an autonomous solution to help manage the hosted virtual machines.

References

[1] Anderson, J. Q. (2010), “The future of cloud computing,” Schubert, L., Jeffery, K. and Neidecker-Lutz, B. (Eds.) *Analysis*, *1*, 1-26, European Commission.

[2] Armbrust, M., Fox, A., Griffith, R., and Joseph, A. D. (2009), “Above the Clouds : A Berkeley View of Cloud Computing,” *Science*, *53* (UCB/EECS-2009-28), 07-013, Citeseer.

[3] Chandrashekar, J. (2009), “The Dark Cloud: Understanding and Defending Against Botnets and Stealthy Malware,” *Intel RTechnology Journal*, 13(2).

[4] Chou, D. C. and Chou, A. Y. (2011), “Seeking Sustainable Computing: The role of Cloud Computing,” *Southwest Decision Sciences Institute Conference*.

[5] Ke, A., Yu, Y., Chen, Y., Zhao, E., Xie, Y., Yu, F. and Gillum, Q. (2009), “BotGraph: large scale spamming botnet detection,” [*Proceedings of the 6th USENIX symposium on Networked systems design and implementation* (NSDI'09)], USENIX Association, Berkeley, CA, USA, p321-334.

[6] Mirashe, S. P., and Kalyankar, N. V. (2010), “Cloud Computing,” [*Communications of the ACM*], *51*(7), 9.

[7] Moretti, C., Bulosan, J., Thain, D., Flynn, P.J. (2008), “All-pairs: An abstraction for data-intensive cloud computing,” *Parallel and Distributed Processing, 2008*, IPDPS 2008. [*IEEE International Symposium*, vol., no., pp.1-11, 14-18 April 2008].

[8] Murakami, J. (2008), “A hypervisor IPS based on hardware assisted virtualization technology,” [*Black Hat USA 2008*].

[9] Rutkowska, J. and Tereshkin, A. (2008), “Bluepilling the Xen Hypervisor,” [*Black Hat USA 2008*].

[10] Tsutomu, N., Yoshihiro, O., Hideki, E., Takahiro, S. and Kazuhiko, K. (2010), “Using a Hypervisor to Migrate Running Operating Systems to Secure Virtual Machines,” [*2010 IEEE 34th Annual Computer Software and Applications Conference (Proceedings)* p37-46].

[11] Williams, D., Weatherspoon, H., Jamjoom, H. and Liu, Y. (2011), “Overdriver: Handling memory Overload in an Oversubscribed Cloud,” [*VEE '11 Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*].

[12] Zeidanloo, H., Shooshtari, M., Amoli, P., Safari, M., and Zamani, M. (2010), “A taxonomy of Botnet detection techniques,” Computer Science and Information Technology (ICCSIT), [*2010 3rd IEEE International Conference* v2, p158–162. IEEE].

[13] Arora, P., Wadhawan, R. C., Ahuja, Er. S. P. (2012), “Cloud Computing Security Issues in Infrastructure as a Service,” [*International Journal of Advanced Research in Computer Science and Software Engineering*] Vol. 2, Iss. 1, Jan. 2012, ISSN: 2277 128X.

[14] Cloud Security Alliance (2010), “Top Threats to Cloud Computing V1.0,” Available at: https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf

[15] Minas, L., Ellison, B. (2009), “The Problem of Power Consumption in Servers,” [*Dr. Dobb’s Journal*, May 2009].

[16] Matalon, S., Klein, R., Walls, C. (2011), “Embedded System Power Consumption: A Software or a Hardware Issue?,” Mentor Graphics, Available at: http://www.mentor.com/resources/techpubs/upload/mentorpaper_68962.pdf

Hardware Loads and Power Consumption in Cloud Computing Environments

Razvan I. Dinita, *Member, IEEE,* George Wilson, Adrian Winckles, Marcian Cirstea, *Senior Member, IEEE*, Aled Jones
Anglia Ruskin University
Cambridge, UK
razvan.dinita@anglia.ac.uk, george.wilson@anglia.ac.uk, adrian.winckles@anglia.ac.uk, marcian.cirstea@anglia.ac.uk, aled.jones@anglia.ac.uk

***Abstract*-This paper describes an optimised and novel approach to an Autonomous Virtual Server Management System in a 'Cloud Computing' environment and it presents a set of preliminary test results. One key advantage of this system is its ability to improve hardware power consumption through autonomously moving virtual servers around a network to balance out hardware loads. This has a potentially important impact on issues of sustainability with respect to both energy efficiency and economic viability. Another key advantage is the improvement of the overall end-user experience for services within the Cloud. This has been investigated through the configuration of a cloud-computing test-bed rig. The key features of this rig and some predictions of what may be achieved with it are described and evaluated.**

I. Introduction

Cloud computing is an emerging technology that devolves computing resources to the Internet [1]. This paper focuses on presenting a conceptually innovative approach to the resilience and optimization of Internet/network usage via mobility of virtualized servers within a cloud. The optimised running of a cloud has been investigated through the use of a virtual networking laboratory, developing new metrics that can be tested to quantify the energy efficiency (and therefore carbon footprint) of an optimised cloud network compared to a non-optimised cloud network and to a non-cloud infrastructure [2-3].

Although the protocols that enable current network topologies to interface in such a way as to support Cloud functionality are well established the effect of an unanticipated amount of people trying to access the same file/service is not yet well understood [4]. The authors of one study break down the energy consumption in a data centre and suggest different ways of reducing it overall [5]. Whilst management tools are available to enable Cloud Administrators to re-configure hardware loadings according to service demands, the approach is largely by-trial-and-error [6].

One consequence of the adoption of Cloud Computing in the commercial sector is that companies do not need to invest in their own hardware infrastructure. This has environmental consequences and their quantification is important for developing strategies for a sustainable environment [7]. Security is also a relevant issue. A botnet is a group of compromised computers connected to the Internet. Each compromised computer is called a bot, and could include individual virtual bots within a virtualised system. Whilst there are tools to protect a Cloud from such malware attacks, a less widely researched area is that of methods of identification of a successful hijacking of a Cloud's virtual operating system [8-9-10]. These and other relevant works [11-12] also discuss the hypervisor architecture and the possible vulnerabilities within them.

II. Research Focus

This research focuses on the optimization and security of Internet/network usage via mobility of virtualized servers within a cloud. A cloud-resident application-specific prototype utility such as a virtual networking laboratory has been developed. Potential applications of such a virtual laboratory will allow suitable metrics to be proposed and tested that can quantify the reduction in the carbon footprint of the IT sector compared to a non-cloud infrastructure.

Objective 1: Critically evaluate the pattern of disruption across a Cloud infrastructure as a result of an overloaded service request. The effect on the network infrastructure of a Cloud (not just the service itself) that is overloaded by a service request is of particular interest here.

Objective 2: Conceptually design a theoretical strategy by which a Cloud could autonomously manage the workloads placed on that infrastructure, and implement and test a software application to achieve this aim for a specific Cloud scenario. A theoretical strategy has been developed by which a virtualised operating system can autonomously optimize the location of virtualized servers within a network. A simple software application to establish this proof-of-principle is currently being built.

Objective 3: Innovatively develop metrics that quantify Cloud vs. centralized service provision in terms of environmental sustainability. Measures of sustainability (e.g. carbon-footprint) are currently being developed and effectively applied to Cloud vs. non-Cloud scenarios.

Objective 4: Develop an application that will identify virtualized system hijacking and undertake a range of appropriate activities from simple notification to service suspension. Parameters indicative of the successful hijacking of a Cloud will be identified. A software tool is currently being developed to monitor the activities of a Cloud in such a way as to be able to identify the successful hijacking of the virtualized servers, and to mitigate the presence of a compromised Cloud.

III. METHODOLOGY

There is very little current literature in the public domain as to how load balancing is achieved in data centres. So far, none of the most recent papers have explored the connection between processor loads, power consumption and VM migration. The authors of this work propose an original methodology to deal with this issue.

A small test bed has been constructed comprising seven physical servers (mix of HP Proliant, Dell R710 and Viglen brands), three Storage Area Networks (HP) and multiple routers and switches (CISCO and HP), and HP ILOs located within the Department of Computing and Technology[1] (Fig. 1 and Fig. 2). Two of the servers are 'public'-facing (that is, they are utilised for teaching purposes) whilst the others are 'private'-facing (i.e. they are utilised for research). The private-facing resource is isolated from the Internet and only accessible from departmental computers via an appropriate security protocol. All of the networking cards have Gigabit type interfaces (>=1Gbps, Gigabit per second, transfer rate).

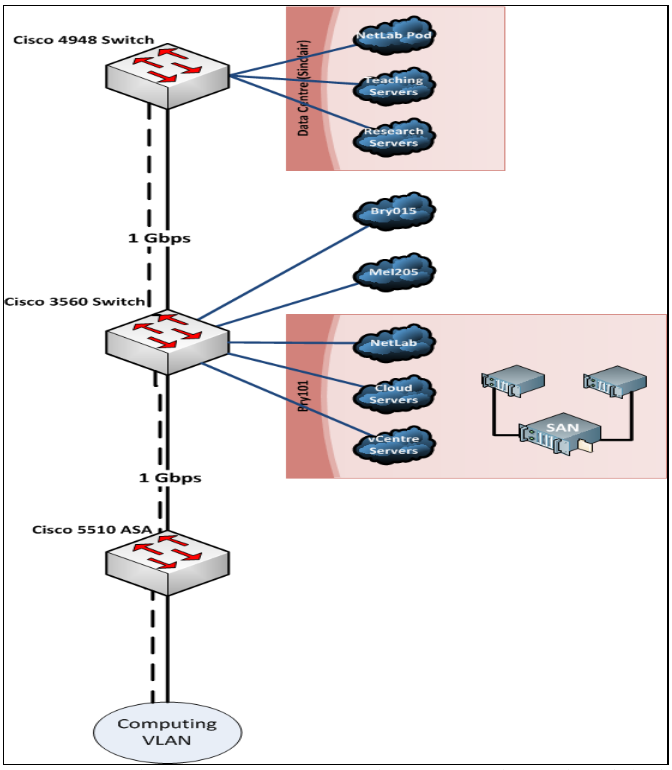


Figure 1 - Cloud Computing Test Bed – custom designed and built infrastructure.

In the context of the present research the development of this resource has focussed on the following:

- Expansion of the private-facing hardware (addition of more PCs – some are already available).
- Development of new software modules to manage disparate hardware and operating systems across a network.
- Building of cloud optimization strategies via a top-down approach using a proprietary software virtualization product (VMWARE) [13].
- Building of cloud optimization techniques via a bottom-up approach using an appropriate mainstream programming language (Scala due to its library support tools for building bespoke network applications).
- Development of a software tool to detect and manage malware attacks on the virtualized servers.

An appraisal of the extent to which the given objectives have been met will now be discussed.

Objective 1: Has been achieved through experimental work using the test bed. Multiple virtual Machines (VMs) have been created on all four servers. Simultaneous requests have been made to all VMs to simulate a memory overload. Both hardware and network loads have been monitored (using a wide range of hardware appliances such as ILOs, PDUs and UPSes), logged (using Microsoft Excel spreadsheets) and analysed. Relevant methodology presented by the authors of paper [5] has also been taken into account which specifically describes issues related to increasing hardware loads on a few servers while at the same time shutting down unneeded ones to save power.

The authors have access to several test hardware that allow simple power consumption readings to be taken through proprietary User Interfaces: HP Integrated Lights-Out (ILO), APC Power Distribution Unit (PDU) and APC Uninterrupted Power Supply (UPS) devices. These devices are directly connected to the physical servers and as such produce relevant power metrics. The power consumption readings will be correlated with processor utilisation readings taken from within the VMWare vSphere Client. Tables will be produced detailing the connection between processor utilisation and power consumption.

Another way of gathering power consumption readings is through the SNMP protocol. All hardware presented in this paper is fully network enabled and as such have been provided with unique IP addresses to which connections can be made. The specialised hardware relevant to this point all support the SNMP protocol (communication protocol that allows two way data connections). A software module is currently being built using the Scala programming language capable of taking advantage of this feature. It is capable of retrieving real time power readings through the SNMP protocol and store them in a database. All collected data is analysed and the results are ultimately be presented in tables. The module also allows sorting and comparing the different results gathered up to and including the most recent ones. This allows a history of readings to be generated relevant to future research.

Objective 2: Achieved through socket programming using the Scala programming language. Sockets locally bind to specific port numbers, which allows direct communication to

[1] Anglia Ruskin University, East Rd, Cambridge CB1 1PT, UK

other (virtual) machines over a (virtual) network. The resulting software application is able to communicate with each server included in the test bed. It is also be capable of directly communicating with other instances of itself and relaying information across the (virtual) network.

The software application has a modular design. As such, each task it is capable of performing represents one module. This design allows easy expansion of its capabilities to allow it to perform more complex operations within a cloud network. It also allows easy maintenance of each module and of the software application as a whole.

The software application will also be capable of autonomously making decisions based on current hardware and network loads to move VMs around from one physical location to another within the same virtual network. The main goal of the application is to reduce the overall power consumption of the cluster of computers it is directly overseeing.

Another approach to objective 2 is by looking at currently available commercial software such as the VMWare vMotion technology to achieve the same end result - either by using a pure VMWare solution or by creating a hybrid solution comprised of VMWare technologies and a set of custom built technologies.

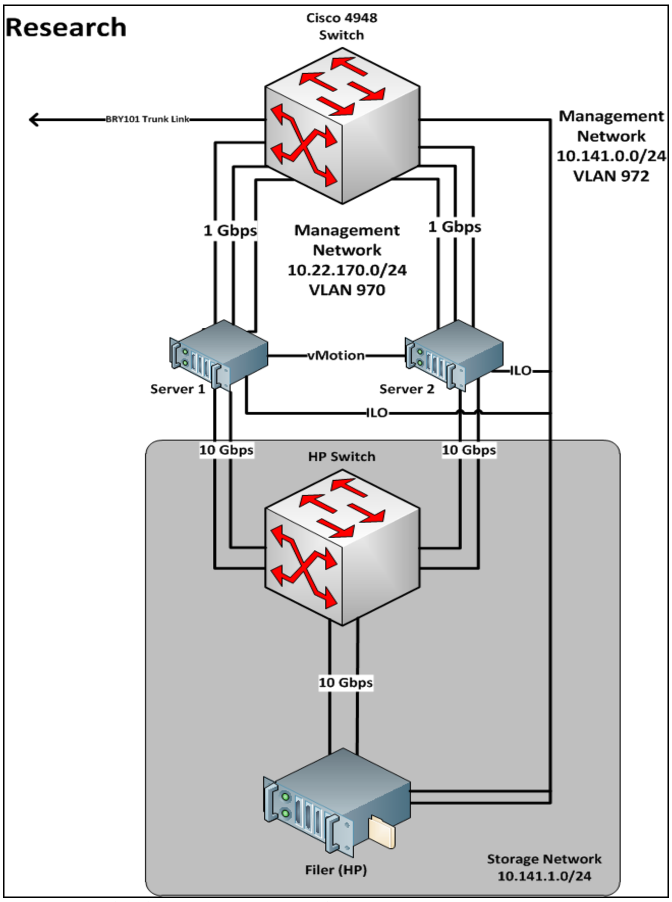


Figure 2 - Cloud Computing Test Bed – Datacentre view.

Moving a VM across a network to a different physical location raises the following difficulties concerns that are currently being addressed:

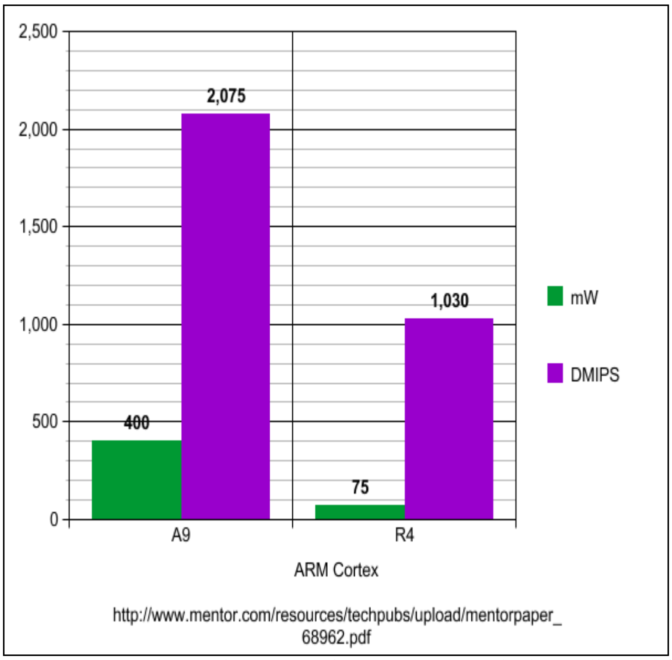
1) Maintaining and Updating the IP address tables across the network so that loss of service is avoided.

2) Ensuring the MAC addresses of the VMs are unique at the new physical location, refreshing them if clashes occur.

3) Refreshing the VM's Domain Name so that it matches the one at the new physical location.

Objective 3: Achieved by measuring server and network hardware power consumption and logging the results. The analyses of these results form the basis for developing the required metrics. An attempt is also being made towards identifying what levels of efficiency savings can be achieved through optimization of the cloud network.

Objective 4: To be achieved through testing of different approaches to service hijacking as presented here [14]. Some previous work on overcoming relevant security issues has been done [15]. That work will be investigated and, if relevant, a solution based on Public Key Infrastructure will be built. Otherwise, alternative approaches to the aforementioned security issues will be researched and developed.



http://www.mentor.com/resources/techpubs/upload/mentorpaper_
68962.pdf

Figure 3 - Relation between miliWatts (mW) and the Millions of Instructions/Second (DMIPS). Data taken from paper [16]

IV. TEST RESULTS

A. Practical View

Power consumption is directly influenced by a great deal of factors [16], and [17]. The factors that this paper will focus on are the Processor, RAM and Hard Disk. These are most relevant in the context of cloud computing because the Processor does calculations, stores the results in the RAM and if applicable commits them to the Hard Disk. Network power consumption is also relevant but since it is necessary and cannot be efficiently controlled its' effects have not been included in this work.

TABLE I. WATT SERVER POWER READINGS VS PROCESSOR LOADS

Processor Load (PL)	0% (Idle)	25%	50%	75%	100%
Watts / Hour	124	168	191	217	239
Watts Power Consumption Rise (WPCR) (%)	0	35.5	54	75	92
Efficiency (PL / WPCR) (%)	0	70	92	100	108

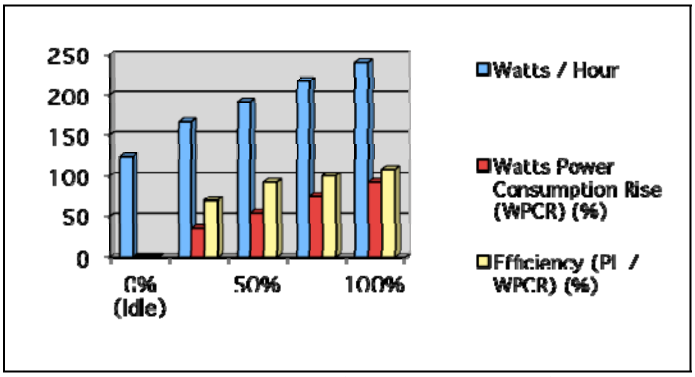


Figure 4 - Watt Server Power Readings vs. Processor Loads

TABLE II. AMPS SERVER POWER READINGS VS PROCESSOR LOADS

Processor Load (PL)	0% (Idle)	25%	50%	75%	100%
Amps	1.2	1.55	1.8	1.95	2.2
Amps Power Consumption Rise (APCR) (%)	0	29.1	50	62.5	83
Efficiency (PL / APCR) (%)	0	86	100	120	120.4

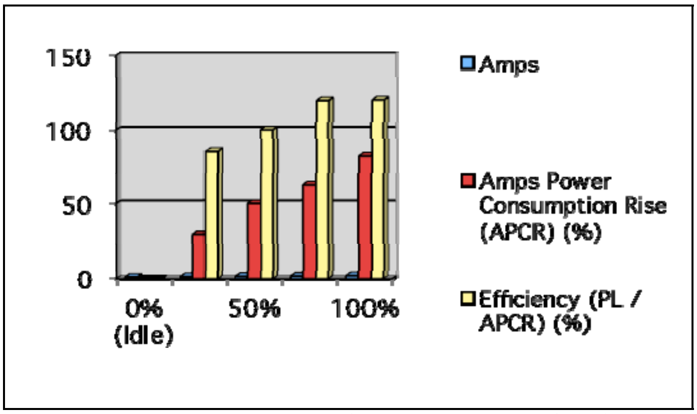


Figure 5 - Amps Server Power Readings vs. Processor Loads (PL)

According to [17] the Processor ("*CPU Quadcore*") and the RAM ("*Memory*") have the highest power consumption of all the hardware types in an actively used server system. There is also a strong relation between hardware loads and power consumption according to papers [16] and [17]. The less hardware load on a system, the more efficiently the server operates and the less power it consumes. Furthermore, paper [18] discusses the importance of CPU loads and takes the approach of making the distinction between the server's own CPU load and application generated CPU load. Thus, the approach this work takes helps reduce the hardware loads by attempting to move 'hot' data (data actively used at any given time) from a highly active server (many user service requests) to a less active server (few user service requests). This approach aims to balance out the hardware loads across the same virtual network and at the same reduce the overall power required to achieve the same results as before the move.

A second approach currently being considered is moving data from less active servers to highly active servers, maximising processor loads. This approach aims to reduce the number of active servers and as such cut down the overall power consumption of idle to moderately used servers.

A similar technique of energy efficient data transfer across a set of network nodes has been investigated by the authors of paper [19]. It describes a set of energy efficient rules that comprise the EARQ protocol which are being followed whenever the control application needs to decide which node to transmit the data next. These rules have been abstracted and considered for Objective 2's application logic.

B. Hardware Stress Tests

In order achieve Objective 1 the authors have put together a special Linux Virtual Machine template capable of performing server load stress tests. This is achieved by having the VM push the Virtual Processor to loads of up to 100%.

The VM comprises of the latest Ubuntu Linux operating system as well as the latest versions of the Apache Web Server and the Tsung open source multi-protocol distributed load testing tool.

Apache is a web service that upon start it listens on a predefined port (usually 80) on the server. It is capable of delivering multiple web pages to millions of clients simultaneously. A simple web page has been created and put in place for the purpose of these tests.

'Tsung' is a complex application that is capable of creating millions of simultaneous connections (also known as virtual clients) to any given web service. The configuration file allows fine control over the length of each connection time wise as well as how fast the number of simultaneous connections grows over a predefined time frame. This allows measurement of server power consumption while the processor load ranges from 0% to 100%. Other features of Tsung are beyond the scope of this research.

Upon launch the VM starts up both Apache and Tsung. Apache is set up to listen on port 80 for connections from anywhere on the network (in this case, from within the same location – localhost or 127.0.0.1). Tsung is set up to create virtual clients for Apache every 0.5 milliseconds over a time frame of 15 minutes. By the end of the given time frame the

TABLE III. Power Consumption Estimates with 100% Uptime

Period of Time (Months)	1 (±732h)	3 (±2196h)	6 (±4392h)	12 (±8784h)
100% PL (239 Watts / Hour) (MW)	0.175	0.524	1.05	2.1
75% PL (217 Watts / Hour) (MW)	0.159	0.477	0.953	1.9
50% PL (191 Watts / Hour) (MW)	0.14	0.42	0.839	1.68
25% PL (168 Watts / Hour) (MW)	0.123	0.369	0.738	1.48
0% PL (124 Watts / Hour) (MW)	0.91	0.272	0.545	1.09

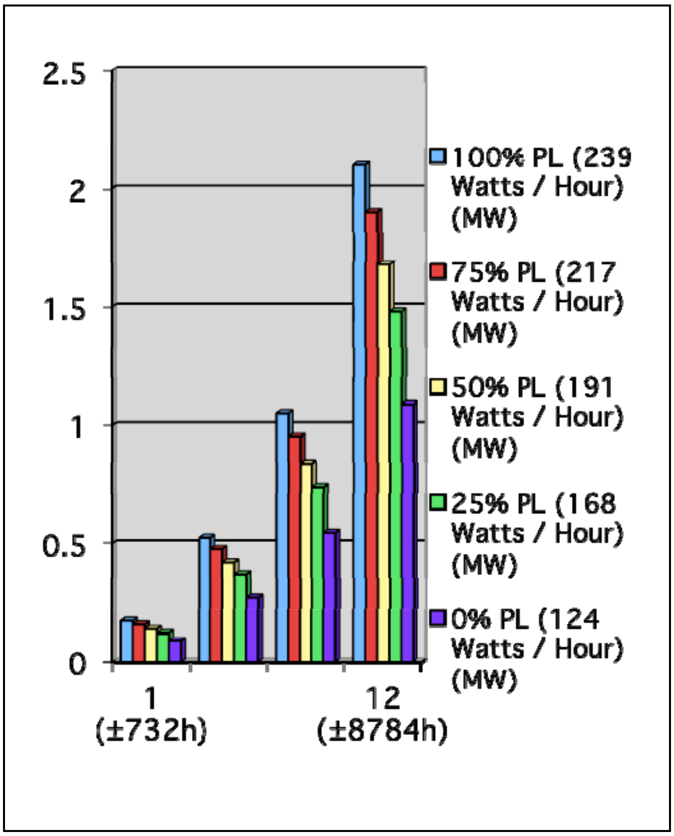


Figure 6 - Power Consumption estimates (100% Uptime) vs. Processor Loads (PL)

number of simultaneous connections will mount up to 1.8 million. This forces the Virtual Processor to slowly go through the needed loads.

For the purpose of these tests the authors have deployed 40 VMs based on the original template VM and have launched them all at the same time to achieve the desired effect.

C. Results

The readings gathered from both the VMWare vSphere client and the relevant hardware devices have been put side by side in the Tables I and II and graphs have been generated based on them in Figures 4 and 5. Estimates of power consumption over longer periods of time have been generated in Table III with associated data graphed in Figure 6. The results show that when

When taking Processor Load percentages are compared with Watt Consumption Rise percentages the higher the Processor Load gets, the more efficient the Watt Consumption becomes. At 100% Processor Load there is only a 92% increase in Watt Consumption. The same applies for Processor Load percentages and Amps Consumption percentages. For 100% Processor Load there is only an 83% increase in Amps Consumption.

For optimal efficiency it appears to be desirable that the Processor Load be kept over 75% for the best Power Consumption efficiency.

From the perspective of Data Centres it appears to be more cost effective to maximise the Processor Load on each server and running less servers rather than having a higher number of servers but with lower Processor Load percentages. Furthermore, using the results presented in Table III the cost of operation from the energy efficiency point of view can be calculated. This was for the following Data Centre use case scenarios; Case 1 (UC1): one server running at 100% PL (based on new findings) versus Case 2 (UC2): four servers running at 25% PL each (original theory). Both use cases assume an operation time span of one year and the price per Kwh used in the calculations is 111.1 Euros / MW [2] (current at the time of writing). The calculated costs are presented in Table IV.

TABLE IV. Cost estimates for two Use Case scenarios

Use Case 1 (UC1) One Server @ 100% PL	2.1 MW * 111.1 Euros / MW = 233.3 Euros
Use Case 2 (UC2) Four Servers @ 25% PL	One Server: 1.9 MW * 111.1 Euros / MW = 211.1 Euros Four Servers: 211.1 Euros * 4 = 844.4 Euros
| UC1 – UC2 | (Euros)	611.1 Euros UC2 shows a 261.9% cost increase compared to UC1

Table IV shows that UC2 exhibits a 261.9% operational cost increase compared to UC1. This is consistent with the results presented in Table III, which show a substantial economic benefit in Data Centre operation costs.

[2] Price taken from http://www.businesselectricityprices.com/kwh.php on 27th November 2012. Original price was shown in British Pence / KWh. http://www.xe.com/ucc/ was used to convert the price to Euro Cents / KWh. Result was then multiplied by 10 to get to Euros / MWh.

V. Conclusions

This paper has described novel development based on original concepts and ideas which optimise and improve the resilience of cloud computing environments. The authors have discussed how cloud computing networks can be improved to help reduce the carbon footprint of the associated hardware through implementing an autonomous solution to help manage the hosted virtual machines. The authors have also provided clear test results related to Power Consumption reduction at different Processor Load percentages and provided solutions, as well as calculated potential operation cost differences between two use cases.

According to Fig. 3, processors consume four times as much power running at full speed (GHz) as running at half speed. Thus, another approach might be to attempt to reduce the Processor clock speed in addition to moving virtual Machines across the network. This however would depend on whether the VMWare Hypervisor (operating system running on the servers) gives direct access to this type of setting. Another issue is that reducing the processor clock speed in order to reduce the power consumption by a factor of at least 3x is only practical in an environment where speed is not mission-critical.

Some future work can be explored on the causes of and whether solutions can be implemented to ensure the resilient operation of a cloud network.

Following to the results presented in this paper the authors have already started development of the aforementioned software application.

References

[1] Mirashe, S. P., and Kalyankar, N. V. (2010), "Cloud Computing," [*Communications of the ACM*], *51*(7), 9.

[2] Anderson, J. Q. (2010), "The future of cloud computing," Schubert, L., Jeffery, K. and Neidecker-Lutz, B. (Eds.) *Analysis*, *1*, 1-26, European Commission.

[3] Armbrust, M., Fox, A., Griffith, R., and Joseph, A. D. (2009), "Above the Clouds□: A Berkeley View of Cloud Computing," *Science*, *53* (UCB/EECS-2009-28), 07-013, Citeseer.

[4] Williams, D., Weatherspoon, H., Jamjoom, H. and Liu, Y. (2011), "Overdriver: Handling memory Overload in an Oversubscribed Cloud," [*VEE '11 Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*].

[5] Berl, A., Gelenbe, E., Di Girolamo, M, Giuliani, G., De Meer, H., Quan Dang, M., Pentikousis, K. (2009), "Energy-Efficient Cloud Computing", The Computer Journal (2010) 53(7): 1045-1051 first published online August 19 2009, doi: 10.1093/comjnl/bxp080, Available at: http://comjnl.oxfordjournals.org/content/53/7/1045.full.pdf+html

[6] Moretti, C., Bulosan, J., Thain, D., Flynn, P.J. (2008), "All-pairs: An abstraction for data-intensive cloud computing," *Parallel and Distributed Processing, 2008*, IPDPS 2008. [*IEEE International Symposium*, vol., no., pp.1-11, 14-18 April 2008].

[7] Chou, D. C. and Chou, A. Y. (2011), "Seeking Sustainable Computing: The role of Cloud Computing," *Southwest Decision Sciences Institute Conference*.

[8] Chandrashekar, J. (2009), "The Dark Cloud: Understanding and Defending Against Botnets and Stealthy Malware," *Intel RTechnology Journal*, 13(2).

[9] Ke, A., Yu, Y., Chen, Y., Zhao, E., Xie, Y., Yu, F. and Gillum, Q. (2009), "BotGraph: large scale spamming botnet detection," [*Proceedings of the 6th USENIX symposium on Networked systems design and implementation* (NSDI'09)], USENIX Association, Berkeley, CA, USA, p321-334.

[10] Zeidanloo, H., Shooshtari, M., Amoli, P., Safari, M., and Zamani, M. (2010), "A taxonomy of Botnet detection techniques," Computer Science and Information Technology (ICCSIT), [*2010 3rd IEEE International Conference* v2, p158–162. IEEE].

[11] Murakami, J. (2008), "A hypervisor IPS based on hardware assisted virtualization technology," [*Black Hat USA 2008*].

[12] Rutkowska, J. and Tereshkin, A. (2008), "Bluepilling the Xen Hypervisor," [*Black Hat USA 2008*].

[13] Tsutomu, N., Yoshihiro, O., Hideki, E., Takahiro, S. and Kazuhiko, K. (2010), "Using a Hypervisor to Migrate Running Operating Systems to Secure Virtual Machines," [*2010 IEEE 34th Annual Computer Software and Applications Conference (Proceedings)* p37-46].

[14] Cloud Security Alliance (2010), "Top Threats to Cloud Computing V1.0," Available at: https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf

[15] Arora, P., Wadhawan, R. C., Ahuja, Er. S. P. (2012), "Cloud Computing Security Issues in Infrastructure as a Service," [*International Journal of Advanced Research in Computer Science and Software Engineering*] Vol. 2, Iss. 1, Jan. 2012, ISSN: 2277 128X.

[16] Minas, L., Ellison, B. (2009), "The Problem of Power Consumption in Servers," [*Dr. Dobb's Journal*, May 2009].

[17] Matalon, S., Klein, R., Walls, C. (2011), "Embedded System Power Consumption: A Software or a Hardware Issue?," Mentor Graphics, Available at: http://www.mentor.com/resources/techpubs/upload/mentorpaper_68962.pdf

[18] Warkozek, G., Drayer, E., Debusschere, V., Bacha, S. (2012), "A new approach to model energy consumption of servers in data centers", 2012 IEEE International Conference on Industrial Technology (ICIT), pp.211-216, 19-21 March 2012, doi: 10.1109/ICIT.2012.6209940, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6209940&isnumber=6209899

[19] Junyoung, H., Jiman, H., Yookun, C. (2009), "EARQ: Energy Aware Routing for Real-Time and Reliable Communication in Wireless Industrial Sensor Networks", 2009 IEEE Transactions on Industrial Informatics, vol. 5, no. 1, pp. 3-11, Feb. 2009, doi: 10.1109/TII.2008.2011052, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4799339&isnumber=4799331

A Novel Autonomous Management Distributed System for Cloud Computing Environments

Razvan-Ioan Dinita, George Wilson, Adrian Winckles, Marcian Cirstea, Tim Rowsell
Computing and Technology
Anglia Ruskin University
Cambridge, United Kingdom
{razvan.dinita, george.wilson, adrian.winckles, marcian.cirstea, tim.rowsell}@anglia.ac.uk

Abstract—**This paper describes a novel modular design of an autonomous management distributed system (AMDS) for cloud computing environments and it presents its implementation with the Scala programming language. The AMDS was designed from the ground up with distributed deployment, modularity and security in mind, using a full object oriented approach. A key feature of this system is the ability to gather and store information from various networking and monitoring devices from within the same computing cluster. Another key feature is the ability to intelligently control VMWare vSphere local instances based on analysis of collected data and predefined parameters. vSphere in turn, once it receives commands from the AMDS, proceeds to issue instructions to multiple locally monitored ESXi severs in order to maximize energy efficiency, reduce the carbon footprint and minimize running costs. The predefined parameters are based on results from a previous paper written by the authors. The AMDS has been deployed on the authors' test bed and is currently running successfully. Test results show highly potential industrial applications in datacenter energy management and lowering of operating costs.**

Keywords—cloud; distributed; energy; optimisation; software

I. Introduction

Cloud computing is an emerging technology that devolves computing resources to the Internet [1]. The term "cloud" is commonly used to describe a cluster of computing hardware linked through a series of networking devices. Each computing component has a Hypervisor installed on it. The Hypervisor has been classified into two categories: type 1, which runs directly on the host's hardware, and type 2, which runs within a conventional operating system [2].

This paper focuses on presenting a novel design of an autonomous management distributed system. It continues the initial conceptual work done by the authors [3] and makes use of type 1 Hypervisors, which are capable of running multiple virtualized machines (VM) simultaneously, all controlled by an independent software package. VMWare, through their Academic Program, has supplied the solution currently deployed on the authors' test bed. The package is comprised of two main components: ESXi, a type 1 Hypervisor, and the vSphere Client, a complete cluster management solution.

Also, as presented by [4] energy efficient management of cloud infrastructures is still a very active and current issue in research, more so in the industry where better energy management usually brings lower datacenter operating costs. The authors have already covered this aspect in an earlier paper [3].

The authors have considered two approaches to optimising the energy management within the VMWare cluster. The first one looked at using the features provided by the software solution to set different parameters to achieve the desired results. Unfortunately, there were no relevant parameters available to be set within the vSphere client. The second approach, and the one presented in this paper, was to design a custom distributed software solution to interface with the vSphere client.

The use of distributed systems has its roots in operating system architectures that were initially studied in the 1960s [5]. The first widespread distributed systems were invented in the 1970s [6] and implemented on an Ethernet infrastructure. [7] also presents a well designed modular system of a SOA based architecture (also invented in the 1960s) for use in Cloud Computing environments. The authors have based their software solution design on the above presented architectures due to their tried and tested reliability for over 40 years.

II. Design of the AMDS

For the benefit of the reader Fig. 1 presents an overview of the AMDS's position within a Cloud Computing environment. The AMDS connects to the four most important components of the cloud system: access point (connection to the outside world), power reading hardware (monitors power consumption), network reading hardware (monitors network flow), and the heart of the cloud system – the proprietary software (VMWare ESXi) that makes decisions regarding the server and storage management.

A. Design Goals

The AMDS was designed from the ground up with three main goals in mind: 1) security, 2) modularity, and 3) parallel processing.

To address security the authors have based their design on work done by [8] in order to introduce an appropriate authentication component. As such, each system component is required to authenticate each time it interacts with any other component.

Modularity has been achieved through implementing sound Object Oriented programming principles. Some core principles relevant to this work outlined by [9] have been applied to the design. Every task the system is required to accomplish is split down into independent, fully reusable modules.

The third goal, parallel processing, is achieved through the deployment of several instances of the AMDS across the test bed. The system has been designed to be deployed on a VM and, through its configuration parameters, locates other system instances in order to establish a live communication link. From thereon, the linked instances coordinate each other's tasks in order to ease the hardware load and maximise efficiency, making the AMDS highly scalable. The system's distributed design takes into account key points presented by the authors of [10].

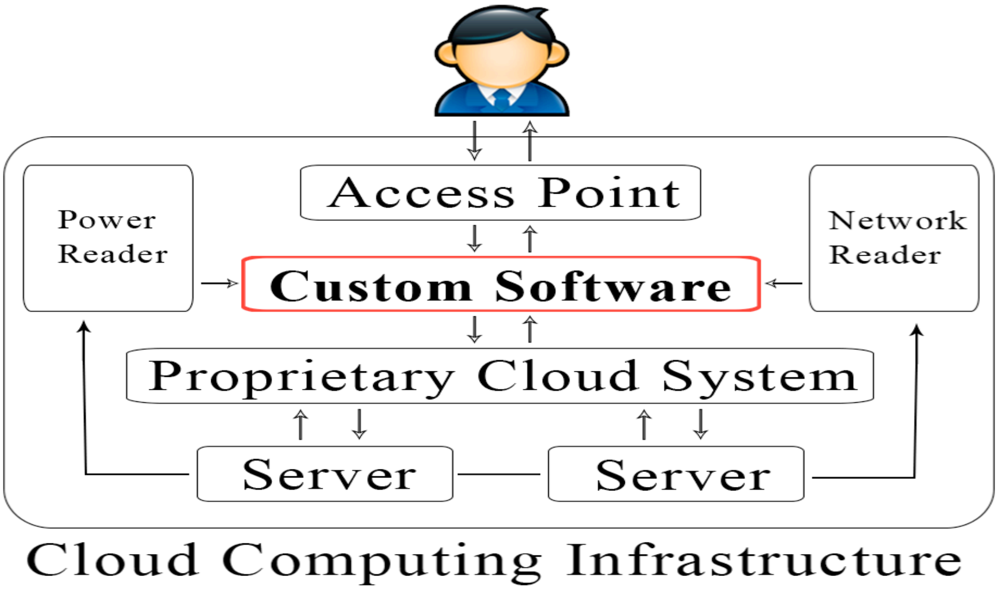


Fig. 1. Overview diagram of the AMDS's position within a Cloud Computing environment

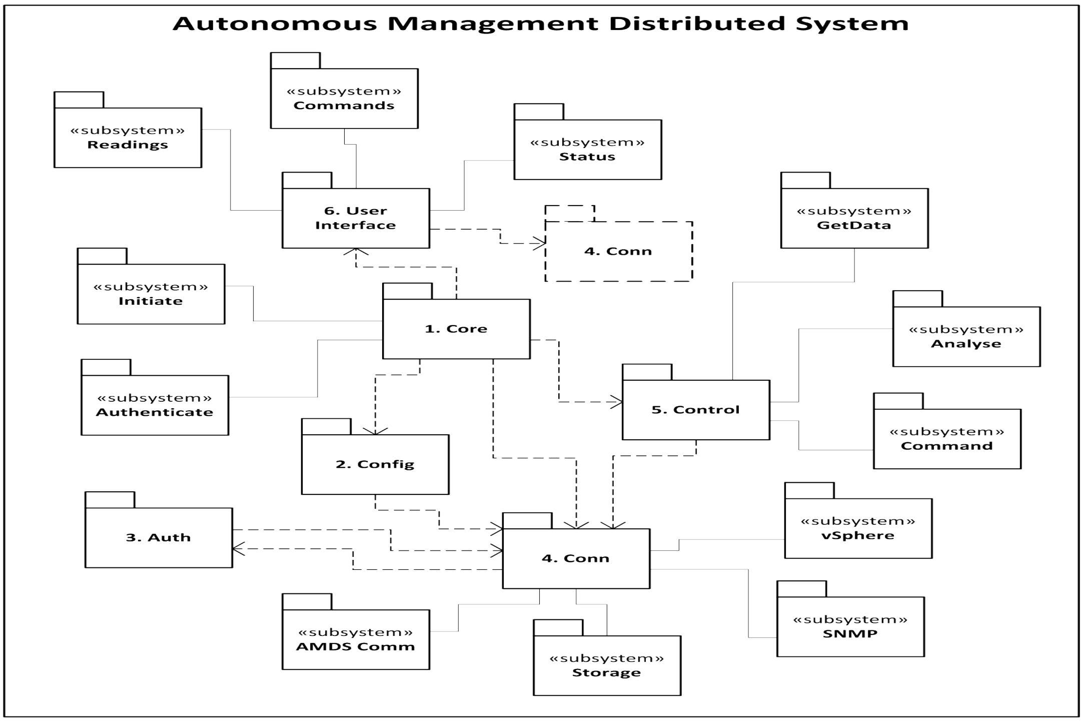


Fig. 2. UML Design Diagram of the AMD

B. Design Breakdown

Fig. 2 presents a UML [11] diagram of the AMDS design. At the time of writing it is composed of 19 different parts, each designed for a specific task and fully reusable. The main features of these component parts will now be described:

1) Core: Main system module. The entry point for the AMDS. From there the system accesses the configuration parameters (Fig. 2, 2. Config) and starts its internal tasks (Fig. 2, 5. Control) and the User Interface (Fig. 2, 6. User Interface). It is responsible for facilitating communication between the different modules attached to it.

a) Initiate: Module entry point. Achieves most of the module functionality. Initialises modules and establishes connections between them.

b) Authenticate: Helper module. Undertakes the initial system authentication. This helps with detection of possible hijack attempts by making sure a current instance remains valid and genuine.

2) Config: Helper module. Responsible for maintaining and providing access to the system configuration parameters. It interacts with the connection module (Fig. 2, *4. Conn*) in order to gain access to the Storage component (Fig. 2, *4. Conn :: Storage*), Config database. It is active throughout the lifespan of the system instance, facilitating on-the-fly parameter alteration.

3) Auth: Key system module. Manages task and connection authentication. It performs checks against the initial determined instance validity and genuineness in an attempt to discover potential system hijack attempts and prevent them. It locks down any connection or task that does not pass the validation step and makes a log entry with relevant details on the security issue.

4) Conn: Main system module. Facilitates all system connections between the modules themselves or between the modules and the computing cluster. It is the main access route to specialised mini-modules as well as attempt to authenticate each connection passing through it by calling upon the Auth module (Fig. 2, *3. Auth*).

a) AMDS Comm: Critical mini-module. Manages communication between instances of the AMDS including the passing of data between them. Acts as a load balancer by creating a bridge between current instance and one other instance. On each connection attempt it calls upon the Auth module to verify the integrity of the outside instance before allowing any kind of information exchange.

b) Storage: Main mini-module. Keeps track of internal databases for each system module that deals with data. It stores information for the Config and Control modules.

c) SNMP: Specialised mini-module. Facilitates passing of information between current system instance and devices that understand the Simple Networking Management Protocol (SNMP). This protocol allows data to pass both ways, making it possible to issue commands and receive results between different devices that use it.

d) vSphere: Critical specialised mini-module. Interfaces the VMWare vSphere client to allow issuing commands and retrieving results. This module bridges the gap between the custom designed AMDS and the proprietary software solution provided by VMWare.

5) Control: Main module. Initiates data collection, storing and analysis tasks, as well as initiate commands to the vSphere Client through the Conn module (Fig. 2, *4. Conn :: vSphere*). This allows for data to be collected from monitoring devices across the computing cluster, stored, analysed and actions to be taken based on the results and the configuration parameters.

a) GetData: Main mini-module. Deals with raw data retrieval. It initiates connections to the SNMP mini-module (Fig. 2, *4. Conn :: SNMP*)., retrieves and stores collected information using the Storage mini-module (Fig. 2, *4. Conn :: Storage*), Raw Data database.

b) Analyse: Main mini-module. Retrieves chunks of raw data from the Storage mini-module (Fig. 2, *4. Conn :: Storage*), Raw Data database, and come up with data capable of being compared to the configuration parameters. It then stores the analysis results using the same mini-module, but in the Results database.

c) Command: Main mini-module. Compares analysis results with the configuration parameters and make intelligent decisions which maximise energy efficiency. After it stores issued commands in the Commands database, it then proceeds to send them to the correct interfacing mini-module from within the Conn module (Fig. 2, *4. Conn*).

6) User Interface: Noncritical system module. Facilitates system monitoring by presenting information stored on the system in human readable form.

a) Status: Main mini-module. Provides an overview of the current system state as well as global statistics, including access to security logs and top level information on database disk usage.

b) Readings: Main mini-module. Provides an in-depth view of each individual database currently utilised by the system instance. All databases maintained by the Conn module (Fig. 2, *4. Conn*) are included. It makes use of data filtering and table display.

c) Commands: Main mini-module. Provides an in-depth view of all command decisions the urrent system instance has taken as well as the accompanying results received from all the commanded systems.

III. Implementation

A. Language Considerations

In the implementation stage the authors considered many programming languages capable of initiating remote connections, including Ruby, PHP, Java, Scala, C++, C#. The main criteria to be considered was portability i.e. make the system so that it can be deployed on as many different operating systems as possible. Only two of the considered languages met the required criteria: Java and Scala.

Java is an established programming language making its first appearance in 1995[1]. It is able to function on any operating system running the Java Virtual Machine (JavaVM). All major systems, including Unix, Linux and Windows are currently capable of running the JavaVM. However whilst the language functionality continues to evolve it does not handle running multithreaded tasks well.

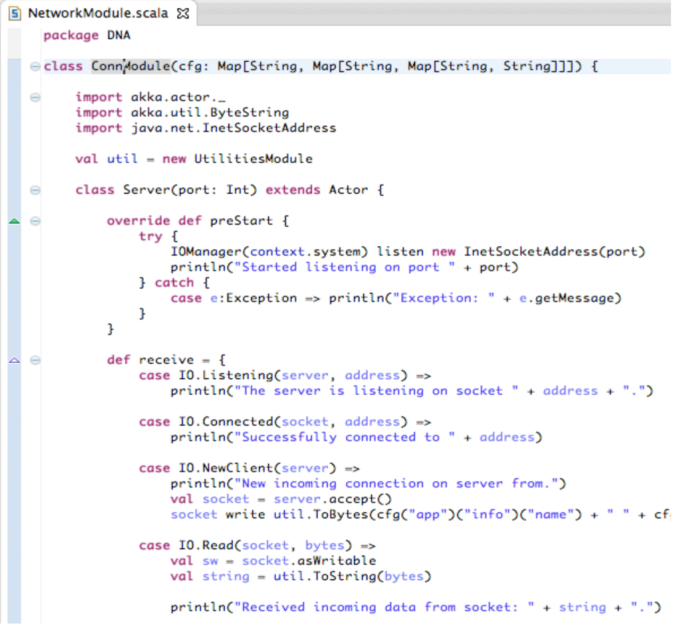
Scala [12] is a relatively new language. In spite of only making its first appearance in 2003, it has grown in popularity very quickly due to its multithreading capabilities as well as its concise way of expressing common programming patterns[2]. This has helped to drastically reduce development time on projects. One major advantage and the main reason why it has gained popularity so quickly is its seamless integration with Java. Scala support in Java for example can be provided by importing an appropriate library, and all Scala programs also run on the JavaVM which means that these can be deployed on all the major systems.

B. Implementation Process

Scala has a unique way of dealing with data structures – there are mutable (can be changed – e.g. *var*) and immutable (cannot be changed – e.g. *val*) variable types. Scala creators recommend using the immutable types because this minimises the risk of random or unintentional data corruption during runtime.

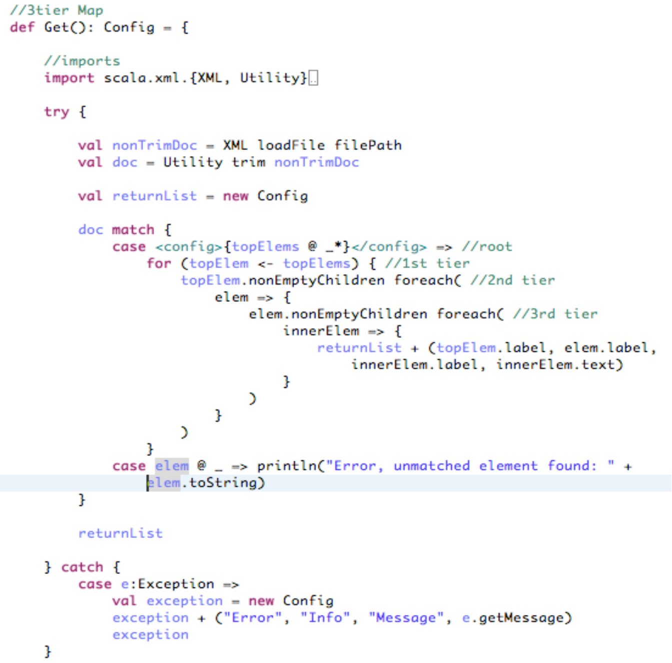
Scala also allows for almost out-of-the-box distributed code implementations through the use of Actors. Scala Actors are capable of independent and asynchronous operation, operating by passing messages from one to another. They function under a command hierarchy and also allow for quick error recovery. Since each actor operates independently of each other, if one encounters a fatal error, the message is cascaded up the chain of command until it reaches an actor programmed to handle that type of issue. It can then proceed to take further actions as necessary e.g. restart the failed actor.

In the development process the authors have used the Eclipse Integrated Development Environment (IDE) to assist with code completions and debugging as necessary. Each module has been implemented using inheritance based Object Oriented programming principles. Fig. 3 shows a few lines of Scala code (part of the Conn module implementation). Java libraries have been used to facilitate remote connections and Actors have been used to operate as message transporters. The code in Fig. 3 is set to receive remote connections and take different actions based on the type of result.



```
package DNA

class ConnModule(cfg: Map[String, Map[String, Map[String, String]]]) {

    import akka.actor._
    import akka.util.ByteString
    import java.net.InetSocketAddress

    val util = new UtilitiesModule

    class Server(port: Int) extends Actor {

        override def preStart {
            try {
                IOManager(context.system) listen new InetSocketAddress(port)
                println("Started listening on port " + port)
            } catch {
                case e:Exception => println("Exception: " + e.getMessage)
            }
        }

        def receive = {
            case IO.Listening(server, address) =>
                println("The server is listening on socket " + address + ".")

            case IO.Connected(socket, address) =>
                println("Successfully connected to " + address)

            case IO.NewClient(server) =>
                println("New incoming connection on server from.")
                val socket = server.accept()
                socket write util.ToBytes(cfg("app")("info")("name") + " " + cf

            case IO.Read(socket, bytes) =>
                val sw = socket.asWritable
                val string = util.ToString(bytes)

                println("Received incoming data from socket: " + string + ".")
```

Fig. 3. Conn Module implementation, screenshot taken from Eclipse IDE



```
//3tier Map
def Get(): Config = {

    //imports
    import scala.xml.{XML, Utility}□

    try {

        val nonTrimDoc = XML loadFile filePath
        val doc = Utility trim nonTrimDoc

        val returnList = new Config

        doc match {
            case <config>{topElems @ _*}</config> => //root
                for (topElem <- topElems) { //1st tier
                    topElem.nonEmptyChildren foreach( //2nd tier
                        elem => {
                            elem.nonEmptyChildren foreach( //3rd tier
                                innerElem => {
                                    returnList + (topElem.label, elem.label,
                                        innerElem.label, innerElem.text)
                                }
                            )
                        }
                    )
                }
            case elem @ _ => println("Error, unmatched element found: " +
                elem.toString)
        }

        returnList

    } catch {
        case e:Exception =>
            val exception = new Config
            exception + ("Error", "Info", "Message", e.getMessage)
            exception
    }
```

Fig. 4. Part of the Storage Module, Config database, screenshot taken from Eclipse IDE

The databases have been implemented using the eXtensible Markup Language (XML). Fig. 4 shows part of the Storage module, specifically part of the method that deals with parsing the Config database information. The code uses Scala libraries for dealing with XML data structures and builds up a Config type object (first line, after the colon, signifies the return type). Exception handling is implemented in every module as a safety measure to prevent unexpected program termination due to unexpected or corrupt data.

[1] http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html

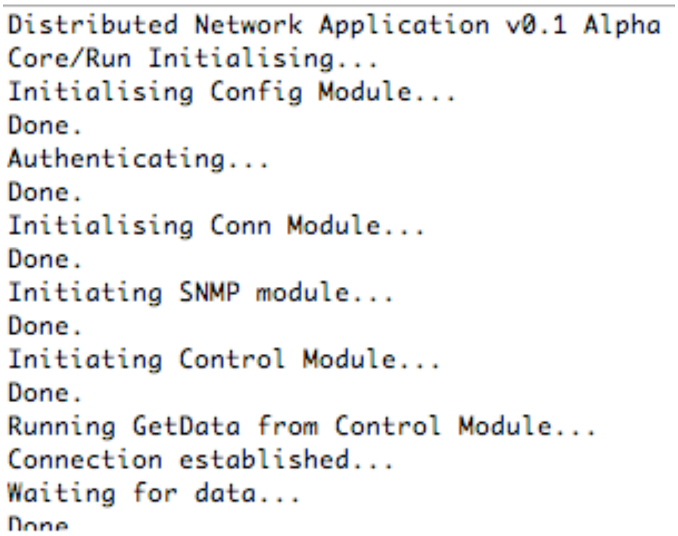
[2] http://www.scala-lang.org/node/25

Another important component of the ASDM is the vSphere mini-module. In its development the authors have made heavy use of the vSphere SDK for Java [13]. This set of Application Programmable Interfaces (APIs) facilitates message exchanges between the Control module and the vSphere Clients currently in the computing cluster. Still in its preview stages, it allows for much interaction between third party programs and the vSphere client itself.

IV. Results

The ASDM is currently deployed and running on the authors' test bed. An Ubuntu Linux based Virtual Machine was chosen to run the application due to the high reliability of the Operating System. The VM has is connected to the ESXi Servers via a closed Virtual Network. This ensures seamless connectivity between virtual and physical hardware, thus allowing the AMDS to receive information from the ILOs and the vSphere Client and to send commands back.

At the moment, data is being collected and analysed for debugging purposes. As seen in Fig. 5, the program is running successfully. The core initializes and authenticates as expected; Conn, SNMP and Control modules are being activated, and raw data starts to be collected for storing and analysis.



```
Distributed Network Application v0.1 Alpha
Core/Run Initialising...
Initialising Config Module...
Done.
Authenticating...
Done.
Initialising Conn Module...
Done.
Initiating SNMP module...
Done.
Initiating Control Module...
Done.
Running GetData from Control Module...
Connection established...
Waiting for data...
Done
```

Fig. 5. Output from ASDM, console view

Since the AMDS has been deployed it has produced a great number of data stored within several databases. The data comes from queries performed by the AMDS on the different networking hardware operating within the cloud environment (Switches, Routers, ILOs, ESXi). The authors have merged and analysed all generated data, the results of which have been expressed in Table I and Fig. 6.

The system efficiency was calculated by using the formulas (1) and (2). In formula (1) P_{rise} is the power consumption rise percentage calculated by dividing $P_{current}$ (server power consumption at any other time – processor load > 0%) by P_{idle} (server power consumption when idle – 0% processor load). In formula (2) E (server operating efficiency) is calculated by dividing L (server load) by P_{rise}.

$$P_{rise} = P_{current} / P_{idle} \qquad (1)$$

$$E = L / P_{rise} \qquad (2)$$

TABLE I. Comparison between normal system operation (without AMDS) and optimized system operation (with AMDS)

	Before AMDS	During AMDS operation
Processor Load (%)	25	100
Power Consumption (Watts / Hour)	168	239
Power Consumption Rise (Watts / Hour)	35.5	92
Efficiency (%)	70	108

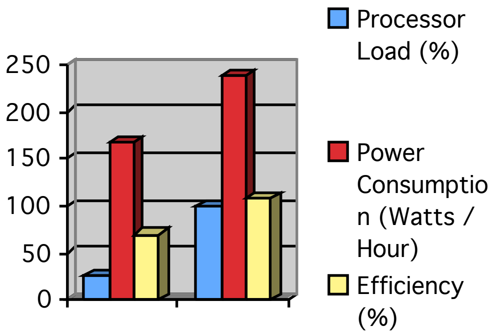


Fig. 6. Comparison between normal system operation (without AMDS) and optimized system operation (with AMDS)

In the second column of Table I the data has been recorded before the AMDS has been enabled (decision making module was disabled). The system efficiency stabilized at 70% due to the fact that several servers operating at 25% of their potential hardware load. The server power consumption at this stage was 168 Watts / Hour, an increase of 35.5% from system idle state.

In the third column of Table I the data has been recorded after the AMDS has been enabled (decision making module was enabled). Almost immediately all system traffic had been redirected towards one of the active servers while the others had been shut down to conserve power, thus bringing the system efficiency up to 108%. Power consumption at this stage was 239 Watts / Hour, an increase of 92% from system idle state.

The results presented above demonstrate how the AMDS is capable of minimising the cloud system power consumption by up to 8%, thus generating an important operating cost reduction.

V. Conclusions

The development of ASDM is an ongoing process. The authors are constantly tweaking and changing the code in pursuit of better performance. The proposed system has several key industrial applications:

1) Green Datacetre. The proposed system is capable of reducing overall energy consumption by intelligently turning physical servers on and off based on data collection from throughout the computing cluster.

2) Lower Datacentre operating costs. This is a direct consequence of the previous statement. Overall lower energy consumption leads to reduced operating costs. This in turn allows for higher profits and more investments to be made.

3) Set-and-forget scenarios. The AMDS, due to its autonomous nature and modular design, is capable of on-the-fly self reconfiguration based on analysis results of gathered data. This flexibility makes it ideal for set-and-forget situations as well as low cost maintenance schedules.

Next, the authors have started looking at intrusion detection in an attempt to develop a successful method of hijack detection. Data collected from running ASDM, including the Databases and Logs, is collected at the same time as an attempt at hijacking the system is in progress. All data is continuously monitored and analysed for signs of the hijack process on the system.

References

[1] Mirashe, S. P.; Kalyankar, N. V.; (2010), “Cloud Computing”, *Communications of the ACM*, 51 (7), 9.

[2] Popek, G. J.; Goldberg, R. P.; (1974). "Formal Requirements for Virtualizable Third Generation Architectures", *Communications of the ACM,* 17 (7), pp. 412 –421, doi:10.1145/361011.361073.

[3] Dinita, R. I.; Wilson, G.; Winckles, A.; Cirstea, M.; Jones, A., “Hardware Loads and Power Consumption in Cloud Computing Environments”, *ICIT 2013 – 2013 IEEE International Conference on Industrial Technology*, pp. 1291-1296, 25-27 Feb. 2013, ISBN: 978-1-4673-4568-2.

[4] Mora, D.; Taisch, M.; Colombo, A. W., “Towards an energy management system of systems: An industrial case study,” *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, pp. 5811-5816, 25-28 Oct. 2012.

[5] Andrews, G. R. (2000), “Foundations of Multithreaded, Parallel, and Distributed Programming”, p. 348, Published by Addison–Wesley, ISBN 0-201-35752-6.

[6] Andrews, G. R. (2000), “Foundations of Multithreaded, Parallel, and Distributed Programming”, p. 32, Published by Addison–Wesley, ISBN 0-201-35752-6.

[7] Karnouskos, S.; Colombo, A.W.; Bangemann, T.; Manninen, K.; Camp, R.; Tilly, M.; Stluka, P.; Jammes, F.; Delsing, J.; Eliasson, J., “A SOA-based architecture for empowering future collaborative cloud-based industrial automation,” *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, pp.5766-5772, 25-28 Oct. 2012.

[8] Irani, G.N.H.; Tawosi, V., “AAMA: A new Authentication and Authorization architecture for modular information systems, a robust object oriented approach,” *Application of Information and Communication Technologies (AICT), 2011 5th International Conference on*, pp. 1-5, 12-14 Oct. 2011.

[9] Caragiozidis, M.; Mouratidis, N.; Kavadias, C.; Loupis, M.; Berger, M., “Design Methodology for a Modular Component Based Software Architecture,” *Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International*, pp. 1122-1127, July 28 2008 - Aug. 1 2008.

[10] Maffei, A.; Hofmann, A., "From flexibility to true Evolvability: An introduction to the basic requirements," *Industrial Electronics (ISIE), 2010 IEEE International Symposium on* , vol., no., pp.2658,2663, 4-7 July 2010.

[11] Unified Modeling Language, http://www.uml.org.

[12] Scala Programming Language, http://www.scala-lang.org.

[13] VMware vSphere™ SDK for Java, http://communities.vmware.com/community/vmtn/developer/forums/java_toolkit

Use of NetFlow/IPFIX Botnet Detection Tools to Determine Placement for Autonomous VMs

Razvan-Ioan Dinita, razvan.dinita@anglia.ac.uk
Adrian Winckles, adrian.winckles@anglia.ac.uk
George Wilson, george.wilson@anglia.ac.uk
Anglia Ruskin University, Cambridge, UK

ABSTRACT

This paper describes a novel method of autonomously detecting malicious Botnet behaviour within a Cloud datacentre, while at the same time managing Virtual Machine (VM) placement in accordance to its findings, and it presents its implementation with the Scala programming language.

A key feature of this method, using output from NetFlow/IPFIX, both of which are capable of producing detailed network traffic logs, is its capability of detecting unusual Client behaviour through the analysis of individual data packet information. It has been implemented as a module of an Autonomous Management Distributed System (AMDS) presented in [Dinita, R. I. et al., 2013], giving it direct access to all the VMs and Hypervisors on the Cloud network.

Another key feature is that it can have an immediate and effective impact on network security in a Botnet attack context by issuing lockout commands to every networked VM through the AMDS. It possesses the ability to intelligently control VMWare vSphere local instances based on analysis of collected data and predefined parameters. vSphere in turn, once it receives commands from the AMDS, proceeds to issue instructions to multiple locally monitored ESXi severs in order to ensure continuous security.

A proof of concept has been developed and is currently running successfully on the authors' test bed.

Keywords: botnet, software, security, detection, autonomous

1. INTRODUCTION

This paper sets out to describe one potential design and implementation of a Botnet Detection software module. This module is aimed at analysing portions of live network traffic in an attempt to detect unauthorised and malicious activity within a Cloud Computing infrastructure. The work presented is building upon previous research undertaken by the authors into autonomous datacentre management and supervision software by bolting the aforementioned botnet module onto the Autonomous Management Distributed System (AMDS) [1].

The botnet detection algorithm used is based on the access pattern-based detection method [2], which looks at client behaviour to differentiate a legitimate client from an infected one. This is based on the assumption that a compromised client will, in most cases, operate in the same manner as the bot that originally infected it. All network data is compared against existing, legitimate client profiles, continuously refined over time, in an attempt to spot and block the compromised ones.

It also makes use of an anomaly-based heuristic algorithm based on the work carried out by [3]. The algorithm comprises of two main detection components: an Internet Relay Chat (IRC) mesh, and a Transmission Control Protocol (TCP) scan detection heuristic. The author has made use only of the TCP heuristic component and adapted it as necessary into the AMDS botnet detection module.

1.1. Botnets Background

The term *bot* is short for *robot*. Criminals distribute malicious software (also known as malware) that can turn your computer into a bot (also known as a zombie). When this occurs, your computer can perform automated tasks over the Internet, without you knowing it. Bots are typically used to infect large numbers of computers. These computers form a network, or a botnet, and are used to send out spam email messages, spread viruses, attack computers and servers, and commit other kinds of crime and fraud.

Peer-2-Peer bots are split into two categories: Masters and Slaves. Each of these uses secure channels to pass information between one another containing a command and details on the originating control module. This allows the Bot Masters to issue commands to the Bot Slaves. Bot masters will use root-kits and anti-VM static- DLL/binary code [4] along with secure channels in order to avoid detection. Once the control channel is established, it is used for communications that employ two different channel operation architectures, the *centralised architecture* and the *decentralised architecture*.

The *centralised architecture* has been used in the past by IRC-based (Internet Relay Chat, communication protocol) botnets, which used IRC servers to issue commands to all malware-infected machines. This mode has many different variations [5] e.g. the final destination could contain a text document with a list of static IP addresses and lists of URLs, so that flexible IP addresses can utilised. The *decentralised architecture*, on the other hand, is a newer communication architecture that enables infected hosts to exchange information via distributed networks such as Peer-2-Peer. This method may reduce the rate of firewall and antivirus detection [5].

Regardless of the type of architecture, there are two types of command and control channels, the *Persistent Channel* and the *Periodic Channel*. The *Persistent Channel* maintains a direct connection with the Bot Master. This connection type is normally employed by IRC bots, however, it is increasingly becoming obsolete due to periodic channels. The *Periodic Channel*, on the other hand, connects to the Bot Master periodically in order to avoid detection. Typically, the destination has had no prior communication with the host. This is normally used to throw network security devices and probes off track.

1.2. AMDS / Botnet Detection Module Network Logical Layout

The Botnet Detection module is a critical part of the AMDS [1] in the context of malicious behaviour detection. It is charged with storing and analysing output network data flows, as produced by the NetFlow/IPFIX setup, in an attempt to internally construct and continuously iterate on a generic botnet detection model. It hopes to achieves this through always going back and forth between new, unidentified flows and past, already analysed flows and comparing specific elements from each with aim to brand the new flows as either healthy of infected.

As it can be seen in *Figure 1* below, the AMDS is deployed on a Virtual Machine Instance (VMI) on the authors' Cloud Test Bed, which has been created based on a VM Template through the VMWare vSphere Client and connected to the internal Cloud Network.

At a logical overview level, the Cloud Infrastructure is composed of a Cisco ASA Router (Adaptive Security Appliance), three VLANs (Virtual Local Area Network), the vSphere Client, the ESXi Server, a group of ILOs (Integrated Lights Out), and the AMDS. Each of these components is underpinned by a series of physical network cables, switches, and routers that facilitate interconnectivity between them. The ESXi Server is comprised of multiple independent VMs interconnected by the three VLANs.

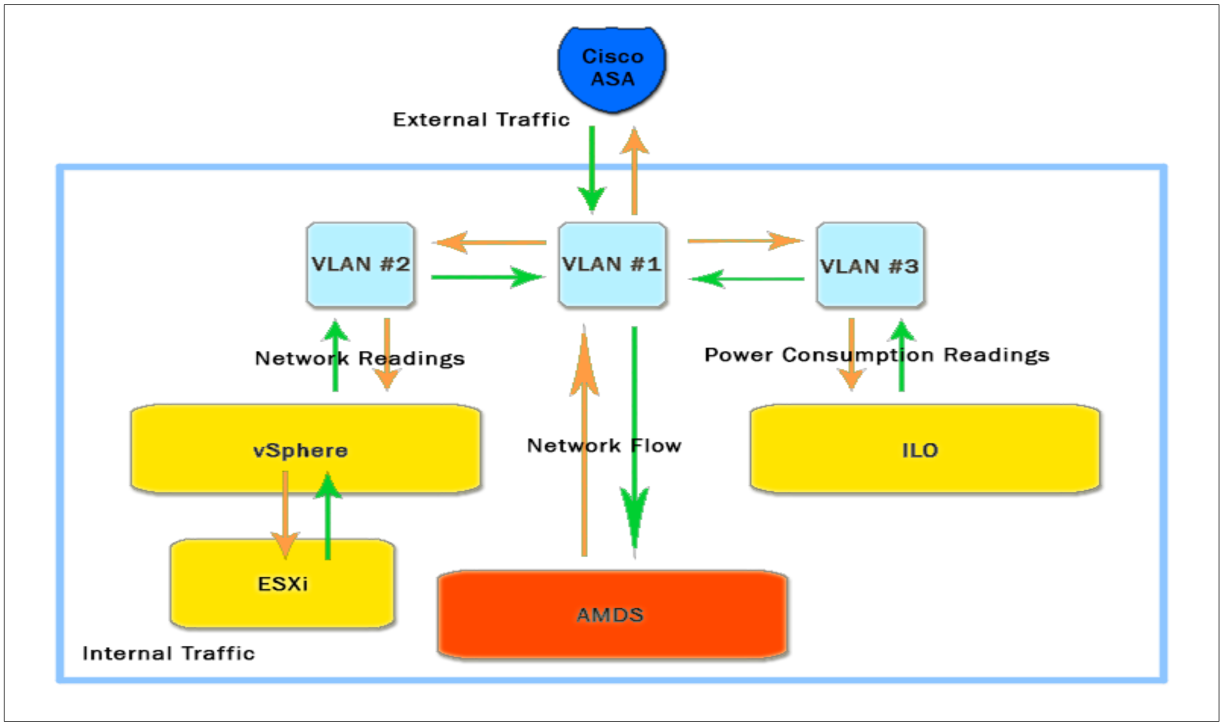


Figure 1 – AMDS/Botnet Module Network Logical Layout

The network operational flow is expressed through two different arrow colours in *Figure 1*, both relevant to the AMDS: *GREEN* reflects network traffic flowing towards the AMDS, while *ORANGE* reflects network traffic flowing from the AMDS towards all other infrastructure components.

Green traffic is composed of data the AMDS requires when performing the infrastructure logical analysis from the points of view of processor loads, power consumption, and network flow, from the following sources:

I. Processor load data is retrieved from the vSphere Client;

II. Power consumption data is retrieved from the ILO group; and
III. Network flow data is obtained from the various switches and routers spread across the infrastructure.

Orange traffic is composed of commands the AMDS issues post-analysis. This includes:

I. VM moving commands to the ESXi Servers through vSphere;
II. Physical server shut-down / start-up commands to various ESXi Servers through vSphere;
III. Network flow restriction requests to the Cisco ASA router; and
IV. Load balancing requests across the infrastructure to other AMDS running instances.

Even though all infrastructure components are located within the same physical network, the VLANs allow splitting it up into smaller logical groups which can be more easily maintained and controlled. Each VLAN is created and maintained by the vSphere Client and only has direct access to the logical component in its immediate vicinity. This allows for the formation of highly compartmented and self-contained/-managed logical groups.

A direct consequence of the information presented above, the Botnet Detection module, which is an inherent part of the AMDS, has direct access to all VLANs. This makes it capable of interfering with regular network data flow based on its post-analysis results. This gives it the power to control every aspect of a physical infrastructure through controlling its logical groups. Once AMDS collects several hours worth of data, it is then capable of issuing commands to vSphere, which in turn will forward these, as needed, to other Components under its control.

2. NETFLOW / IPFIX BACKGROUND

2.1. NetFlow Based Traffic Monitoring

NetFlow has been developed in-house initially to provide improved packet switching capabilities to some of their network devices in 1990. It then has steadily grown into a complex network operational analysis[1] tool.

NetFlow is capable of providing maximum network awareness, and, if used as part of a complex cloud infrastructure, it is able of giving insight into the different types of data packets flowing through the network at any given time. It makes use of 5 to 7 IP packet attributes to generate traffic flow reports, which can be later on analysed by system administrators.

NetFlow utilises the following information in its traffic flow caches: packet size, IP addresses and ports for both packet source and destination, class of service, device interface, and protocol type. In addition, it also records flow timestamps, next hop IP address, subnet mask, and TCP flags. All of these flow parameters aim to provide a holistic network view used in many different scenarios, of which the one of interest is detecting malicious behaviour in a cloud network infrastructure.

As it can be seen in *Figure 2*, upon enabling a device to utilise NetFlow, it then proceeds with recording all traffic coming into the network. These recordings, after undergoing NetFlow processing, are stored in a database entitled NetFlow Cache in the form of a table, each row containing one data flow cache. These flows have a shorter or longer lifetime as determined by the flow timestamp, depending on whether traffic has stopped flowing, a FIN signal has been received indicating the end of the flow, or they have exceeded their pre-set lifetime.

The *NetFlow Enabled Device* (*Figure 2*) is, in this case, the NetFlow Exporter. Its primary concern is capturing traffic data and transforming it into flows. This data is then transferred over to a *NetFlow*

[1] http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_
paper0900aecd80406232.html

Collector Database Flow Cache (*Figure 3*) where it is analysed and converted into meaningful reports that present an overview of the processed network data traffic.

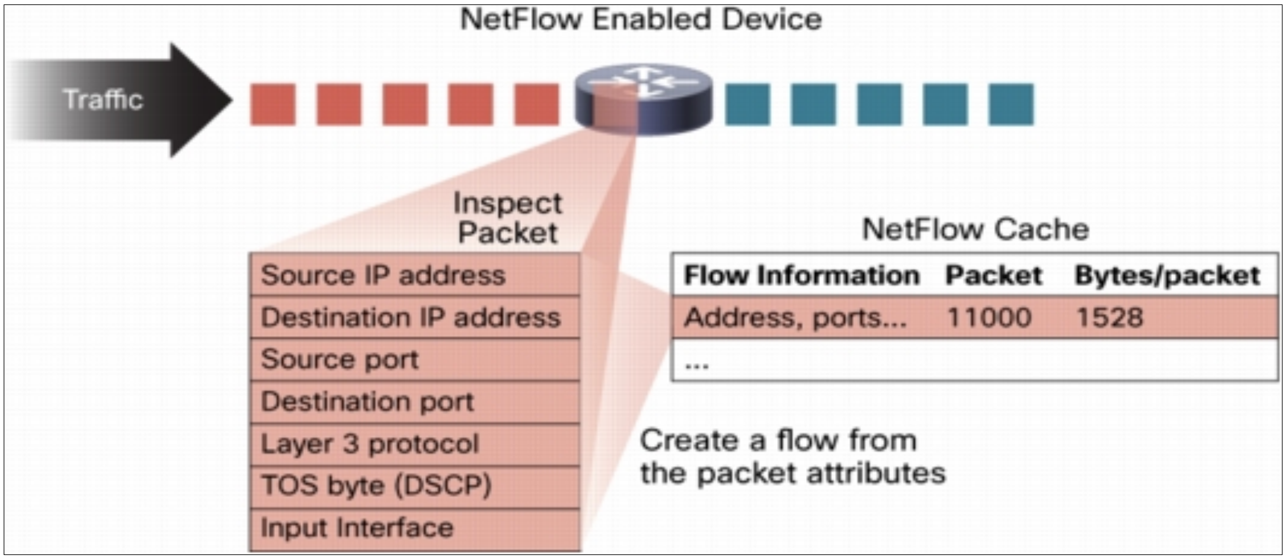
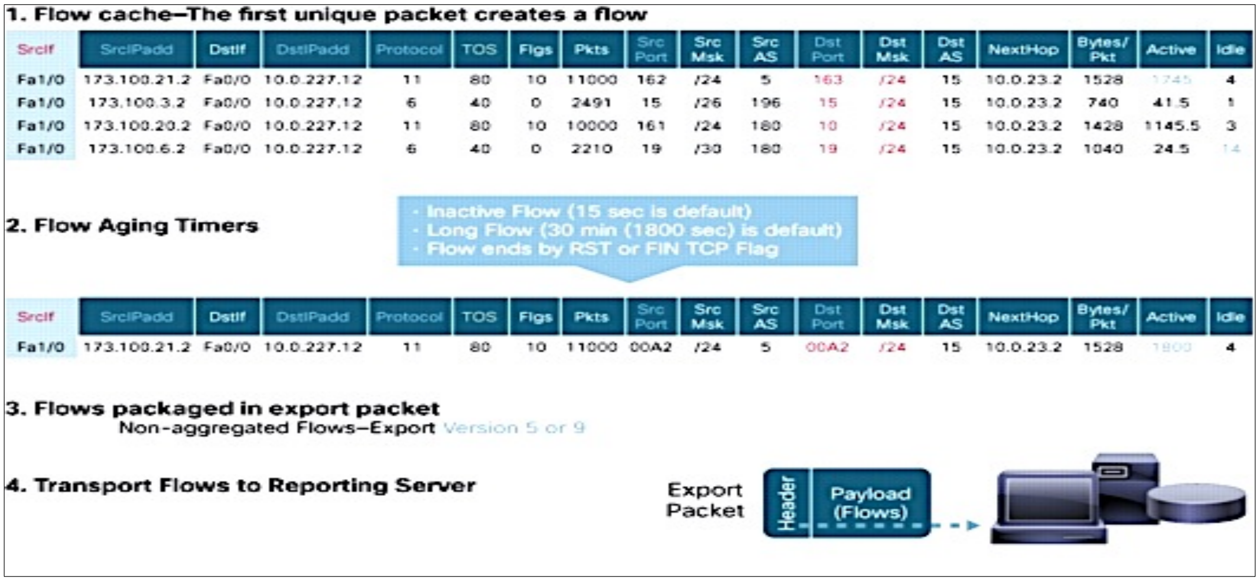


Figure 2 - NetFlow Flow Cache Generation[2]



1. Flow cache–The first unique packet creates a flow

SrcIf	SrcIPadd	DstIf	DstIPadd	Protocol	TOS	Flgs	Pkts	Src Port	Src Msk	Src AS	Dst Port	Dst Msk	Dst AS	NextHop	Bytes/ Pkt	Active	Idle
Fa1/0	173.100.21.2	Fa0/0	10.0.227.12	11	80	10	11000	162	/24	5	163	/24	15	10.0.23.2	1528	1745	4
Fa1/0	173.100.3.2	Fa0/0	10.0.227.12	6	40	0	2491	15	/26	196	15	/24	15	10.0.23.2	740	41.5	1
Fa1/0	173.100.20.2	Fa0/0	10.0.227.12	11	80	10	10000	161	/24	180	10	/24	15	10.0.23.2	1428	1145.5	3
Fa1/0	173.100.6.2	Fa0/0	10.0.227.12	6	40	0	2210	19	/30	180	19	/24	15	10.0.23.2	1040	24.5	14

2. Flow Aging Timers

- Inactive Flow (15 sec is default)
- Long Flow (30 min (1800 sec) is default)
- Flow ends by RST or FIN TCP Flag

SrcIf	SrcIPadd	DstIf	DstIPadd	Protocol	TOS	Flgs	Pkts	Src Port	Src Msk	Src AS	Dst Port	Dst Msk	Dst AS	NextHop	Bytes/ Pkt	Active	Idle
Fa1/0	173.100.21.2	Fa0/0	10.0.227.12	11	80	10	11000	00A2	/24	5	00A2	/24	15	10.0.23.2	1528	1800	4

3. Flows packaged in export packet
Non-aggregated Flows–Export Version 5 or 9

4. Transport Flows to Reporting Server

Figure 3 - Example NetFlow Cache[3]

[2] http://www.cisco.com/c/dam/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_ white_paper0900aecd80406232.doc/_jcr_content/renditions/prod_white_paper0900aecd80406232-1.jpg

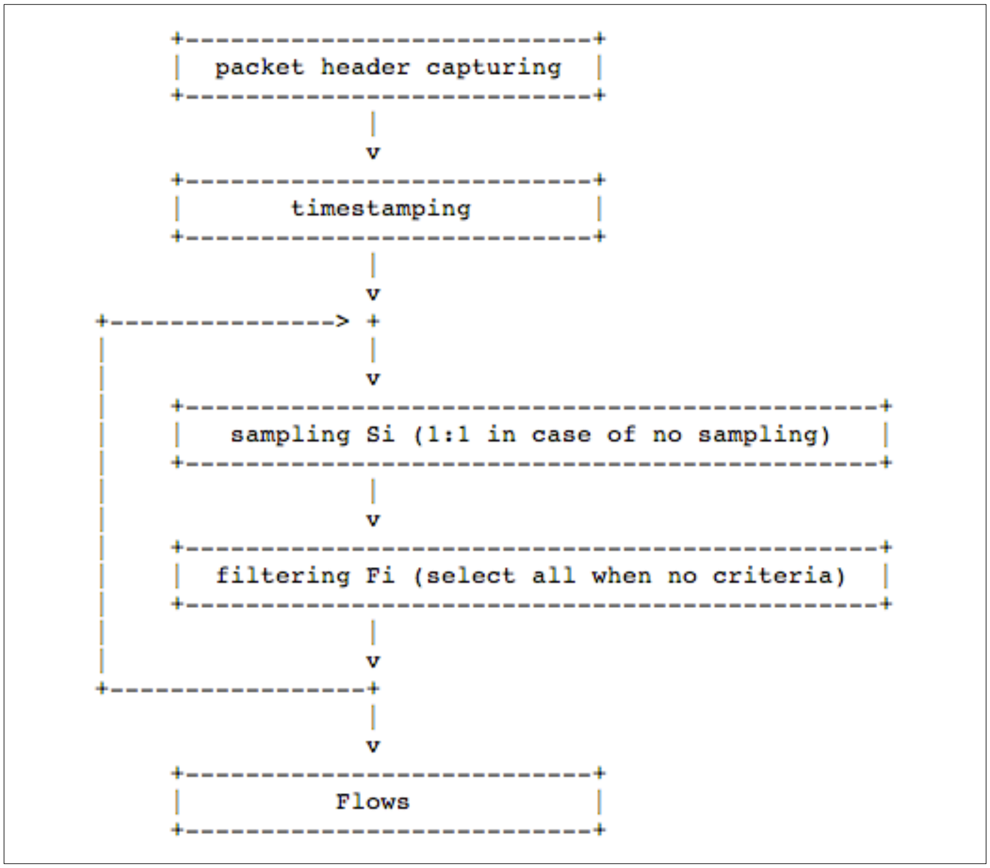
[3] http://www.cisco.com/c/dam/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_ white_paper0900aecd80406232.doc/_jcr_content/renditions/prod_white_paper0900aecd80406232-2.jpg

As it can be seen in *Figure 3*, a typical NetFlow Cache database table contains many different columns generated based on parameters both taken from the packet itself and created by NetFlow in its flow generation. It keeps track of both new and old flows by separating them into two different tables: one for active flows, another for ended flows (either because they had exceeded their pre-set lifetime or the traffic had stopped).

2.2. IPFIX Based Traffic Monitoring

On the other hand, the IP Flow Information Export (IPFIX) format is an Internet Engineering Task Force (IETF) protocol designed to be a universally accepted standard for capturing traffic flow through network devices. IPFIX is responsible for defining the format of captured flows as well as detailing the flow method of transfer between the exporter and collector.

IPFIX, similarly to NetFlow, is capable of breaking down data packets within the observed network traffic according to their attributes. As such, the flow metering process offers several configuration possibilities, which can be used to narrow down the packets included in the flow. There can be defined different sampling and filtering functions, each of which is dealing with a certain aspect of data selection.



```
          +---------------------------+
          |  packet header capturing  |
          +---------------------------+
                       |
                       v
          +---------------------------+
          |        timestamping       |
          +---------------------------+
                       |
                       v
     +---------------> +
     |                 |
     |                 v
     |    +-------------------------------------------+
     |    |  sampling Si (1:1 in case of no sampling)  |
     |    +-------------------------------------------+
     |                 |
     |                 v
     |    +-------------------------------------------+
     |    | filtering Fi (select all when no criteria) |
     |    +-------------------------------------------+
     |                 |
     |                 v
     +-----------------+
                       |
                       v
          +---------------------------+
          |           Flows           |
          +---------------------------+
```

Figure 4 - IPFIX Packet Selection Criteria[4]

As it can be seen in *Figure 4*, the packet selection process happens immediately after a packet has had its header (addressing and destination information) examined and a timestamp assigned.

[4] http://www.rfc-editor.org/rfc/rfc5470.txt

First, the sampling functions are applied that determine whether the packet needs to be included in the flow generation process through straightforward selection preferences. A sample function's duty might be to only select every 100th packet. If there is no sampling function defined, then all packets are included.

Next, the filtering functions are applied that determine whether a packet needs to be included based on selection patterns applied to its attributes. For example, a packet could only be included in the flow generation process if the its associated protocol is TCP and destination port is less than 1024. If no filtering functions are defined, then all packets are included.

3. BOTNET DETECTION MODULE DESIGN

As it can be seen in *Figure 5* below, the Botnet Detection software module design specifies the logical position of its five major components and the information flow between them, illustrated by the numbered yellow arrows. The data flow direction is given by the yellow arrows, while the sequence in which it takes place is indicated by the numbers 1 to 7, 1 being the data entry point and 7 being the data exit point, both of which being facilitated by the AMDS Connection module. For more information on the Connection module's design, please see [1].

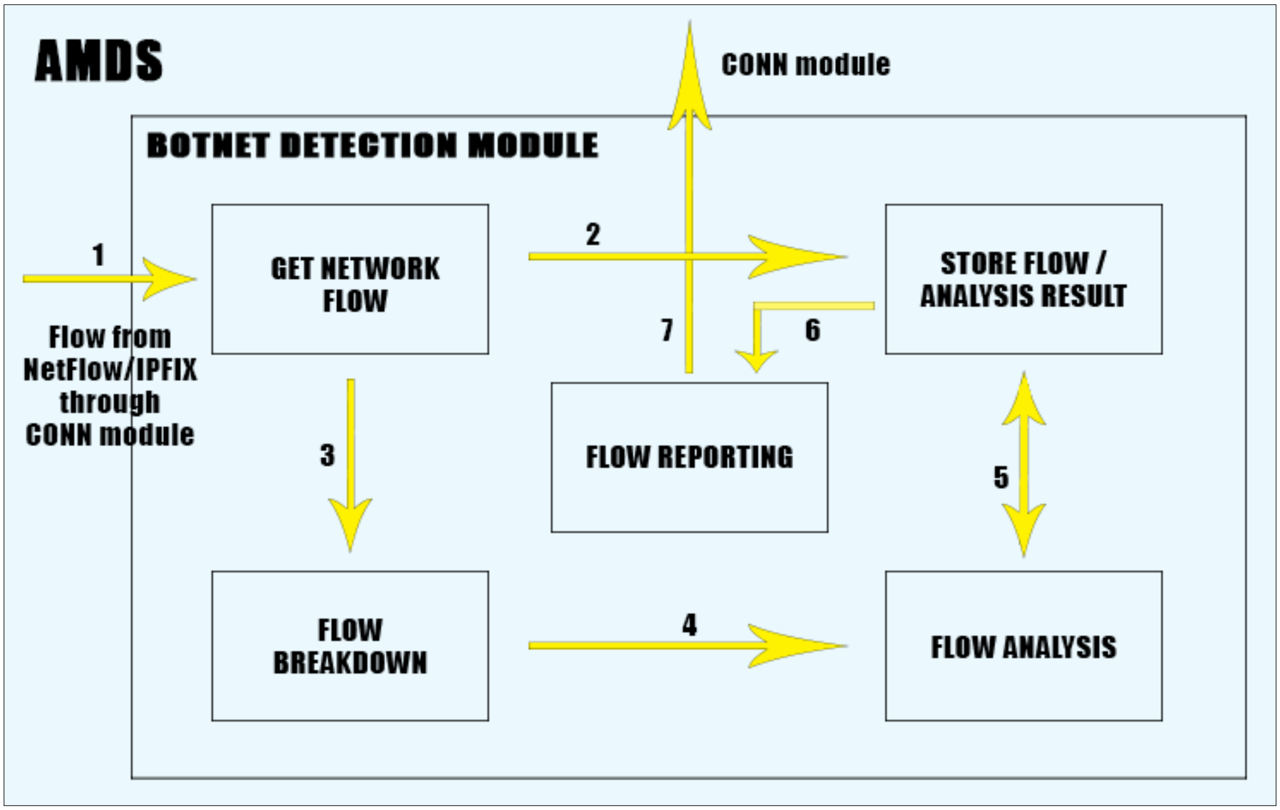


Figure 5 - Botnet Detection Module Design

This new module's major components are all designed to each fulfil a specific function, thus feeding into the AMDS modular design concept. They will now be described at an overview level, as follows:

I. *Get Network Flow*. This component, as its label suggests, is responsible for requesting and accepting network data flows from the NetFlow/IPFIX setup through the AMDS Connection module. It first initiates the request, which is then forwarded by the Conn module using an appropriate communication interface, and it then awaits a response in the form of a Network Flow. Once it receives this information, it passes it on to both the storage and breakdown components.

II. *Store Flow / Analysis Result*. This part of the Botnet module is responsible with storing, as a long-term solution, raw Network Flows as forwarded by the previously discussed component and analysis results as transmitted by the *Flow Analysis* component, described below. It uses an internal storage system rather than relying on the other AMDS modules for security considerations. Another responsibility it carries refers to passing on the analysis results to the *Flow Reporting* component.

III. *Flow Breakdown*. This element is charged with extracting key bits of information from the raw network flows, as required by the detection algorithm, and forwarding them to the *Flow Analysis* component. Generally, it looks for packet size, IP addresses and ports for both packet source and destination, class of service, device interface, and protocol type.

IV. *Flow Analysis*. In this part is where the bulk of the module functionality exists. It is the embodiment of the detection algorithm and, as such, is charged with utilising the information extracted from the raw network flows. The main purpose of this component is to attempt to detect malicious botnet activities by comparing current findings with past findings. This is achieved through querying the *Store Flow / Analysis Result* component for old records of previously analysed data and comparing healthy flows with current, yet unidentified flows. In the event of an inconclusive analysis result, it flags the current flow as such. Finally, it forwards its findings to the storage component.

V. *Flow Reporting*. This element's main responsibility is to retrieve analysis results and forward them to the User Interface AMDS module for review purposes. It also is charged with forwarding analysis statistics based on past results, which helps provide an overview of this module's activities.

4. EXPERIMENTAL SETUP

The scope of this experiment is limited to AMDS network sampling through the use of NetFlow and IPFIX open source software by the Botnet Detection Module. It has several key points it is achieving:

I. Sample 10% of all network data flow using IPFIX / NetFlow.

II. Sort collected samples into logical groups based on parameters such as data packet Size, Source, Destination, and Commands.

III. Test AMDS' ability to query stored data packets and analyse their respective groups in an attempt to discover unusual network behaviour.

IV. Aid with constructing an operational model based on packet analysis results capable of detecting suspicious client behaviour.

The experiment has employed the following parameters:

A. 1Gbps network connection speed between the clients and the AMDS, which allows a maximum of 1 billion bytes per second.

B. Data packet sample size was set at 10% of all traffic at the point of collection.

C. Average data packet size ranged between 500 and 1000 bytes of data.

D. Infected (Botnet) packets have been used randomly starting with Sample #500.

In *Figure 6* below, a new logical node has been created and inserted in-between existing logical network nodes (VLAN #1-3, and the ASA). This allows for potentially all data packets, the ones coming into the network as well as the ones going outside the network, to be stored in a local database and inspected at a very low level. Every packet has the option of being grouped up with other similar packets for easier comparison.

The IPFIX / NetFlow logical node is capable, through outside (other locally networked devices) interference, to sample random data packets passing through them. Although they are capable of sampling

100% of the data, this is not recommended as it would slow down network flow as well as increase power consumption on the node they reside. For these reasons only 10% of all data is sampled, stored, and grouped up in the local IPFIX / NetFlow database.

The local data packet database has the ability to be queried for small portions of data at a time for easier analysis. Also, the AMDS has the capability of achieving this task through one of its built for purpose modules. Once retrieved, the AMDS creates a graph of data packets by comparing their Size, Source, Destination, and Commands they carry. The more regular data it analyses, the higher the chance of it detecting unusual behaviour on the network, and the lower number of false positives.

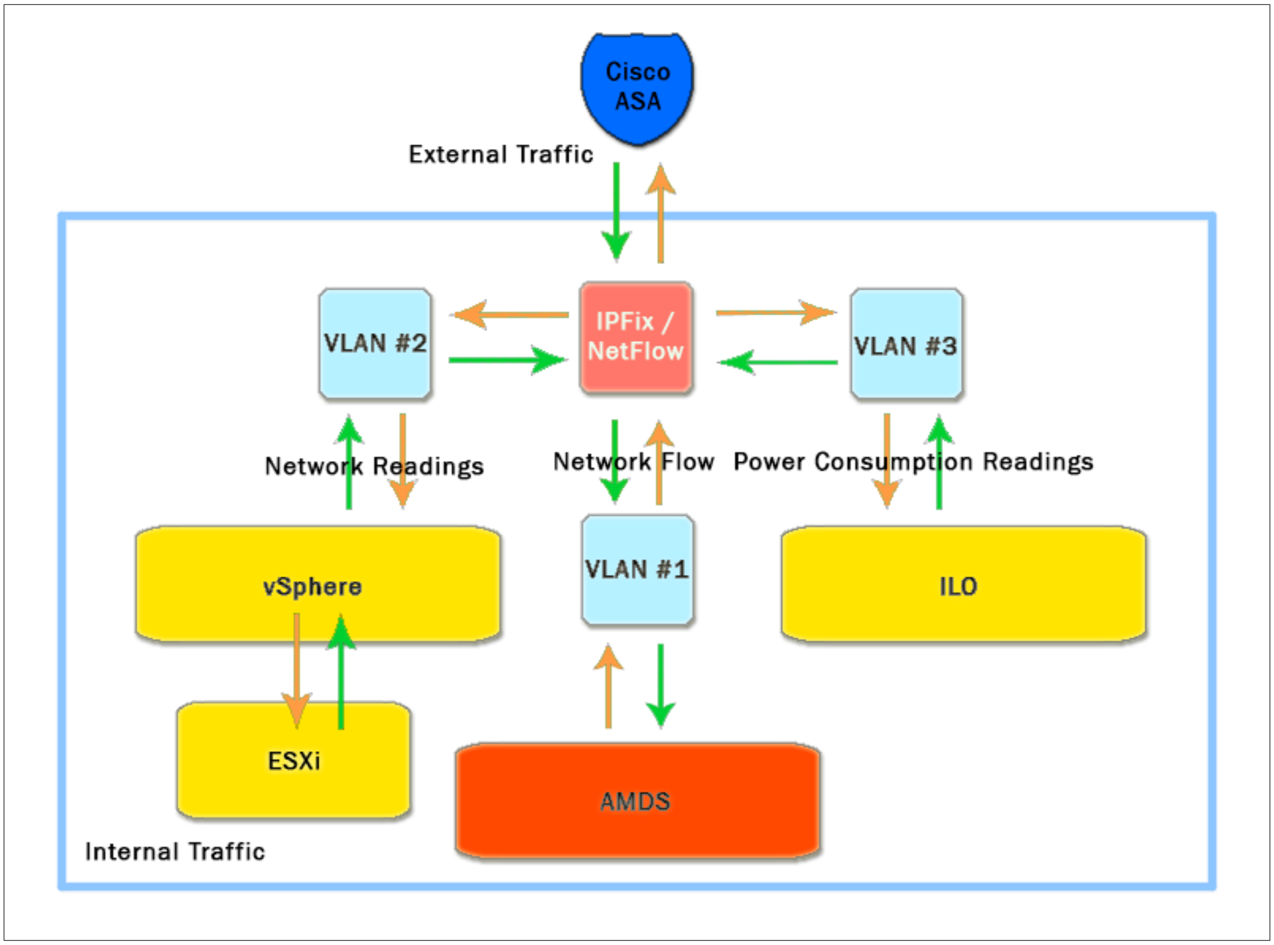


Figure 6 - AMDS Traffic Sampling Network Layout

5. EXPERIMENTAL RESULTS

The experiment has run over an extensive period of time and it has produced a great deal of log data. This data, along with some of the experiment parameters, can be seen in *Table 1* below.

For the first half of the experiment, as it can be seen in *Table 1*, regular packets with an average size of 500 bytes have been filtered through the AMDS Botnet Detection Module. This has been used as a training mechanism for the heuristic algorithm, so it would later on have a healthy packet model to compare infected packets against.

As it can be seen in *Figure 7*, using the 1Gbps network link has evaluated into approximately 2 million data packets, of which 200 thousand have been sampled at each of the 500 initial readings for analysis. Having an increased average packet size has impacted the samples containing infected Botnet packets in addition to the regular packets by having a reduced size. These samples were made up of between 100 thousand and 133 thousand packets on average.

For the second half of the experiment, a random percentage of Botnet generated data packets have been introduced instead of the regular data packets used in the first half of the experiment. This had a direct impact on the data packet size as this has increased the average packet size of the samples by 50%, from 500 to 750 bytes each. The Botnet packets have been created by combining botnet and client commands in one single packet, resulting in a data packet with an average size of 1000 bytes.

Sample #	# of Packets (1000s)	Avg. Packet Size (B)	# Infected (1000s)	# Detected (1000s)	Detection Rate (%)
50	200	500	0	0	0
250	200	500	0	0	0
500	133	750	18	5	28
1000	100	1000	28	12	43

Table 1 - Packet Analysis Results

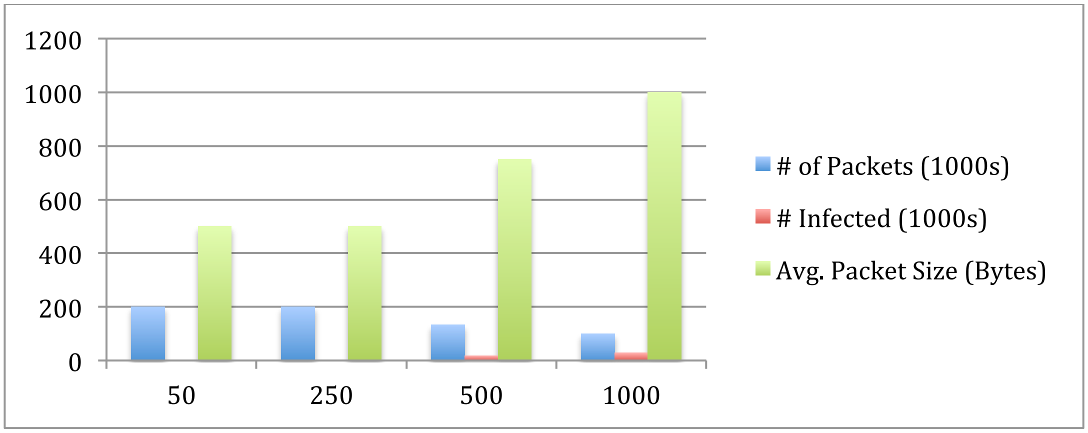


Figure 7 - Packet Distribution per 10% Sample

As it can be seen in *Figure 8*, the AMDS, through the use of its Botnet Detection heuristic algorithm, has managed to detect approximately 28% of all infected packets at the start of the Botnet attack. This detection rate has steadily increased up until the end of the experiment to approximately 42% of all infected packets.

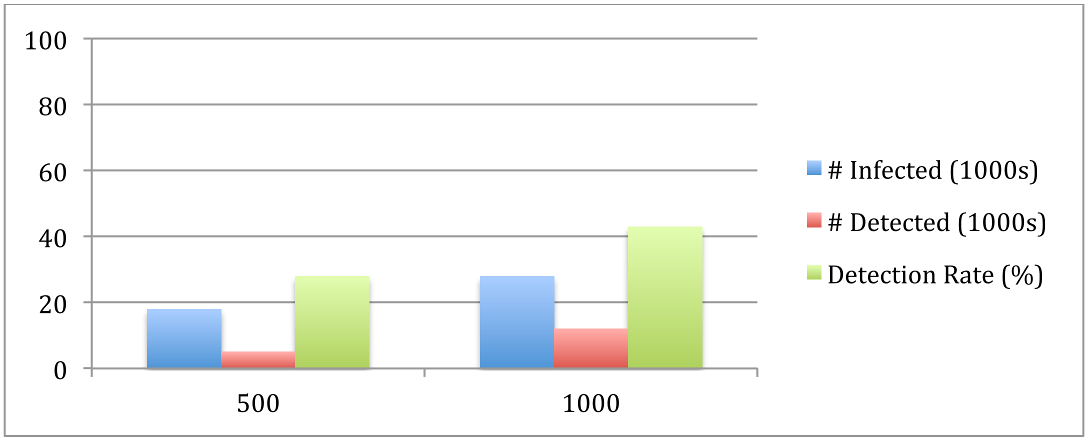


Figure 8 - Botnet Packet Detection Rate

6. CONCLUSION

The implications of the results shown in *Figure 8* go beyond just detecting a potential botnet attack in a datacentre. Although this fact alone give these results meaning, it also allows for an abstract software model to be defined, which can then be implemented using different programming languages in a multitude of different computing environments.

The main value of this experiment is apparent and will become even more so in time, when after several more data sets have been analysed, a comprehensive detection model will be created and continuously refined, which can used to test all future packet samples in an attempt to discover new, zero-day malicious activities.

The results presented above give a clear indication of the potential of the AMDS having its Botnet Detection Module activated. Applying the heuristic algorithm to more and more data packet samples allows the AMDS Botnet Detection module better understand what Botnet data packets look like, and detect more similar packets or even unknown Botnet packet type in the future.

REFERENCES

[1] Dinita, R. I., Wilson, G., Winckles, A., Cirstea, M., Rowsell, T. (2013). A Novel Autonomous Management Distributed System for Cloud Computing Environments. In *Industrial Electronics Conference (IECON), 2013 39th Annual Conference of*. IEEE.

[2] Zeidanloo, H. R., Shooshtari, M. J. Z., Amoli, P. V., Safari, M., & Zamani, M. (2010, July). A taxonomy of botnet detection techniques. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on* (Vol. 2, pp. 158-162). IEEE.

[3] Binkley, J. R., & Singh, S. (2006, July). An algorithm for anomaly-based botnet detection. In *Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI)* (pp. 43-48).

[4] Jeong, H. C., Im, C. T., & Oh, J. H. (2010). *U.S. Patent Application 12/942,700*.

[5] Liu, L., Chen, S., Yan, G., & Zhang, Z. (2008). Bottracer: Execution-based bot-like malware detection. In *Information Security* (pp. 97-113). Springer Berlin Heidelberg.