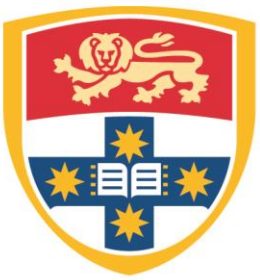
THE UNIVERSITY OF
SYDNEY



THE UNIVERSITY OF
SYDNEY

Resilience Strategies for Network Challenge Detection, Identification, and Remediation

Yue Yu

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

School of Information Technologies
University of Sydney

2013

Abstract

The enormous growth of the Internet and its use in everyday life make it an attractive target for malicious users. As the network becomes more complex and sophisticated it becomes more vulnerable to attack. There is a pressing need for the future internet to be resilient, manageable and secure.

Our research is on distributed challenge detection and is part of the EU Resumenet Project (Resilience and Survivability for Future Networking: Framework, Mechanisms and Experimental Evaluation). It aims to make networks more resilient to a wide range of challenges including malicious attacks, misconfiguration, faults, and operational overloads. Resilience means the ability of the network to provide an acceptable level of service in the face of significant challenges; it is a superset of commonly used definitions for survivability, dependability, and fault tolerance.

Our proposed resilience strategy could detect a challenge situation by identifying an occurrence and impact in real time, then initiating appropriate remedial action. Action is autonomously taken to continue operations as much as possible and to mitigate the damage, and allowing an acceptable level of service to be maintained. The contribution of our work is the ability to mitigate a challenge as early as possible and rapidly detect its root cause.

Also our proposed multi-stage policy based challenge detection system identifies both the existing and unforeseen challenges. This has been studied and demonstrated with an unknown worm attack. Our multi stage approach reduces the computation complexity compared to the traditional single stage, where one particular managed object is responsible for all the functions.

The approach we propose in this thesis has the flexibility, scalability, adaptability, reproducibility and extensibility needed to assist in the identification and remediation of many future network challenges.

Acknowledgements

I would like to express my gratitude to all those who gave me the possibility to complete this thesis and PhD study at the University of Sydney. First and foremost, I would like to acknowledge the people, who mean world to me, my wonderful parents. I can't imagine a life without their love and blessings. I would like to thank for their endless love, support and encouragement. They deserve my whole hearted thanks. It wasn't an easy decision to back university and start long term commitment for the PhD project after working in the industry as professional for years. Sometimes, life is about giving up something at this stage and fulfilling the dreams. That was a big decision and major turning point in my life.

My beloved parents are always the ones who gave me strength to reach for the stars and chase my dreams. They raised me with a love of knowledge and supported me in all my pursuits. I am very much indebted to my parents, as they are the greatest persons in my life. They encourage and help me at every stage of my personal and academic life. Their unconditional support, their patience and understanding are greatly appreciated. I consider myself the luckiest in the world to have such a supportive family, standing behind me with their love.

I also would like to express my sincere gratitude to my supervisor, Professor Michael Fry, for the continuous support of my PhD study and research, for his patience, motivation, enthusiasm, encouragement and immense knowledge. He is the one who gave me the opportunity to follow my dream, let me participate in a very interesting project. His guidance helped me in all the time of research and writing of this thesis.

A special thanks to Professor Aruna Seneviratne. It is my great honor to be his student. I appreciate all his contributions of time, ideas, and funding to make my PhD. experience productive and stimulating. Thank Aruna for giving me the strength and patience to

work through all these years, motivating and inspiring every bit of me towards new possibilities in life. He is the one who has vision with the network filed, and guided me through the development of good collaboration, and eventually lead me to reach the goal of my project.

The expert assistance, hospitality and mentorship of Assoc Prof. Bob Kummerfeld and Prof. Judy Kay are particularly important for the whole period of my PhD. Their constant guidance, cooperation and support have always kept me going ahead. The joy and enthusiasm they have for their research was contagious and motivational for me, even during tough times in the PhD. pursuit. They are the ones bring me into the CHAI group, let me become a member of the group family.

My special words of thanks should go to Professor Bernhard Plattner, not only for his good advice but also unsurpassed knowledge of network security. I truly appreciated his willingness to help, and as a result, it was a pleasure to have our joint work published on the international conference. Especially thanks for giving me the opportunity to visit ETH Zurich in Switzerland for two months. His guidance, suggestions and support have been invaluable for my project development. Thanks for introducing me to the ETH Zurich network group and SWITCH. To Dr. Xenofontas Dimitropoulos and Prof. Eduard Glatz, I was grateful for the collaboration, assistance and interpretation of some results presented in this thesis. Thank Dr. Dominik Schatzmann, for the detailed discussion with SWITCH. To Dr. Bernhard Ager, Dr. Brian Trammell, Dr. Stephan Neuhaus and others in the ETH Zurich, it was great to join your group seminar every week and have individual discussion about your security project.

I extend my sincere word of thanks to Prof. David Hutchison, Dr. Paul Smith and Assoc Prof. Alberto Schaeffer-Filho, for their valuable support and teamwork in the European Union ResumeNet project. In addition, I would like to thank Dr. Thomas Gamer and Dr. J.C. Maureira for their useful discussions on the OMNeT.

I also would like to acknowledge that part of our project was carried out under the European Union Research Framework Programme 7 via the ResumeNet project with contract no. FP7– 24619, so thank all the partners of the ResumeNet project for their contribution to this

work. Also I acknowledge NICTA (National ICT Australia) for providing me with the necessary funding and fellowship to pursue research.

Finally, I would like to thank all my friends, particularly the ones in NICTA and University of Sydney, Your friendship makes my life a wonderful experience. I cannot list all the names here, but you are always in my mind.

My Publications Drawing On the Work

Yue Yu, A Distributed Challenge Detection System for Resilient Networks, DSRC (in conjunction with QShine2010), 7th International ICST Conference, Vol. 74, LNICST (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), pp. 584-597 , Springer, Houston, USA, Nov, 2010

Yue Yu, Michael Fry, Alberto E. Schaeffer Filho, Paul Smith, and David Hutchison, An Adaptive Approach to Network Resilience: Evolving Challenge Detection and Mitigation, Design of Reliable Communication Networks (DRCN), pp.172-179, IEEE, Krakow, Poland, 2011

Alberto E. Schaeffer Filho, Paul Smith, Andreas Mauthe, David Hutchison, Yue Yu, and Michael Fry, A Framework for the Design and Evaluation of Network Resilience Management, IEEE/IFIP Network Operations and Management Symposium (NOMS), USA, Apr, 2012

Yue Yu, Michael Fry, Bernhard Plattner, Paul Smith, and Alberto E. Schaeffer Filho., Resilience Strategies for Networked Malware Detection and Remediation, International Conference on Network and System Security (NSS), Vol. 7645 of Lecture Notes in Computer Science, pp. 233-247. Springer, China, 2012

Alberto E. Schaeffer Filho, Andreas Mauthe, David Hutchison, Paul Smith, Yue Yu, and Michael Fry, PReSET: A toolset for the evaluation of network resilience strategies. The 13th IFIP/IEEE International Symposium on Integrated Network Management (IM), Belgium, 2013

Table of Contents

List of Figures

Chapter 1

Introduction

With the growth of networks and the integration of services, increasingly severe consequences come from the disruption of networked services. Quality of life, the economic viability of businesses, and the security of nations are directly linked to the resilience, survivability, and dependability of the global network. However, the network becomes vulnerable with the increased dependence and sophistication of services. The scale of growth and deregulation bringing numerous service providers has resulted in a network that is difficult to manage. There is a pressing need for better resilience, manageability, and security for the future network. Therefore, our research in distributed challenge detection is aimed to make networks more resilient to various challenges. Resilience means the ability of the network to provide an acceptable level of service in the face of challenges to normal operation. This service includes the ability for users and applications to access information when needed, the maintenance of end-to-end communication

association, and the operation of distributed processing and networked storage.

Research in distributed challenge detection is designed to make networks more resilient to a wide range of challenges, including:

- unintentional misconfiguration or operational mistakes;
- large-scale natural disasters;
- malicious attacks from intelligent adversaries against the network hardware, software, or protocol infrastructure including DDoS (distributed denial of service) attacks;
- environmental challenges of mobility, weak channels, and unpredictably long delay;
- unusual but legitimate traffic loads
- provider failure.

Therefore, the definition of resilience is a superset of commonly used definitions for survivability, dependability, and fault tolerance.

Our research investigates a more coherent and systematic approach to meeting challenges to the Internet than has occurred before. The key strategy includes monitoring the network to detect the onset of challenges in real time. Then diagnosing and identifying the challenge in order to initiate appropriate remedial action, it could also be used to characterize a set of malicious or normal operational challenges. This novel remedial action allows an acceptable level of service to be maintained.

1.1 Background

Computer and communication networks are increasingly critical in supporting business, leisure and daily life in general. There is also an obvious increase in cyber-attacks on networked systems. Thus, there is a compelling need for resilience to be a key property of networks. The nature of the challenges typically requires the use of mechanisms across multiple layers of the protocol stack and in disparate locations of the network. Therefore, ensuring the resilience of a network requires the systematic design and evaluation of

resilience strategies, the careful coordination of various monitoring and control mechanisms, and the capture of best practices and the experience of network operators into reusable configurations of resilience mechanisms. It requires the on-demand adaptation of network configurations, including specialized resilience functionality, in response to performance degradation, component faults or security threats. Because of the consequences of defining poor management configurations, e.g., further degrading service when the network is under duress, it is necessary to carefully specify and test their performance.

This research has been carried out as part of the EU ResumeNet project (Resilience and Survivability for Future Networking: Framework, Mechanisms and Experimental Evaluation) that is supported under the EU FP7 FIRE (Future Internet Research and Experimentation). Network resilience is considered in the context of a general two-phase high-level strategy, called D^2R^2 + DR: as demonstrate in Fig.1, it includes Defend, Detect, Remediate, Recover + Diagnose, Refine [Sterbenz et al., 2010].

The resilience aim can be generally achieved via a six-step strategy:

- Defence, according to which the Internet is made robust to challenges and attacks;
- Detection of an adverse event or challenge that has impaired normal operation of the Internet and degraded services;
- Remediation in which action is autonomously taken to continue operations as much as possible and to mitigate the damage;
- Recovery to normal operations once the adverse event has ended or the attacker has been repelled;
- Diagnose the root cause of the challenge that impaired normal operation. This could be used to improve the system design and affect the recovery to a better state;
- Refinement of future behavior based on reflection of the previous cycle

The first phase comprises the use of defensive measures to protect the network from foreseeable challenges, the ability to detect real-

time challenges that have not been anticipated and subsequently remediating their effects before the network operation is compromised, and finally disengage possibly sub-optimal mechanisms via specific recovery procedures. The second phase caters for the longer-term evolution of the system, through the diagnosis of the causes of the challenge and the refinement of the system operation. In particular, D^2R^2 can be seen as a conceptual online control-loop for network resilience operation. Central to the strategy is the management and reconfiguration of interacting detection and remediation mechanisms operating in the network infrastructure. This is the focus of the work reported in this thesis.

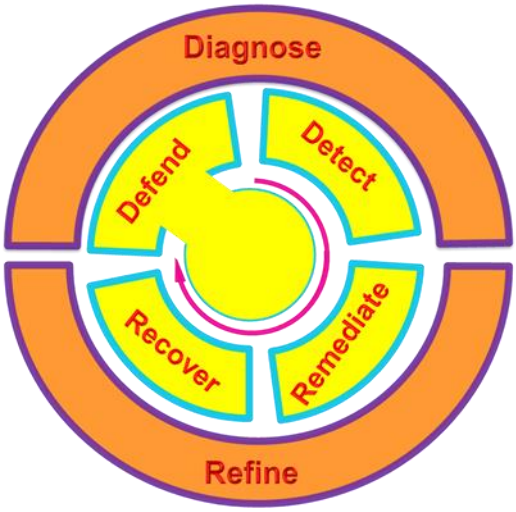


Fig. 1. Network resilience strategy

The central notion, first reported in [Yu et al, 2011], is that of multi-stage resilience strategies, in which the configuration of detection and remediation mechanisms deployed in the network is dynamically refined as new information about challenges becomes available. Policies are used to control the operation of these mechanisms, and how they should be reconfigured in the face of new types of challenges or changes in their operational context.

1.2 Current Approaches

A great deal of research effort has focused on the development of detection and classification techniques, whose output could be used to trigger the dynamic adaptation of networks to ensure resilience. For example, for detecting attacks, signature-based systems are

currently widely used [Roesch, 1999], and there exist various approaches to anomaly-based detection of challenges, such as traffic volume anomalies [Hussain et al., 2003] or changes in network traffic feature entropy [Lakhina et al., 2005]. (For a survey of anomaly detection approaches we refer the reader to Chandola et al. [2009]). Similarly, a number of classification techniques exist that can be used to identify the cause of an anomaly. (For a survey of machine learning-based traffic classification techniques we refer the reader to work by Nguyen and Armitage [2008]). Despite the multitude of mechanisms and techniques proposed, it is often not clear how these should be combined and coordinated to enforce effective resilience strategies in complex multiservice networks. We expand further on related work below.

1.3 A Proposed Approach to Network Resilience

Communication networks must be resilient to a multitude of malicious attacks and other challenges to their operation. A classification of challenges to the global Internet and interdependent networks has been defined, in which challenges are any characteristic or condition that impacts the normal operation of the network [Schaeffer-Filho et al., 2013]

A resilience strategy to address such challenges requires the management and reconfiguration of interacting detection and remediation mechanisms that operate in the network infrastructure. Initially, detection mechanisms support the identification and categorization of challenges to the network. They may vary from a simple link monitor that can determine whether high volumes of traffic are being observed, to sophisticated detection systems and traffic classifiers that detect anomalous changes in traffic features. Similarly, a range of remediation mechanisms may be used for containing the effects of a challenge. For example, various forms of traffic shaping can be used, from simply blocking traffic to probabilistic rate limiting, which can be applied at different protocol levels and to individual network device ports. Firewalls and OpenFlow switches [Yu et al., 2011], for example, can be used to block or shape network traffic. A typical network infrastructure of the kind we are considering, which includes a range of resilience mechanisms, is shown in Fig. 2.

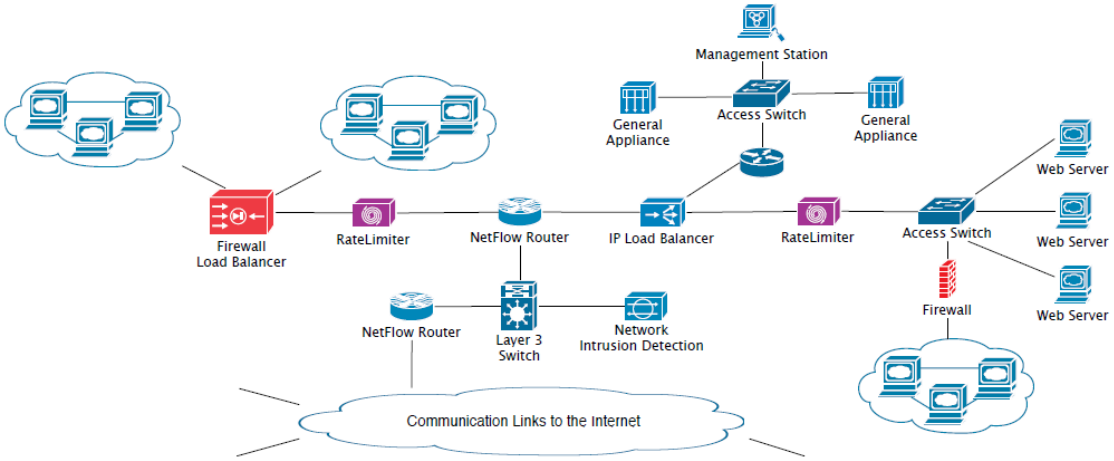


Fig. 2. Network infrastructure with mechanisms implementing a set of resilience functions and services

In our work, resilience strategies are defined as a set of policies that reconfigure the operation of resilience mechanisms at run-time in response to events, such as high link utilisation, malicious attacks or equipment failures, for example [Limoncelli, 2012]. Policy-based management is used to control the operation of these mechanisms in the face of new types of challenges. Thus, different types of mechanisms can be selectively enabled or reconfigured in specific operational contexts.

1.4 A Distributed Challenge Detection Framework

Figure 3 introduces a distributed challenge detection framework, developed under the ResumeNet project, which is the context for the research presented here. From the outset, we need to understand the complexity of systems, the various challenges and their root cause. Assessing which challenge affects the system in which way is essential to deciding the corresponding mitigation strategy. For example, a web server could be overloaded in the short term with many requests, greater than it is provisioned for, and this could be a DDoS attack or a ‘flash crowd’ event, which is the unusual but legitimate demand for service. We need to distinguish between challenges that have similar symptoms, since they require different remediation.

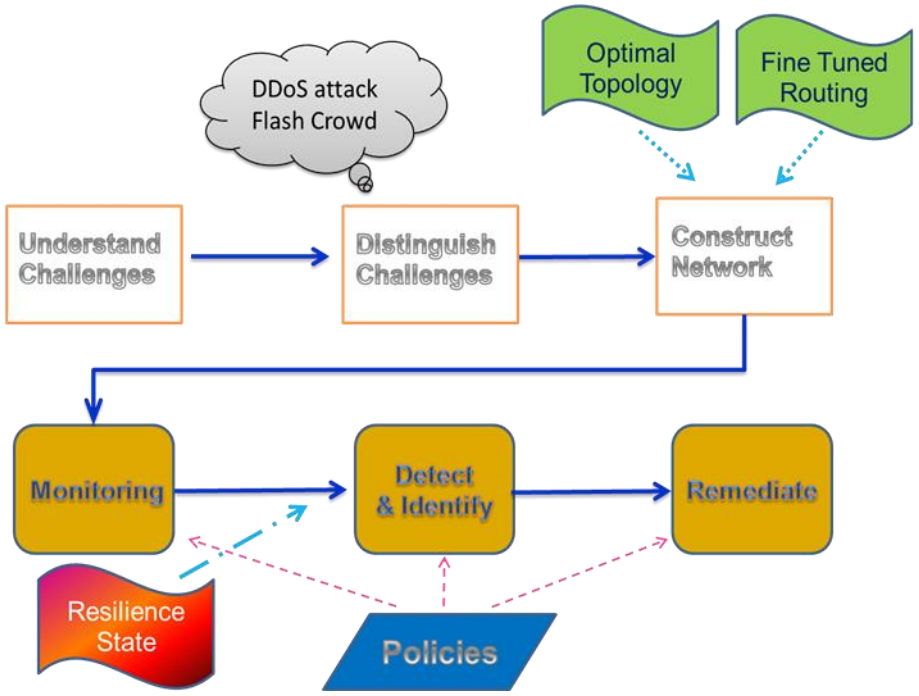


Fig. 3. Distributed Challenge Detection Framework

To do this, the first task is to construct a network that could tolerate foreseen adversarial events. The optimal topology needs to be designed. It utilizes the best possible way to interconnect the access nodes with the corresponding transmission technology. For example, a fine-tuned routing mechanism will allow good conditioned connections to be built so that they could react to the failure quickly.

However, for challenges that circumvent network defences, we need a distributed monitoring and assessment platform that could detect and identify challenges in real time, and initiate remediation in order to neutralize the challenge and maintain acceptable levels of operation. Real time detection and remediation is the focus of this thesis.

1.5 Contributions

In summary, this thesis proposes and investigates a multi-stage, policy based strategy for achieving network resilience. The initial idea arose through discussions between ResumeNet project partners. The work of this thesis, first reported in (Yue et al, 2011),

has contributed to consequent detailed development and assessment.

The current state-of-the-art, as reviewed in chapter 2, has tended to focus on improved detection mechanisms for particular challenges, such as anomaly detection or fault identification. There has been little focus on how to embed resilience to such challenges within operational networks, while addressing temporal and resource constraints. Furthermore, in contrast to the state-of-the-art, we present a unified approach for dealing with diverse network challenges.

The multi-stage, policy based framework demonstrated in this thesis is capable of operating in real time, progressing from initial detection through identification and remediation. By initially using lightweight detection and then progressively applying more heavyweight analysis, a key contribution of our work is the ability to mitigate a challenge as early as possible and rapidly detect its root cause. The multi stage approach can effectively reduce computational complexity compared to existing single stage approaches.

A technical contribution of this thesis is to realize and evaluate this framework as a number of policy-enabled Managed Objects (MOs) which co-operatively enforce the resilience of the network. Each MO will only be triggered by a local manager MO at the time it is needed rather than executing continually. We have implemented and demonstrated the framework within a simulation environment. This circumvents the immense difficulties of experimentation within large operational networks, but is by definition an abstraction of real networks. However, we advocate that the same approach could be implemented in network devices with open APIs.

Through a number of case study implementations this thesis demonstrates how the proposed approach can enable:

(1) the rapid deployment of appropriate resilience strategies when challenges are observed at run-time,

(2) autonomic challenge detection and identification, with simultaneous deployment of remediation strategies,

(3) the use of this common framework to address diverse challenges such as volume challenges, network malware and network faults,

(4) detection and mitigation of both known and unknown challenges,

(5) flexibility and extensibility to new challenges as they emerge and evolve.

Finally, the work of this thesis has contributed a simulation environment that can serve as the basis for future work and evaluation.

Some advantages of our approach, as demonstrated in this thesis, are as follows.

Efficiency and scalability: our simulation results demonstrate the effectiveness and efficiency of our detection strategy for a large network topology. It scales well in a network with large number of hosts and servers, and also provides a response within a reasonable time.

Generality: the multi staged policy based platform has the generality to make the network more resilient by detecting a wide range of challenges, for example, DDoS attacks, worm propagation, network faults etc. Different challenges can be investigated using the same platform.

Reusability: our network resilience strategy is realized by the cooperation among a number of distributed objects. For different challenges, we have different detection and remediation strategies. However, some MOs are reusable and can be applied to various challenge scenarios. Careful combination of some modules might be synergistic and bring a broader range of challenge detection.

Extensibility: new challenges will continuously appear. We should be able to easily extend our proposed platform to detect new challenges. This could be realized by developing new policy-base strategies and/or adding new MOs.

Flexibility and Ease of Use: the policies can be configured in a flexible and easy way. The parameters of the remediation strategy can simply be adjusted to filter different characteristics of the traffic.

1.6 Thesis Organisation

The thesis discusses a resilience strategy for detection and remediation of network challenges. The rest of this thesis is organized as following:

In chapter 2, we introduce the background and the related work. A survey of existing approaches is provided, including network monitoring, network detection and remediation approaches, and commercial products.

In chapter 3, we provide the insight of a multi-stage policy-driven approach to network resilience. The motivation and progressive challenge identification will be discussed. Policy- based management strategies and the resilience pattern is proposed to explain how could we apply policies as the strategies to build into our framework in order to make the network more resilient. Benefits of the approach will then be highlighted

In chapter 4, we compare different experimental platforms. The well known published off-line IDS datasets, for example, DARPA and KDD Cup will be discussed. We explain the reason why these datasets were not chosen for our experiment. Following this, we evaluate various network simulator options and clarify why OMNeT is selected as the experimental platform.

In chapter 5, the volume challenge resilience strategies are proposed. A specific DDoS attack resilience approach is presented to explain how we develop strategy for DDoS resilience. The incremental DDoS Detection and Remediation strategy and experimental result will be demonstrated. Following this we will discuss how to differentiate flash crowd events from DDoS attacks. Then discuss the current flash crowd identification methodologies.

In chapter 6, our framework is extended to perform the malware attack resilience strategy. The workflow for malware attack resilience is proposed, coupled with experimental results for various types of existing worm, unknown worm and port scan malware detection and remediation.

In chapter 7, we show our multi stage network fault resilience strategy. Two approaches are demonstrated here. One is extended

from our previous developed multi stage entropy based network resilience approach to identify network fault. The other one is developed as the collaboration work during my visit to ETH Zurich, the classifier based approach could work both on the real dataset SWITCH network also our network simulation.

In chapter 8, we discuss key metrics to evaluate network resilience strategies. The metrics include generality, efficiency and scalability, flexibility and evolution. We then summarize the main contributions of this thesis and identify future work in our research.

Chapter 2

Related Work

In this chapter, we summarize our earlier work on characterizing the state-of-the-art in challenges detection and classification. We then discuss how related commercial managed security solutions compare to the approach presented in our work. After this, we discussed how the evaluation of the performance carried in the previous work.

2.1 Network Monitor

The vulnerabilities of the current network and the need for resilience are widely acknowledged. There has been considerable research into network monitoring, anomaly detection, fault tolerance, attacks, anomaly modeling engine separately. We reviewed the latest technologies in these areas. With the current network monitoring techniques, threshold based random walks for fast portscan detection is unscalable [Jung et al., 2004]. The proposed TCAs

(threshold crossing alerts) require the cooperation of the manufacturers to run on the network devices, which will be difficult. Or they can run on separate hardware, which will be complex [Wuhib et al., 2006]. Jackson et al. [2007] cope with the distributed monitor problem in internetworks, but the capability to monitor every link cannot be assumed.

Today most detection use signature-based IDS that detect known attacks only. In contrast, anomaly detection is effective in identifying known and unknown attacks. Real time volume based anomaly detection is resource challenge [Fry et al., 2010]. The causes for DoS attacks and mechanisms for defending is surveyed [Peng et al., 2007], however it is not yet practical to identify attack paths and we require global cooperation to combat DDoS (Distributed denial of service) attacks. To evaluate the impact of faults, fault injection is considered as the first stage. It offers a cost and time effective way to test system. The method to inject faults into the real network is proposed in [Labovitzt al., 2001]. Other approaches are presented to study network survivability. Random events affect node and link availability, so cause the failures. There is a complete survey of fault localization [Sethi, 2004]. Open research problems still remain with multi-layer fault localization, temporal correlation, and distributed diagnosis.

Increasingly, passive monitoring systems are used to collect data to observe the underlying behavior of Internet, - fully instrumenting any but the smallest of networks is impractical and it is impossible to even partially instrument large parts of the Internet. Shavitt et al. [2005] present DIMES, a highly distributed global Internet measurement infrastructure, which measure the structure of the Internet using a large set of interacting measurement agents. The main goal is to take full snapshots of the Internet graph, in the AS, PoP, and router level, annotated with delay and loss statistics, in fine time resolutions. In future, the authors plan to embed the measurement results in a geographic metric, and develop measures which use the Internet evolution characteristics in various regions as an indicator of economic and social evolution. However, the simulation shows the BGP topology has about 25% more nodes than the DIMES topology. This may need to improve in future. Until today, this project still update so that we could join the testing and volunteer

to be the agent, the monitored data will be helpful to establish our research approach as well.

DIME uses endsystem agents to monitor network properties. In contrast, ubiquitous endsystem network monitoring presented by Cook et al. [2006] is more closely related to in-network monitoring approaches. It provides more detailed view of traffic and solves the inability to collect the required information using network-only measurements. The measurement systems must themselves apply the end-to-end principle. It restores much of the lost visibility and enables new applications. It provides additional privacy protection compared to other approaches, also enables selected highly trusted systems to have different privacy policies. Using real data from an enterprise shows that collecting and querying data from endsystems in a large network is practical. In addition, service level monitoring is necessary to ensure that service levels are adhered to SLA. The key tool for this is threshold crossing alerts (TCAs), which can notify a service provider that a certain parameter has exceeded a certain threshold value, directing attention to those areas where prevention needs to be taken. Wuhib et al. [2006] present a protocol (TCA-GAP) and a decentralized architecture that implements a new category of TCAs for parameters that need to be aggregated across a network, as opposed to parameters that can be observed from a single device. Nevertheless, TCA-GAP can run on the network devices themselves but will require the cooperation of the device manufacturers, which will be difficult to fulfill. Or they can run on separate hardware, which will be complex.

Jackson et al. [2007] analyze a subset of the general monitor placement problem where the goal is to maximize the coverage of the entire universe of potential communication pairs. It suggests that the preferred strategy to place monitors should be to instrument one or two specific inter-AS links per AS for many ASes. However this approach do not address sampling in their monitors, they deal with the problem of distributed monitoring in internetworks, where the capability to monitor every link cannot be assumed.

2.2 Network Detection and Remediation

Diagnosing anomalies is critical for network operators and end users. Lakhina et al. [2004] propose a general method to diagnose

anomalies. It is based on a separation of the high-dimensional space occupied by a set of network traffic measurements into disjoint subspaces corresponding to normal and anomalous network conditions. This separation can be performed effectively by Principal Component Analysis (PCA). This method can accurately detect volume anomaly; correctly identify the underlying origin-destination (OD) flow which is the source of the anomaly; and accurately estimate the traffic involved in the anomalous OD flow. However, the methodology needs to be extended to diagnose additional network-wide anomalies, including routing related anomalies. Also, if incorporate these algorithms in a toolset, it can better prevent anomalies.

Lakhina et al. [2004] demonstrated the efficiency of statistical anomaly detection in aggregated network traffic. The most popular way to aggregate the traffic is as a Traffic Matrix, where the traffic is divided according to its ingress and egress points. This has been first successfully used by Lakhina et al. [2004]. Soule et al. [2007] further studied the reasons for choosing traffic matrices instead of any other by comparing three traffic aggregation formalisms: ingress routers, input links and OD pairs (i.e. traffic matrices). The traffic aggregation level has a significant impact on the number of detected and on the false positive rate. It also mentions the aggregation can be harmful to anomaly detection in two different ways. So the future needs to study more formalism, detection methods, data sets and how different formalisms are biased towards detecting certain types of anomalies. The aggregation with PCA is left for the future work. Ringberg et al. [2007] did the ongoing work. They show that tuning PCA to operate effectively in practice is difficult and requires more robust techniques. The methodology uses their interface with the PCA anomaly detector written by Lakhina et al. [2004]. They identify four main challenges of using PCA. Then mentioned before PCA can be used automatically, need more effective techniques for determining the dimensionality of the normal subspace, preventing its contamination, and identifying flows responsible for a given PCA detection.

Ringberg et al. [2007] also investigated the related work and demonstrated PCA has also been combined with distributed monitors to provide more efficient traffic anomaly detection. This has been designed by Li et al. [2006] named MIND as a distributed multi-

dimensional indexing system for network diagnosis. It is the first operational overlay with new features. Future work will use MIND to perform online anomaly detection in a large network and then implementing the actual algorithms that examine detailed traffic traces based on MIND's query results and also resolve open issues in the overlay design area. Furthermore, Peng et al. [2007] present distributed IDS based on an information sharing model. They address what information and how to share information problems by using the cumulative sum algorithm to collect statistics at each local system, it is simple but robust to monitor changes. Also they use an offline machine learning approach to coordinate the information sharing among the distributed IDS, which helps to decide when to share information so that both the communication overhead and the detection delay are minimized.

On the other hand, Androulidakis et al. [2009] evaluate the impact of intelligent flow sampling techniques on the detection of network anomalies. The research principle is the exploitation of the fact that for specific-purpose applications such as anomaly detection a large fraction of information is contained in a small fraction of flows. They demonstrate that by using sampling techniques that opportunistically and preferentially sample traffic data could improve detection. To evaluate the impact of intelligent sampling techniques, an entropy-based anomaly detection method on a packet trace is adopted. Recently Shanbhag et al. [2009] design a real-time parallel anomaly detection system by using multiple existing anomaly detection algorithms in parallel on thousands of network traffic subclasses to accurately and sensitively detect anomalies. They also propose a novel aggregation process to aggregate the vast amount of data generated. It increases the accuracy of the overall system beyond that of any single anomaly detection algorithm.

After all, Chandola et al. [2009] provide a comprehensive overview of the research on anomaly detection. They grouped existing techniques into different categories based on the underlying approach adopted by each technique. For each category they identified key assumptions which can be used as guidelines to assess the effectiveness of the technique. For each category, they provide a basic anomaly detection technique, and then show how the different existing techniques in that category are variants of the basic technique. After that, they identify the advantages and

disadvantages of the techniques, which will help to know which anomaly detection technique is best suited for a given problem. The set of technologies covered classification based, clustering based, nearest neighbor based, statistical, information theoretic, spectral anomaly detection. The application domain contains cyber-intrusion detection, fraud detection, medical anomaly detection, industrial damage detection, image processing, textual anomaly detection and sensor networks. They also provide a discussion on the computational complexity of the techniques.

So the future in this field, will 1st, unifies the assumptions made by different techniques regarding the normal and anomalous behavior into a statistical or machine learning framework. 2nd, develop new techniques in contextual and collective anomaly detection area. 3rd, the need for distributed with privacy-preserving anomaly detection techniques. 4th, with the emergence of sensor networks, processing data as it arrives has become a necessity. 5th, finding more and more applicability is in complex systems.

2.2.1 DDoS

Most intrusion detection systems in use today, such as the popular Snort [Roesch et al., 1999], are based on signatures. More generic anomaly detection is often limited by the availability of computational resources, which is scarce in many environments, such as wireless mesh or sensor networks [Hugelshofer et al., 2009]. DDoS detection is only currently carried out on the access network that is under attack, and new forms of attack that function at the application layer are hard to distinguish from normal events, requiring non real-time classification [Peng et al., 2007]. Volume anomaly detection in realtime is resource challenging, leading to different choices of sampling that can impact the accuracy of detection [Brauckhoff et al., 2010]. There are no techniques that can be applied to a variety of challenge types, which suggests a range of detection mechanisms need to be in place simultaneously. Classification of challenges in real-time is not yet feasible, leading to a trade-off between complexity and accuracy of identification [Peng et al., 2007]. Consequently, differentiating between attacks and normal traffic overloads, and thus identification of appropriate mitigation, is still difficult to achieve in real-time.

With the development of the internet, today any computer connects to the Internet is under threat of viruses, worms and attacks from hackers. Hansman et al. [2005] present a method for the categorization of both computer and network attacks. It consists of four dimensions which provide a holistic taxonomy to deal with inherent problems. Other proposed taxonomies include computer worms [Nicholas et al., 2003], computer intrusions [Daniel et al., 1998], and intrusion detection systems [Stefan et al., 2000]. They are all helpful for us to move the taxonomies towards a knowledge base, which could detect correlations and classify attacks, then assign the matched defense mechanism to the individual attack. The defense-centric taxonomy can be used to predict whether or not defenses could detect a particular attack based on its classification. [Killourhy, 2004]

The DoS attack is increasing rapidly these years, Hussain et al. [2003] introduce a framework for classifying DoS attacks based on header content, transient ramp-up behavior and spectral analysis. In addition to help understand attack dynamics, it is important for the development of realistic models of DoS traffic, it can be packaged as an automated tool to aid in rapid response to attacks, and can also be used to estimate the level of DoS activity. Mirkovic et al. [2004] further investigate two taxonomies for classifying DDoS attacks and defenses. The attack classification highlights commonalities and important features of attack strategies, which define challenges and dictate the design of countermeasures. The defense taxonomy classifies the existing DDoS defenses based on their design decisions. It then shows how these decisions dictate the advantages and deficiencies of proposed solutions. It is a first attempt to cut through the obscurity and structure the knowledge in this field. However this paper lacks of comparing strengths and weaknesses of each proposal, and discusses potential countermeasures against each defense mechanism. Peng et al. [2007] make up this by presenting a comprehensive survey of the causes for DoS attacks and the state-of-art mechanisms for defending against DoS attacks. They evaluate the implementation difficulties, challenges, strengths and limitations for different techniques. They also mentioned important steps to combat DoS attacks are increase the reliability of global network infrastructure and global cooperation, also conclude

by highlighting opportunities for an integrated solution to counter DDoS attacks.

As all the IDS we know of keep per-connection or per-flow state. Kompella et al. [2004] initiate research with the question whether one can detect attacks without keeping per-flow state. It shows that such aggregation will cause problems as behavioral aliasing and spoofing. The stealthy port-scanning cannot be scalably detected without keeping per-flow state while several categories of DoS attacks can. And their proposed Partial Completion Filters (PCFs) can detect both scanning attacks and partial completion attacks even with small traffic. The previous network monitor part mentioned Jackson et al. [2007] placing monitors within ASes. Castelucio et al. [2009] build upon this and propose an overlay network that provides an IP-traceback scheme at AS level to deal with DDoS threat. It contrasts with previous works because it does not require a priori knowledge of the network topology and allows single-packet traceback and incremental deployment. They propose a new extension to the BGP update-message community attribute that enables information to be passed across AS, also highlight a new sequence-marking process to remove ambiguities in the traceback path. The future work need to investigate the feasibility of integrating this system with a router-level traceback system. Therefore, perform the traceback at two levels: 1st, the traceback can discover ASes from where packets are sent and 2nd, the traceback can be performed inside these ASes, increasing the chance of getting closer to the attacker sources so that perform more efficient filtering.

Aamir et al. [2013] draw the conclusion based on the survey of the different techniques of DDoS, infrastructure layer (Layer 3) attacks is more favored by the attacker than application layer (Layer 7) attacks in overall scenario. Major DDoS attacks according to the statistics are SYN flood attacks (Layer 3), GET floods (Layer 7) and ICMP/UDP flood attacks (Layer 3). Some previous techniques are focus on the network layer and application layer. And special methods are developed for the application layer DDoS attack, for example, the CAPTHA puzzle images could be applied to clients to evade machine based automated DDoS attacks. But the application layer DDoS attack is identified as the future challenge as it is difficult to capture and may not be volume based.

2.2.2 Malware

Worms are still a widespread and persistent form of malware, attacking targets via various channels and strategies. Historically, most worms have propagated by scanning random IP addresses to infect vulnerable hosts. The worm will send an infectious payload to the target port of hosts running the desired services. In 2001, the code red worm infected 360,000 hosts in 10 hours [Yu et al., 2006]. In 2003, the SQL Sapphire Slammer [Moore et al., 2003] worm infected tens of thousands hosts in less than half an hour. In 2004, Cabir appeared as the earliest known worm targeting mobile phones. Now, worms infect smart phones via the Internet, Storage Cards, SMS, MMS, and even Bluetooth. Spoofed SMS messages are used to steal personal bank information by the ZeuS MitMo [La Polla, 2012]. In May 2012, trend micro reported that the "WORM_STEKCT.EVL_ worm spreads via facebook and instant messengers [Pantanilla et al., 2012]. Also in May 2012, Iranian government computers were attacked by Flame, which is complex modular malware that includes worm-like features. It is said to have a "close relation" with the previous Stuxnet and Duqu worms [Flame, 2012].

Port scans and worms are a major attack vector for infection and propagation by botnets. Thus worm attack detection and remediation remain as serious issues, as is the ability to adapt strategies in the face of evolving malware. There have been a number of surveys of worm detection [Li et al., 2008]. The focus is generally on detecting worm attacks during the propagation phase. The most common approach is based on signatures, which monitor the payload of packets to detect sequences of known worms. Examples of signature-based detection include snort and bro, sonicwall IPS, and checkpoint IPS. An alternative approach is to monitor protocol sequences of known worms, such as unsuccessful TCP connection attempts. However, signature-based detection suffers from scaling problems. As the number of different worm types grows, so too do the signature databases that are referenced by detection systems, increasing the overhead of real-time detection. Furthermore, in certain resource-constrained environments, such as wireless mesh networks, conventional signature detection systems are not feasible due to their overheads [Hugelshofer, 2009].

Signature-based detection is limited to detecting known worms. It cannot detect unknown attacks. Anomaly-based detection observes network traffic looking for abnormal behavior and generates alarms for anomalies. The challenges lie in distinguishing normal and anomalous behavior, and the setting of thresholds to detect anomalies. If not designed carefully, such systems could generate many false alarms [Li et al., 2008]. There are also trade-offs between accuracy and complexity. Simple tools such as volume monitors will detect large-scale attacks, but may also generate alarms for normal traffic. Alternatively, more sophisticated and accurate detectors may incur unreasonable overheads when operating in real time.

Previous work has also addressed worm containment. In [Chen et al., 2010] an approach to slowing worm propagation is proposed. However, it would be desirable to achieve complete elimination once a particular attack is identified. Another method of containment is address blocking. Once a host is identified as a scanner, all traffic from this host will be throttled [Chen et al., 2007]. This approach does not scale well to large-scale attacks from multiple sources.

Camouflaging worms (C-Worms) is difficult to capture since it could hide itself by blending with the environment in such a way that it looks analogous to the normal data packet of the network. Jeevaakatiravan et al. [2013] proposed a novel approach for detecting C-Worms. The Centralized Worm Detector (CWD) algorithm is deployed based on digital signature technique to authenticate each node and monitor the network. With CWD algorithm, the malicious C-Worm nodes are discarded and the recovered network is monitored.

In the work reported here we overcome the limitations of existing techniques by progressively monitoring a range of traffic features. Worm type anomalies can be identified in real time at an early stage using incomplete information. Interim remediation strategies can be applied until full or partial identification of the worm type is concluded. Furthermore, we offer a platform that enables ongoing development and refinement of malware detection and remediation strategies.

2.2.3 Network faults

The growth of the network also promotes the possibility of the various fault occurrences in the network. The fault could degrade the

network performance by affecting the network reliability, latency, throughput, delay, service level and dependability. Algirdas et al. [2004] give the main definitions relating to dependability, address the threats to dependability and security (faults, errors, failures), their attributes (reliability, availability, safety, integrity, maintainability), and the means for their achievement (fault prevention, fault tolerance, fault removal, fault forecasting). The fault-error-failure model is central to the understanding and mastering of the various threats that may affect a system. While, future research needs to analyze issues of trust and the allied topic of risk management, searching for unified measures of dependability and security. Also new technologies and concepts of man-machine systems will require continued attention to their specific dependability issues.

As fault management is being addressed as one of the major components of the network management suite. The generic network fault diagnosis system consists of four parts. Two network fault management approaches are reviewed by Lazar et al. [1992], which cover probabilistic approach and the finite state machine (FSM) approach. But their problems in general are very hard and many are NP-complete. However, it might be possible to find sub-optimal solutions that work well on average and on real scenarios. A more thorough analysis for modelling fault diagnosis is presented by Katzela et al. [1995]. It proposes a graph based network model suitable for alarm correlation and fault localization. They consider the dependencies among the different objects and a novel approach to estimate an alarm domain, and then design an algorithm for fault diagnosis. They also propose a fault diagnosis algorithm suitable for systems for which an independent failure assumption is valid.

On the other hand, the increased complexity of hardware and software resources and the asynchronous interaction among components make fault detection and recovery very challenging. Kim et al. [2008] present innovative concepts for fault detection, root cause analysis and self-healing architectures analyzing the duration of pattern transition sequences during an execution window. It implements root-cause analysis not only to detect the faults once they occur, but also to identify the source of fault allowing us to perform automatic fault recovery. Yu et al. [2008] demonstrate that the statement-based reduction provides much greater reduction of the test suites, saves testing expense, but negatively affects the

effectiveness of the fault localization. The vector-based reduction provides less reduction in test-suite size, reduces test expense, but provides negligible impact on the effectiveness of the fault localization. This study is helpful in addressing the testing and debugging costs during the software development process.

As previous mentioned, Katzela et al. [1995] introduced fault localization from model-based systems. But it has two main drawbacks: 1st, they require accurate dependency information amongst network entities, which is usually not available for large enterprise networks; 2nd, fault inference involves complicated computation and scales poorly with network size and complexity. In contrast, Ting et al. [2009] present an approach only requires elementary topological information and fault signatures to support matching over high-volume event data. It proposes a framework Meta (Monitoring network Events with Topology Assistance). They introduce a new class of indexable fault signatures that encode temporal evolution of events generated by a network fault as well as topological relationships, also present an efficient learning algorithm to extract such fault signatures. However when testing the accuracy of fault diagnosis find a larger D provides more leeway in identifying the fault node, especially test recall, this may need to improve in future. So the further research will cover 1) incorporation of domain knowledge in training fault signatures; 2) exploration of alternative models of temporal evolution, 3) search for data structures that can be incrementally adapted as network evolves; and 4) incorporation of a richer set of topological relationships derived from multi-layer networks.

Sethi et al. [2004] also mentioned the weakness with the divide and conquer algorithm [Katzela et al, 1995], which always explains all the observed alarms, but may fail to give their best explanation. It does not handle lost or spurious symptoms and may be used only if there are failure dependencies among objects. For the case of a system in which all objects fail independently of one another, a different algorithm was proposed but it has a polynomial worst case complexity. Moreover, Sethi et al. [2004] present a comprehensive survey of fault localization techniques and discuss their advantages and shortcomings. These techniques derive from different areas of computer science, including artificial intelligence, graph theory, neural networks, information theory, and automata theory, and

include model-based reasoning tools, model traversing techniques, case-based reasoning tools, graph-theoretic techniques, and the codebook approach. However, fault localization in complex systems remains an open research problem. The most challenging issues concern multi-layer fault localization, distributed diagnosis, temporal correlation, fault localization in mobile ad hoc networks, and root cause analysis in a service-oriented environment.

In hierarchical network, any fault could possibly bring down the network. Bhagvan et al. [2013] propose Fault Tolerable hierarchical Network (FTN) approach as a solution to the problems of hierarchical networks. The innovative method could identify possibilities of network failure and correspondingly provides specific recovery mechanism. Their experiment result also shows this method is better than the conventional approach over the network parameters: delay, throughput.

2.3 Commercialized Products

Most available commercial managed security solutions are primarily signature-based, in which network traffic is compared to predetermined attack patterns known as signatures [Cisco, 2011], [IBM, 2011], [Enterasys, 2011]. Typically, these systems offer automated intrusion response based on detected signatures only, and in many cases require a human operator to interpret anomalous behavior and take discretionary actions to mitigate it. None of these systems provide the same level of customization offered by the progressive multi-stage approach proposed in our work.

HP tipping point has the feature to automatically distribute digital vaccine when it is released, so that the new signature could be deployed to protect against the network threats. The threat management centre (TMC) researches and distributes digital vaccine periodically. HP TippingPoint application also gives the option for the network administrator to write custom filters for using on IPS and SMS devices. However the HP official released digital vaccine is not open source. So the network administrator must be very well trained to be able to customize the signature with stand-alone digital vaccine toolkit. HP is not responsible for the user customized signature, so the mistaken written signature may cause the false alarm when scan the network.

McAfee also has the NIPS, which help the network to defend against various malicious attacks. And McAfee Network Security Manager is used to view, configure, and manage Network Security Platform Sensor appliance deployments. The released signature set could be scheduled to automatically update and distribute on the sensors. Except the regular signature set, the incremental emergency signature sets that include attack signatures not yet available in regular signature sets. The incremental emergency signature sets are meant to address late‐breaking attacks that may need to be addressed immediately. The emergency signature sets could only add the new signatures, but don't contain the full set of signature set. However if the new attack hasn't been covered by the emergency or regular signature set, then it will bring the risk to the network.

Traditionally, system administrators have had the perception (which is not entirely wrong) that the automatic launch of remedies might create additional security risks. One of the main advantages of the approach we propose over commercial solutions is that policies can be carefully crafted and evaluated on the simulation environment, and manually modified by a human operator if needed. Also, even after policies are deployed, they can be modified to cater for specific contexts and operational events. Moreover, the multistage approach also permits introducing intermediate stages of remediation, which might not be optimal by themselves, but will allow the system to operate with limited performance until the definite root cause is reliably identified (either automatically or with some level of human involvement).

2.4 Evaluation

2.4.1 Detection Rate and False Alarm

In network monitor, Jackson et al. [2007] compare the percent of flows detected then prove that on the region that maximizes the cost effectiveness breadth-first outperforms depth-first. To investigate the contribution of different endsystems to the overall traffic, Cook et al. [2006] analyzed the fraction of traffic observed by endsystems monitoring as increasing subsets of endsystems are instrumented. It shows that instrumenting just 1% of endsystems was enough to

monitor 99.999% bytes. In the service level monitoring, Wuhib et al. [2006] determine the correctness of the threshold detection protocol by measuring the fraction of false negatives/false positives vs. actually occurred threshold crossings. Ting et al. [2009] measured the performance of Meta and the baseline approach in detecting network faults under varying configuration of fault occurrence rate and topological correlations.

In fault detection, Lazar et al. [2004] mentioned, the choice of the filter in FSM involves a trade-off between the fault detection rate and the number of different events passed through. The fault detection rate is very important for the network fault detection. Kim et al. [2008] evaluate the anomaly based fault detection approach capabilities. Then show the detection rate and missed false alarm for all scenarios they did with different number of grams. Therefore prove that their approach is superior in various scenarios.

According to Lakhina et al. [2004], the validation approach for anomalies detection is centered on answering questions: (1) How well can the method diagnose actual anomalies observed in real data? (2) How does the time and location of the anomaly affect performance of the method? In each case, detection success is measured by two metrics. One is the detection rate. The other is false alarm rate, which is the fraction of normal measurements that trigger an erroneous detection. Soule et al. [2007] proved traffic matrix is the optimal way to aggregate the traffic by comparing statistical anomalies, true or false positive rate with others. A false positive is defined as normal traffic that has been detected as an attack and a false negative as an attack which has not been detected. [Peng et al. 2007] Ringberg et al. [2007] evaluated PCA's sensitivity by comparing the false positive and detection rate in different networks with three aggregation formalisms. To evaluate the efficacy of PCF in scalable behavior based detection, Kompella et al. [2004] monitor the detection rate, false positives and false negatives in every time bin. A similar observation made by Hussain et al. [2003]. In addition, Peng et al. [2007] test the model accuracy of detecting reflector attacks by comparing the false alarm rate for different broadcast thresholds. To prove that the aggregated anomaly metric performs better than any individual algorithm, Shanbhag et al. [2009] compare the receiver operating characteristic (ROC) curves for the algorithms and the PAD system. A ROC curve

is a graphical plot of the true positive rate vs. the false positive rate for a binary classifier as its threshold is varied. In the anomaly detection survey, Chandola et al. [2009] mentioned in IDS, the issue arises with the large sized input is the false alarm rate. Peng et al. [2009] also demonstrated two measures for DoS attack detection are detection time and false positive rate. A good detection technique should have a short detection time and low false positive rate.

2.4.2 Latency and Efficiency

In the service level monitoring, except false alarm, another way Wuhib et al. [2006] assess the quality of threshold detection by measuring the delay between the time a crossing is reported by the protocol and the time of its actual occurrence. Ting et al. [2009] measure the scalability of the fault diagnosis in Meta by evaluating the average processing time of each incoming event, under varying condition of fault occurrence rate, with and without the multi-folded optimizations introduced. Then prove the efficiency of the execution.

In schemes for fault detection, to evaluate importance with the information of the dependencies between the objects for the fault localization process, Katzela et al. [1995] demonstrate that the fraction of time the divide and conquer algorithm finds the optimum solution is higher than either the independent algorithm. So it's more efficient. They also observe the relative error rate and conclude the relative error of the divide algorithm is low especially in the case of the same independent probabilities of failure.

To evaluate MIND performance, Li et al. [2006] measure several aspects of: Insertion path length, data insertion latency, query cost, the query latency, the data and traffic distribution across nodes and links, then show that the MIND could provide low insertion and response time. Shanbhag et al. [2009] test system ability to identify anomalies in packet traces by replaying the traces shown. This ensures that the PAD system receives realistic traffic and the results are reproducible. Castelucio et al. [2009] analyze traceback efficiency according to the presence of the proposed AS-level IP-traceback system. Traceback efficiency is defined as how much of the attacker path is discovered. They did the simulation for AS network topologies containing different number of ASes. They also

evaluate efficiency with an increasing number of attacking sources in strategic placement and random placement.

2.4.3 Computation Complexity and Assumption Strength

When Avizienis et al. [2004] evaluate development failures, mentioned they are usually due to an underestimate of the complexity of the system to be developed. So complexity analysis is the key to verify the techniques. To design the fault localization framework, Katzela et al. [1995] demonstrated the algorithm should be simple, have low order polynomial computational complexity, provide quick and accurate response, and should be easily adapted to a distributed environment. According to soul et al. [2007], traffic matrix is the popular way to aggregate traffic in statistical anomaly detection. However the downside is the computation complexity. With DoS detection, all techniques are based on one or more assumptions. Peng et al. [2007] conclude that the efficacies of detection are evaluated with their assumption strength and technical complexity. They list all the assumptions and find most are not strong, since attackers can change their attack patterns to overthrow the assumption and evade detection.

2.4.4 Scalability and Robustness

In decentralized service-level monitoring, to make the TCA robust, TCA-GAP dynamically reconfigures the overlay after addition and removal of nodes, also after node and link failures [Wuhib, 2006]. In a topological analysis of monitor placement, Jackson et al. [2007] show the percentage of flows detected, for all the combinations of both inter-AS links and ASes instrumented they explored. The sequence, monotonically increasing from 1 to a maximum of 8683 to check the scalability as the number of ASes instrumented increase. Li et al. [2006] did a large-scale experiment to explore MIND's scalability by deploying the MIND prototype on 102 nodes. Then describe the insertion latency across the entire experiment. This deployment enabled them to fail individual nodes in a controlled fashion and observe the availability of data at various levels of replication. To make the information sharing model for distributed IDS scalable, Peng et al. [2007] use hierarchical architecture for wireless sensor network. It significantly reduces the number of broadcasts for a small increase in delay. Kim et al. [2008] classify

data based on the specification supplied with the multi-tier web benchmark by building four different scenarios. Then prove there's slight impact of bulk training data set with the anomaly based fault detection by showing the good result.

2.5 Summary of the State of the Art

In the distributed challenge detection field, there has been considerable research with network monitoring, anomalous detection, fault tolerance, DDoS attacks separately, here we reviewed the latest technologies in these areas, highlight the relationship, mentioned their strengths, weakness and future work. However so far - limited work has been done to systematically embed resilience into the future Internet or develop and experiment with novel, distributed mechanisms for monitoring resilience to detect challenges as they occur. So our research contributes towards the development of new algorithms and experimental systems to perform challenge detection and identification. The universally useful measurement aspects are reviewed. The future platform is flexible, that means few assumptions has been made about operating parameters and robustness, meaning that it could give satisfactory performance under variations in topology and traffic.

The distributed challenge detection project select particular network-level and service provision scenarios for deepening the mechanism-level analysis and carrying out experimental evaluation. The intention has been to develop as much as possible the different components of the resilience framework. Challenges and their impact assessment, metrics, policies, and cross-layer techniques, including monitoring, are the four main components of the proposed framework. In summary, the published state-of-the-art in challenge detection and classification varies in the resources that are required, the timeliness and accuracy of their operation, and the challenges they can effectively operate with. Therefore, it must be possible to flexibly organize these mechanisms and dynamically adapt their operation in a way that is sympathetic to their characteristics and the likely challenges that will occur. Moreover, configuring these mechanisms will be complex, especially when one considers their interaction with those to remediate challenges – an issue that existing work does not address.

Chapter 3

A Multi-Stage Policy-Driven Approach to Network Resilience

In this chapter, we will explain the motivation for the development of a multi-stage policy-driven approach to network resilience. Then we will show how this approach can progressively identify and mitigate challenges. Policy based management strategies will be introduced. Lastly, the benefits of this approach will be highlighted.

3.1. Motivation

A crucial part of a resilience strategy involves real-time detection of challenges, followed by identification to initiate appropriate remediation. We observe that the current state-of-the-art to challenge detection is insufficient. Our goal is to advocate a new autonomic, distributed challenge detection approach. The feasibility of our framework will be demonstrated through experimentation. We

conclude that our case studies provided valuable insight into resilient networks, which will be useful for further research.

To provide network resilience, a number of mechanisms are needed, such as monitoring systems, tools to collect IP flow information for use by intrusion detection and classification systems, and those to mitigate challenges. However, in complex networks comprising resilience mechanisms at different layers of the protocol stack and at disparate locations of the network, it is often difficult to define how these mechanisms should be coordinated. It is also difficult to define how the configuration of such mechanisms should change over time, in response to new types of challenges, context changes, or new requirements (e.g., new types of SLAs). Furthermore, configurations should be sympathetic to the points identified in the related work, namely the varying overheads, timescales, and accuracy of available detection and classification mechanisms. In this section, we elaborate on these concerns and their relationship to the mitigation of challenges, and discuss the motivation of using a policy-based management framework for defining configurations of resilience mechanisms.

3.2. Progressive Challenge Identification

Previous work has suggested that due to the overhead of capturing and inspecting packets it is not possible to employ widely-used intrusion detection systems on the access points of typical community-driven wireless mesh networks [Hugelshofer et al., 2009]. Furthermore, for a number of detection activities, increased traffic load compounds this problem. Network anomaly detection and classification systems can make use of flow information collected from vantage points in the network [Lakhina et al., 2005]. There is an overhead associated with collecting this flow information – a way of managing this overhead is to introduce sampling; however, as mentioned earlier, this may have an adverse effect on detection [Brauckhoff et al., 2010]. Again, increased load, e.g., caused by a DDoS attack, is likely to make the resources available for flow collection and detection scarce. Consequently, the effectiveness of detection and classification systems when the network is heavily utilized needs careful consideration, as this may have an impact on

their ability to provide results, and correspondingly a system's ability to perform appropriate mitigation of a challenge.

Closely related to the issue of the overhead of detection mechanisms is the timeliness and accuracy of the results they yield. This is important because of the necessity to remediate a challenge as early as possible, potentially minimizing disruption. Our assumption is that detection mechanisms that yield coarse-grain findings about the nature of a challenge, e.g., detect the presence of an anomaly, are more timely and have a lower overhead than those that provide more fine-grain information, e.g., give details of the nature of the anomaly. Furthermore, we assume that fine-grain information allows better decisions to be made regarding how to mitigate a challenge. We discuss this notion further, with the use of an example.

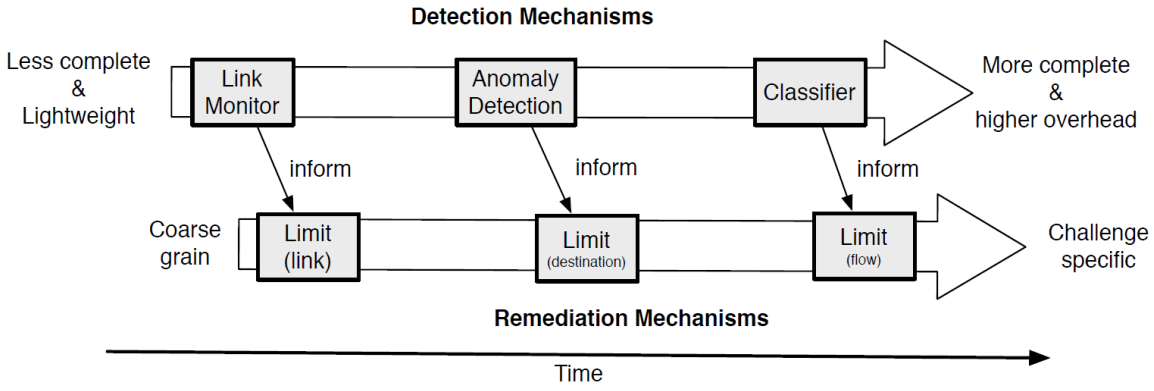


Fig. 4. Coarse to fine grain challenge identification and remediation

Consider the mechanisms depicted in Fig. 4 [Yu et al., 2011] that could be used to detect and remediate abnormal traffic loads. An increase in traffic volume may be the result of a DDoS attack, flash crowd or device misconfiguration. A simple link monitor can provide local information about an unusual increase in the volume of traffic observed at a router. If the traffic volume is disruptively high, traffic on a given link can be shaped (limited) in order to protect downstream infrastructure and traffic. The detection is relatively lightweight, fast and coarse grain; correspondingly, the remediation taken will affect both potentially malicious and benign traffic. A more sophisticated volume-based anomaly detection technique [Lakhina et al., 2004] may determine the specific destination address of the

increased traffic, and allow shaping of traffic to that destination only – this would reduce the overall amount of traffic being incorrectly penalized in the network.

Finally, a more heavyweight machine learning classifier has the ability to accurately discriminate traffic flows of malicious provenance from the normal traffic, and permits a more fine grain traffic shaping strategy to block attack flows alone. In summary, it can be seen that because of these concerns, the configuration of mechanisms that realize a resilience strategy needs to evolve at run-time, based on the challenge information that is currently available. In particular, taking into account the characteristics of detection and classification systems, and their interplay with mechanisms used to mitigate a challenge. We propose to build on the capabilities of a policy-based management framework to address this issue.

3.3. Policy-based Management Strategies

Our resilience strategy - is policy based. This builds on earlier work by [Schaeffer-Filho, 2011]. We reuse policies here for the development of network resilience strategies. The configuration of the network, and resilience mechanisms therein, that realizes a resilience strategy can be modified without interrupting service operation [Damianou et al., 2001]. This is a necessary feature for the realization of the progressive multi-stage resilience approach, as it allows the configuration of resilience mechanisms to adapt in response to the prevailing resources, timeliness and accuracy of detection and classification mechanisms.

Fig. 5 illustrates how we use policies to reconfigure the operation of resilience mechanisms during run-time. Events representing either the occurrence of challenges, e.g., a DDoS attack, or changes in context, e.g., resource availability, may be used as a trigger for the reconfiguration of the system. Reconfiguration strategies are represented as policies, which define how the operation of the several components in the network should be modified in response to pre-specified events. The policy interpreter evaluates the events and makes decisions subject to the current set of policies. Such decisions may concern the tuning of parameters of the mechanisms, the re-wiring of their interconnections, and also the dynamic enabling

or disabling of the mechanisms currently deployed in a particular strategy.

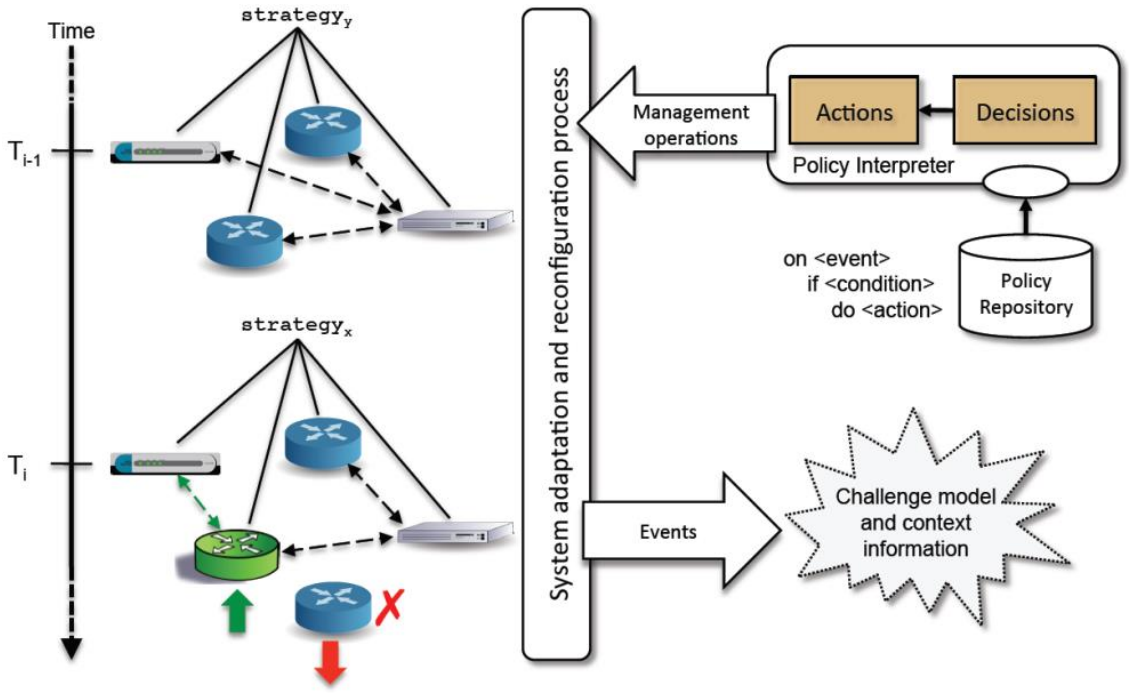


Fig. 5. Policy-based reconfiguration of mechanisms during run-time

At any time, the current set of mechanisms may generate new events which in turn may trigger a different set of policies. This continuous process will constitute a policy-driven feedback control-loop [Lupu et al., 2008], in which events trigger policies for the reconfiguration of resilience mechanisms, which may in turn generate other events that will trigger different policies and so on. Note that the set of policies defining the possible reconfiguration actions is not fixed, and different policies may be loaded or unloaded over time to reflect better resilience practices or a better understanding of the challenges the network is facing.

3.4. Benefits of the Approach

When we detect the onset of a challenge or measure the performance of the network in relation to resilience targets, adaption of the network could occur. These two factors lead to a set of coarse-grain and fine-grain adaptations. Coarse-grain adaptation involves the deployment of resilience patterns, which are

configurations of resilience mechanisms, capable of combating a specific challenge. A challenge analysis module provides information about the challenges that are affecting a network. This is used to select which resilience patterns should be invoked. Fine-grain adaptation involves setting or adjusting the parameters of the mechanisms that are currently deployed in the network as part of a pattern.

In our joint work within the ResumeNet project, a pattern is a policy configuration of resilience mechanisms and their relationships. Different challenge types will demand specific sets of mechanisms to monitor features in the network (e.g., current traffic load or alarms generated by an anomaly detection mechanism), and initiate remediation actions to combat anomalous behavior (e.g., blocking malicious flows or selectively dropping packets). This assumes the existence of autonomous mechanisms supporting a range of resilience functions in the network.

In our joint work, a framework and a process for the design and evaluation of network resilience management is presented [Schaeffer-Filho et al. 2012]. The framework enables (1) the offline evaluation of resilience strategies to combat several types of challenges, (2) the generalization of successful solutions into reusable patterns of mechanisms, and (3) the rapid deployment of appropriate patterns when challenges are observed at run-time.

The contribution of this thesis is a simulation platform to evaluate the performance of resilience strategies. The toolset is based on the integration between the OMNeT++ simulator and Ponder2 framework. It supports the simulation of a range of challenge scenarios and the resilience strategies used to combat these challenges. The toolset enables us to identify best practices and the most effective policy configurations for challenges such as DDoS attacks, flash crowds and worm propagations.

3.5. Summary

In this chapter, we have discussed the motivation of using a policy-based management framework for defining configurations of resilience mechanisms. We described a progressive challenge identification strategy, which means our initial detection is relatively

lightweight, fast and less complete; correspondingly, the remediation strategy is coarse grain. After further analysis, the root cause of the challenge could be identified and fine grain mitigation applied.

We propose that our resilience framework is flexible and reusable and can cater for similar challenges manifesting at different parts of the network, or variations of a specific form of attack. Our experimental work now explores this proposal.

Chapter 4

Comparison of Experimental Platforms

Building the distributed challenge detection system in the real world is not an easy task. Firstly, a large topology network is required to get the meaningful result. Secondly, such network needs large topology, which will increase the hardware cost and administration effort. Thirdly, with the current available real world platform, e.g. AARNET, PlanetLab, it is not convenient to experiment the attack on these networks. Because it's hardly to ensure the anomaly detection system will not disrupt the normal operational network, which will cause more severe effect. Fourthly, we need to have the full control with all the nodes in the system so that they could easily be configured to suit our case [Sethi, 2004]. However, this is also difficult to be realized in the real world.

The evaluation of large-scale challenges, such as DDoS attacks and worm propagation, is difficult because these activities are typically highly distributed in nature and disrupt normal network behavior. Consequently, resilience strategies to mitigate them can require the

coordination of various monitoring and control mechanisms across different administrative domains, protocol levels and heterogeneous infrastructures. The use of testbeds for evaluating network performance and protocol design can involve high costs of hardware and development effort [Smith et al., 2010]. Moreover, real testbeds are generally not suitable for the evaluation of large-scale challenges that tend to affect multiple autonomous systems.

As an alternative, to mitigate costs and address scaling issues associated with testbeds, we advocate the reproduction of network challenges and resilience mechanisms in a simulation environment. To this end, we have developed a toolset that couples policy-based management and network simulation.

In this chapter, we will first assess the possibility to build the experiment on offline IDS datasets and network simulation. The popular public available dataset DARPA and KDD Cup will be discussed. Following this, we will compare the network simulators, e.g. NS-2, NS-3. OMNeT, SSFNet. Then we will explain why we choose OMNeT as our testbed instead of others. At last, the simulated architecture and implementation will be demonstrated.

4.1 Off-line IDS datasets

4.1.1 DARPA

One possibility to carry our experiment is to use the off-line IDS datasets such as DARPA [Qiu et al., 2007], KDD Cup and etc, which could help us to relive from the above real world difficulties but with the realistic dataset. DARPA is collected between 1998 and 2000 from Lincoln Lab. The 1998 DARPA intrusion detection covers 38 attack types. The training data and test data slightly improved in the 1999, so that 201 instances of 56 types of attacks distributed. Whilst the 2000 DARPA Intrusion detection scenario specific data sets include two attack scenarios, also DDoS attack carried out.

4.1.2 KDD Cup

KDD Cup dataset gathered in 1999. It has 41 features and labeled as normal or an attack, with exactly one specific attack type. Based on our findings, we decide not to use them as well. First, the dataset are not up to date, so that the most recently attacks and the

unforeseen attacks couldn't be measured by these existing dataset. Secondly, Mahoney and Chan [Kotz, 2004] built a trivial IDS and ran it against the DARPA tcpdump data. They found numerous irregularities and demonstrate that the DARPA dataset was fundamentally broken. Thirdly, the performance measure applied in DARPA'98 Evaluation, ROC Curves, has been widely criticized [Fessi et al., 2009]. Lastly, it was still useful to evaluate the true positive performance. However, any false positive results were meaningless [Doerr et al., 2010].

4.2 Network Simulators

Network simulators could overcome all the mentioned hurdles and meet the needs by integrating real world applications. However, it still requires us to compare different simulators to recognize the most appropriate environment. So we surveyed the widely applied simulators, NS-2, NS-3, OMNeT, SSFNet, JiST/SWANS and J-Sim. NS-3 [Mahoney et al., 2003] is the latest updated platform comes from several different network simulators and developed from 2008. While SSFNet [Tavallaee et al., 2009], JiST/SWANS [Brugger et al., 2007] and J-Sim[Weingartner, 2009] nearly inactive since 2004, 2005, 2006 respectively. NS-3 and OMNeT are continuously supported today.

The policy-driven resilience simulator presented in our work is based on an integration between the OMNeT++ simulator [Varga et al., 2008] and the Ponder2 policy framework [Twidle et al., 2008]. The toolset allows the evaluation of resilience strategies consisting of instrumented mechanisms within the simulation, whose behavior can be adapted during run-time – e.g., setting flags, dropping connections, triggering or stopping monitoring sessions, etc. The next sections will present the main design decisions and requirements related to this toolset, as well as describe the architecture and the attacks and resilience mechanisms supported.

4.2.1 The requirement for network simulator

The toolset is based on the integration of a standard network simulator to a policy management framework. We have considered the use of the most popular general purpose network simulators, including NS-2 [NS-2 Website], NS-3 [NS-3 Website], OMNeT++

[OMNeT Website], SSFNet [SSFNet Website] and OPNET [OPNET Website]. The choice of a suitable platform was constrained by a number of requirements, namely:

Platform extensibility: the simulator must be extensible, not only in terms of protocols models, but also its ability to be instrumented to allow communication with the policy framework;

Availability of models: the availability of a large number of network models and protocol implementations is required to allow faster modelling of networks, and their resilience strategies;

Performance & scalability: the simulation platform must be scalable and present good performance to allow faster and larger simulations of realistic network topologies;

Modeling generality: the simulation environment should support the modeling of network components and protocols consisting of mechanisms for resilience that will reside at protocol layers 1–7.

In terms of platform extensibility, OPNET is the only commercial tool and the source code for its simulation kernel is not publicly available. Since we are required to extend the simulator to facilitate the communication with the policy framework, this is not a viable option. Most simulators offer an extensive library of network models, apart from NS-3, which is still a relatively new endeavor and whose models need to be ported from NS-2 manually, and SSFNet, whose development was discontinued in 2004 and the availability of new protocol models is now limited. Moreover, NS-2 has been consistently reported to offer limited scalability and performance [Cavin et al., 2002]. Lastly, all simulation environments considered are suitable for modeling general communication networks and protocols at different levels.

4.2.2 NS-2 and NS-3, SSFNet, JiST/SWANS and J-Sim

Our work needs large topology, but NS-2 has the scalability issue with memory usage and simulation run-time [Weingartner et al., 2009]. However, NS-3 and OMNeT are scalable. Considering the run time, JiST/SWANS are the fastest, whilst J-Sim is the slowest. OMNeT is slower than NS-3 but faster than NS-2 [Weingartner et al., 2009]. NS-3 has lowest computational and less memory demands

whereas JiST/SWANS exhaust memory [Kargl et al., 2007]. OMNeT consume more memory than NS-3 but less than NS-2. In terms of GUI, NS-3 and NS-2 are relying on source code but OMNeT has a rich GUI with online visualization. Below we list the similarities and differences between NS-2 and NS-3.

NS-3 Similarities to NS-2:

- C++ software core
- GNU GPLv2 licensing
- ported ns-2 models: random variables, error models, OLSR, Calendar Queue scheduler, (more planned)

NS-3 Differences with NS-2:

- Python scripting (or C++ programs) replaces OTcl
- most of the core rewritten
- also based on the yans and GTNetS simulators
- new animators, configuration tools, etc. are in work

We compare the different simulators and list their main differences as below:

NS-3:

- Ns-3 is not backward compatible, ns-2 scripts cannot be run in ns-3
- Reuse many models in ns-2
- Replacement for the popular ns-2
- Focus to date has been on setting the long-term architecture
- Trying to avoid some problems with ns-2, such as 1) interoperability & coupling between models, 2) lack of memory management, 3) debugging of split language objects

NS-2:

NS-2 is being only lightly maintained at present due to the development work on NS-3

- It is very hard to use when scenarios get bigger and exceed several hundred nodes, which is not the case for JiST/swans
- In the manet, consume more memory than swan

OMNeT:

- Lags behind the ns-2 simulator on availability of communication protocol models
- Simulation API is more mature and more powerful than ns-2's
- Simulation kernel can be embedded in other applications
- Ns-2 lacks many tools and infrastructure components that OMNeT++ provides: support for hierarchical models, a graphical
- Editor, separation of models from experiments, graphical analysis tools, simulation library features such as multiple rng streams with arbitrary mapping and result collection, seamlessly integrated parallel simulation support, etc.
- The ns-3 goals include some features (e.g. Parallel simulation, use of real-life protocol implementations as simulation models) that have already proven to be useful with OMNeT++
- Simulation results reporting is not adequate

SSFNet:

- DML lacks expressing power and features to scale up to support large model frameworks built from reusable components.
- SSFNet also lacks OMNeT++'s INI files, all parameters need to be given in the DML.

J-Sim:

- Similar to OMNeT++ in that simulation models are hierarchical and built from self-contained components, but the approach of assembling components into models is more like NS-2

- The use of TCL makes implementing graphical editors impossible.
- Model development and debugging can be significantly faster than C++, However, simulation performance is significantly weaker than with C++, and it is also not possible to reuse existing real-life protocol implementations written in C as simulation models.
- No independent (3rd party) simulation models
- Poor post-technical support
- Far-less pre-built modules and protocols than in other simulators
- Simulation result reporting is not adequate

OMNeT simulation API is more powerful than NS-2's. NS-2 is only lightly maintained now. NS-3 will eventually replace NS-2, but it is not backward compatible. It is trying to avoid some problems with NS-2. The NS-3 goals include some features (e.g. real-life protocol, parallel simulation) that have already proven to be useful with OMNeT. Moreover, the new animators, configuration tools and etc. are still in work. In addition, OMNeT offer basic modules, which is extensible using C++, whilst NS-2 is not easily modifiable. OMNeT provides well online documentation and active discussion forum. Overall, our preferred platform should be updated and reliable, so the flexible and generic simulator OMNeT well suits us.

Based on the comparison, OMNeT++ was chosen as the most suitable platform for the development of the resilience simulator. OMNeT++ is considered as one of the most widely used simulators for research in the area of communication networks [Gamer et al., 2009]. In our work, we present a description of the architecture and main components of our OMNeT++ based implementation, and a comprehensive case study and set of experiments that validate the use of the toolset for the evaluation of resilience strategies.

4.2.3 OMNeT

OMNeT is a public source C++ based object oriented discrete event simulator for modeling communication networks, multiprocessors

and other distributed or parallel systems [Varga, 2008]. It applies in diverse domains and written in two languages, NED designed for the network topology and C++ programmed for the modules. The compound module assembles from reusable simple modules. OMNeT utilize Tkenv as the GUI and it is easily debugging and trace. It could animate the flow of messages and present the node state changes in the network charts. Hierarchical module structure in OMNeT++ facilitates dealing with complexity. It could show packet transmissions while a simulation is running. The topologies of module connections are specified in the NED language. In addition, OMNeT++ sources are never patched by models. Simple modules are then reusable, and can be freely combined like LEGO blocks to create simulations [Varga, 2008].

Build on OMNeT, INET extends it by package of network protocols and offers objects, which combined with the channels to complete the network. Testing our system contains two steps, create various attacks and detect the anomalies. When consider the background traffic generation, IDS testing is classified into four categories. Compared to no background traffic, real or sanitized background traffic, testing by generating background traffic approach has benefits such as data freely distributed, no unknown attack and repeatable simulated traffic [Mell, 2003]. So ReaSE is chosen as our realistic background traffic and DDoS attack generator. It extends INET by server and client entities.

To build our network, firstly the realistic AS level topologies is generated to connect several separate administrative domains. Each AS is categorized as stub AS or transit AS. One transit AS is built to provide connections through itself to other networks. The stub AS is connected to only one other AS. This ensures each AS is accessible by crossing transit AS only. Two stub ASes and one transit AS are configured, named SAS1, SAS2 and TAS0. SAS1 connect to SAS2 through TAS0. Secondly, the router level topology within each AS is specified. Each AS has core, edge and gateway routers placed. The distinction between different routers is realized by allocating different bandwidth. Within the AS, it has total min 8 routers and max 15 routers. A few meshed core routers with low node degree that forward aggregated traffic of a high number of gateway routers with high node degree [Gamer et al. 2008]. Each edge router connects between 2 and 13 hosts to the network complete the hierarchical

topology. Therefore, each AS has different topology sizes and fills with nodes independently.

Thirdly, the network built with different traffic profiles to ensure the reasonable mixture of various protocols. The traffic profiles covers web, Interactive, mail, misc and ping traffic, which are based on transport protocols TCP, TCP, TCP, UDP and ICMP respectively. The router level topology's host systems are classified into clients and servers. Clients correspond to the ReaSE module InetUserHost, whilst servers represents by Web, Mail, and Interactive server. Fourthly, the bandwidth between different types of nodes are assigned from ReaSEGUI, also we configure the server fraction value, which specify the percentage of all router modules of each router-level topology are replaced by special server nodes. Overall, 136 hosts and servers are placed cross the AS in our network. At last, since ReaSE integrate the real attack tool tribe flood network to conduct the DDoS attack, so it is utilized to perform a random distribution by replacing randomly selected clients InetUserHost with DDoS zombies. The compound module DDoSZombie contains simple module TribeFloodNetwork with other INET modules that are essential to achieve the functionality of an attacking system [Gamer et al., 2008].

4.2.4 The simulated architecture and implementation

OMNeT++ is a general discrete event simulator that provides the basic machinery and tools to write simulations. However, in order to specifically support the modeling of communication networks, the INET framework[2] provides extension models for several wired and wireless networking protocols, including UDP, TCP, SCTP, IP, IPv6, Ethernet, PPP, 802.11, MPLS and OSPF. OMNeT++ consists of C++ modules that communicate via message passing. Messages are exchanged through input/output gates. Simple modules can be combined in hierarchies in order to build more complex components, called compound modules (e.g., mail servers, routers, etc). OMNeT++ also provides tools for designing network topologies (the NED language and editor) and supports plug-in extensions (e.g., a customized event scheduler).

The main motivation for the integration of a policy framework to a simulation environment was to enable the evaluation of the dynamic

reconfiguration of network mechanisms in a resilience strategy. Fundamentally, any simulation environment could be used to evaluate hard-wired resilience strategies only, whereas we required the evaluation of strategies that deploy and reconfigure resilience mechanisms on-demand, according to attacks or network conditions that are monitored dynamically.

To implement this dynamic behavior, our joint work chose Ponder2 due to the familiarity with this policy framework. Ponder2 implements a policy execution framework that supports the enforcement of both obligation and authorization policies. Ponder2 policies are written in terms of user-defined managed objects, e.g. adapters for interfacing with real network equipment. In our case, a managed object represents component that run in the simulation environment. In the following, we describe the choices available to realize the integration between the simulation environment with the policy-management framework.

1) Integration Techniques: Several techniques to allow the integration between a network simulator environment and external third party applications were discussed in [Mayer et al., 2008]:

Socket connection: proxies running within the simulation maintain socket connections to external applications. Sockets wait for connections and are responsible for delivering messages from the simulated components to the third party application, and vice-versa.

Source code integration: this method is straightforward for simple applications, which require that the third party application needs to be compiled with the simulation. However, this may be difficult for larger applications due to dependencies in the build environment.

Shared libraries: is based on the integration between the simulation tool and the binary code of the third party application. It is similar to source code integration but avoids problems related to the building process, because the build environments are kept separated.

The integration between OMNeT++ and Ponder2 is based on proxies, which is similar to the socket connection method. However, we are using XMLRPC3 proxy servers running within the simulation instead. Socket-based integration is suitable when the third party application does not need large volumes of data from lower layer

protocols [Mayer et al., 2008]. Instead, in our implementation, exchanges are limited to selected control events and corresponding management commands. This technique may, however, cause CPU scheduling and synchronization issues since simulations run faster and consume more CPU than applications running in real-time. We expect that these issues can be mitigated because, differently from [Mayer et al., 2008], we do not exchange packet-level information (large quantity, fast processing) with the policy framework.

2) Implementation: Instrumented mechanisms in the simulation environment implement an XMLRPC server through the MechanismExporter component. This component is used to register and export the management interfaces for the resilience mechanisms available in the simulation. A management interface provides callback functions to management operations that can be used to reconfigure a resilience mechanism, for example, to adjust the throttling rate of a rate limiter. For each type of mechanism, a ControlObject defines the management functionality to be exported via this management interface, and maps invocations to their respective method implementations on an InstrumentedComponent. This mapping relies on a table <*name, pointer*> that matches different invocations to the correct instance of a specific mechanism.

Whereas the components above implement an XMLRPC server for exporting the management functionality to the policy framework, a socket interface has been built to communicate and translate observed events from the simulation environment to the policy framework. Events are used to indicate conditions observed in the simulated network that may require management actions, such as the detection of an attack.

The EventPublisher component is responsible for establishing a connection with a Ponder2 instance and generates events of the form:

eventName?arg1 = val1; arg2 = val2; arg3 = val3; :::

These events are converted to a byte stream and sent via the socket connection to the Ponder2 instance. At the Ponder2 side, an EventBroker parses the byte stream received from OMNeT++ and maps it to Ponder2 events. A Ponder2 event may trigger one or more event-condition-action (ECA) policies, and the actions specified

by a policy define what resilience mechanisms executing within the simulation environment should be reconfigured and how. References to these mechanisms are obtained via the MechanismLookup component. When a Ponder2 policy is triggered, actions are invoked using the XMLRPC protocol for the respective mechanism, which is abstracted by an instance of the RemediationAdaptorRPC component.

Policies specify the required adaptations based on conditions observed during run-time operation of the network (as opposed to hardcoded protocols) [Schaeffer-Filho et al., 2011]. The integrated toolset allows us to analyze a range of wide-scale challenge scenarios and assess the effectiveness of a set of management policies controlling the operation of resilience mechanisms implemented as simulated components.

4.3 Summary

This chapter we discussed the experiment platform options, the off-line IDS dataset and popular network simulators has been considered. The merits and drawbacks among different network simulators, such as NS-2, NS-3, OMNeT, are also listed. Eventually OMNeT is chosen as the most suitable testbed. Then we proposed the simulation architecture and implementation.

To simulate various challenges, normally complex simulation scripts are needed to model the network protocols, topology, and the challenges. The challenges are separated into malicious and non-malicious challenges. The malicious challenge (e.g. DDoS attack) could be monitored by network monitoring models and detected by anomaly detection models. The DDoS attack resilience and malware resilience will be discussed in chapter 5 and chapter 6.

The challenges like operational mistakes, unintentional misconfiguration, accidental fiber cuts, and node failures could be grouped as non-malicious challenges. This type of challenge represents most adverse events observed in practice and could be simulated as random node and link failures [Cetinkaya et al., 2010], which cause the network failure. And this will be demonstrated in chapter 7.

Chapter 5

Volume Challenge Resilience

In this chapter, we show a policy driven network resilience platform to detect DDoS attack. This chapter is structured as follows: firstly, we outline our multi-stage policy-driven approach for network resilience. Then describes a case study based on DDoS attack resilience and also some initial results obtained using our simulation platform. After that, we present a discussion on the issues related to policy-based management. Follow this, the methodologies for differentiating DDoS attack and flash crowd have been introduced. Finally, we outline conclusions.

5.1. Strategy for DDoS Resilience

In the previous work, we found that the published state-of-the-art in challenge detection and classification varies in the resources that are required, the timeliness and accuracy of their operation, and the

challenges they can effectively operate with. For example, localized detection in fluctuations of traffic volumes can give a rapid and relatively lightweight indication of the onset of challenges, such as Distributed Denial of Service (DDoS) attacks or flash crowd events, whereas a sophisticated classification system can yield more accurate information about the challenge, e.g., the identification of malicious flows, over a longer period of time based on flow information collected from distributed network locations. However, these techniques are unlikely to be suitable for detecting and classifying other forms of challenging behaviors, such as software faults.

The most difficult task associated with our proposed multi-stage approach is to define how resilience mechanisms should be configured, and how those configurations should evolve over time in response to events, changing context and requirements. These changes should be achievable without interrupting the operation of the resilience mechanisms. To address this problem, we propose the use of a policy-based management framework, which decouples management strategies from the mechanisms that realize them.

As a proof-of-concept of our approach, we have developed a multi-stage resilience strategy to progressively ameliorate the effects of a DDoS attack on a Web service hosted on an ISP's infrastructure. Initial results, based on controlling resilience mechanisms implemented in OMNeT++ via policies, are presented and indicate the efficacy of our approach.

In order to provide protection to the access network of an Internet Service Provider (ISP), as shown in Fig. 6, from the effects of a resource starvation attack, such as a DDoS attack, targeted at a Web service hosted on the server farm. The figure shows the physical deployment of a number of resilience mechanisms. It can be seen in Fig. 6 that such an attack originating from the 10Gbps Hypernet toward a Web service hosted at the server farm has the potential to disrupt other hosted services (on the server farm), and the ISP's enterprise and domestic customers. Clearly, it is important that an attack be mitigated rapidly to reduce the impact to these customers, and protect the infrastructure, such as access routers, from crashing under exceptionally high load.

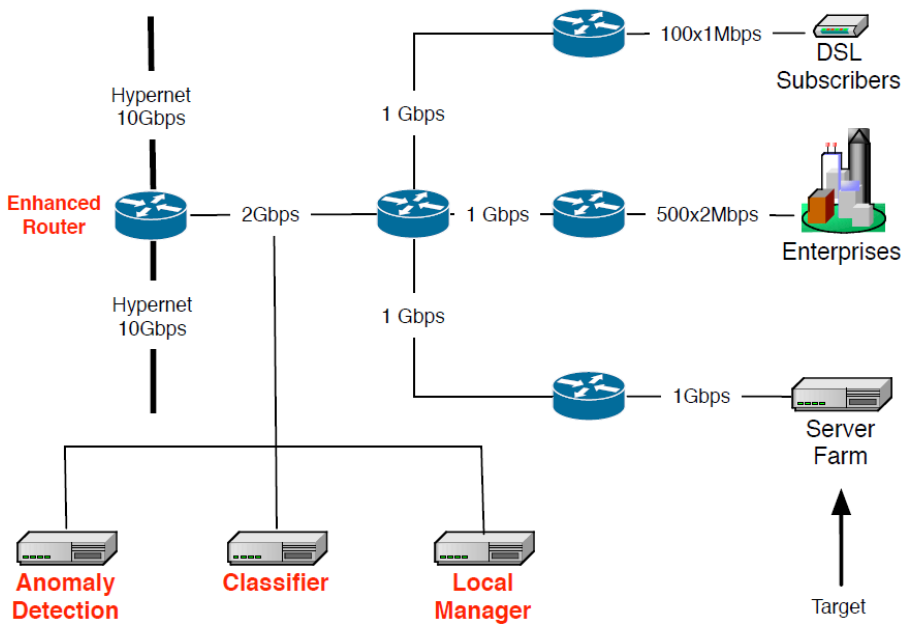


Fig. 6. The example topology used for our case study, showing the mechanisms used to ensure resilience of the network to high-traffic volume challenges

5.2. Experimentation

In what follows, we describe how policies are used to define the management aspects and interactions between the various resilience mechanisms shown in Fig. 7, to realize a resilience strategy for our case study. We then present a proof-of-concept implementation, via simulation, of the multistage policy-based approach. Our experiments make use of a package of OMNeT++ network simulation modules that implement the resilience mechanisms shown in Fig. 7. These are controlled by policies that are expressed using the Ponder2 framework [Twidle et al., 2008]. We describe how OMNeT++ and Ponder2 have been integrated and outline our initial results.

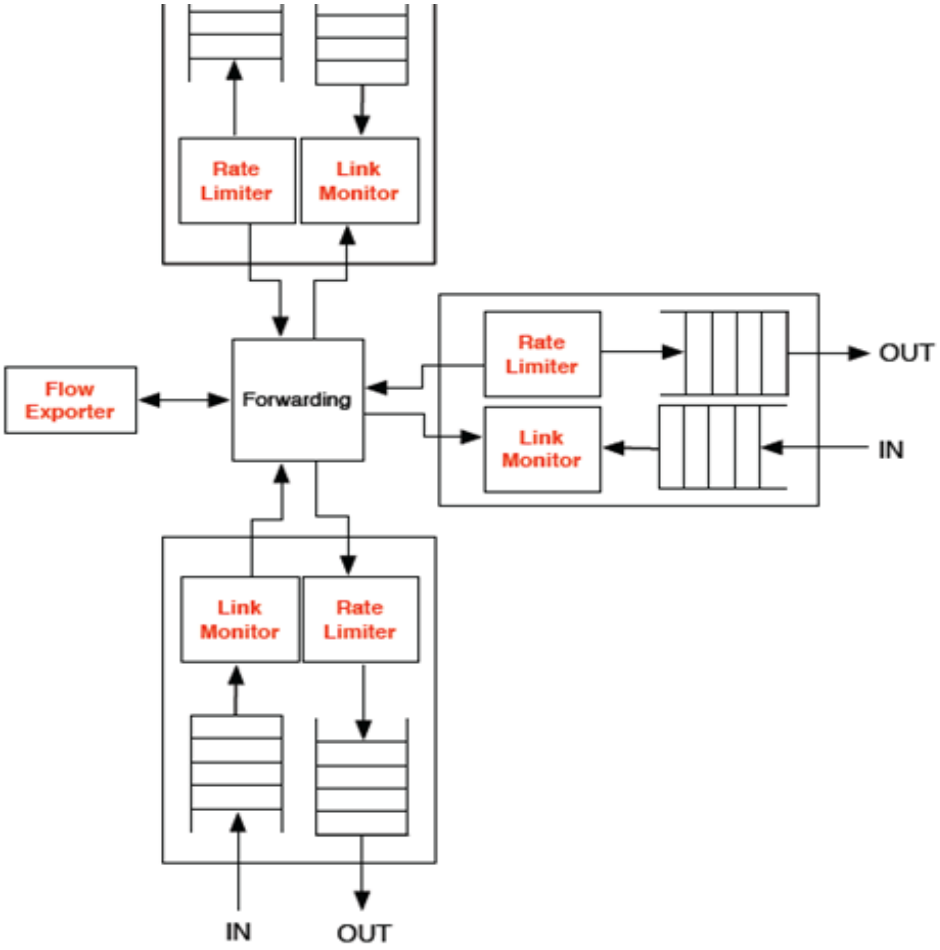


Fig. 7. A schematic representation of the enhanced router showing the resilience mechanisms used

5.2.1 Incremental DDoS Detection and Remediation

We propose an approach to DDoS attack resilience that incrementally improves remediation as more fine-grain information about the nature of the attack is gleaned from various detection and classification systems. Resilience mechanisms are realized as a number of policy-enabled *Managed Objects (MOs)*, shown in Fig. 8, that must co-operatively enforce the resilience of the network. A physical device, e.g., a router, will typically implement several logical managed objects, e.g., a link monitor and an IP flow exporter, as shown in Fig. 7. The algorithm for incremental challenge identification and remediation applied in this particular scenario is outlined in Fig. 8, and has the following steps:

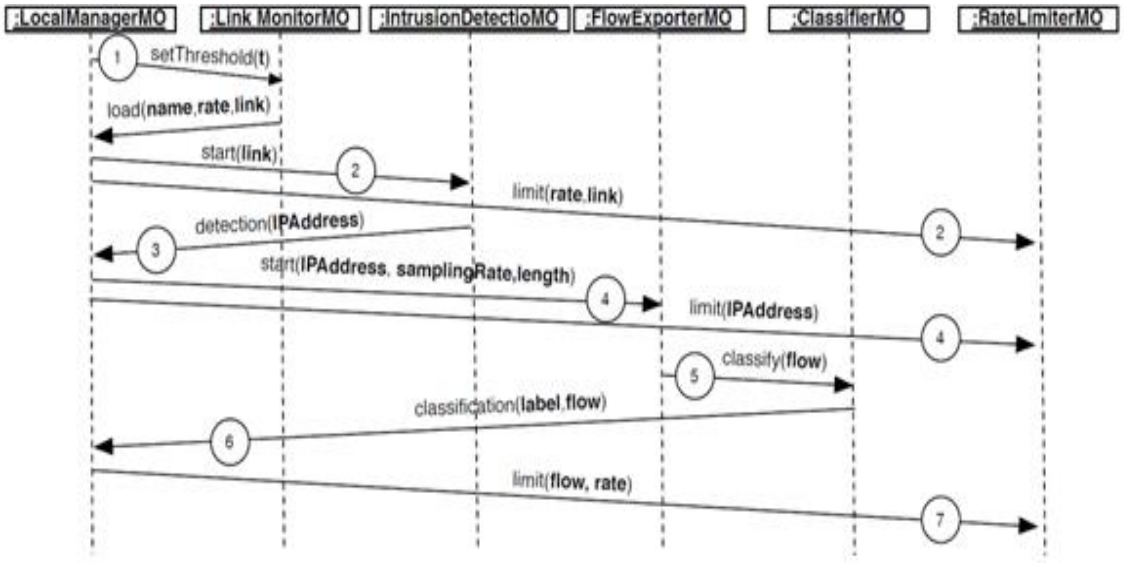


Fig. 8. Algorithm for incremental challenge identification and remediation

(1) LinkMonitorMO evaluates link utilization at a given periodicity, with its threshold rate being set by LocalManagerMO. LinkMonitorMO notifies LocalManagerMO of any sustained traffic rate above this threshold.

Distributed network monitoring is needed to detect coordinated attacks. The module we developed to perform the traffic monitor on the link is called LinkMonitor module. The monitor could continuously collect traffic information so that values display on the link in real time. Our development based on the OMNeT cDatarateChannel. cDatarateChannel and ThruputMeteringChannel are the name of the developed links in OMNeT simulation. After programming and debugging with the ThruputMeteringChannel, we further extend its function to monitor threshold. To gain a comprehensive overview of the link we monitor, the display could be customized with different attributes. Properties such as link color, propagation delay, data rate, can be assigned to connections. The characters could be monitored include packets number, current packet/sec, average packet/sec, current bandwidth, average bandwidth, channel utilization, average utilization, traffic volume, threshold. Channel utilization is the ratio of current traffic to the maximum traffic, which assists to understand the network performance and troubleshoot failures.

This module implemented as the channel so that offers the flexibility to collect information from any link within any network to gain the clear detailed view of its typical behaviour. The threshold monitor could notify that a certain parameter has exceeded a certain threshold and direct attention to those areas, so we could be immediately alert. It could effectively evaluate the network traffic to pinpoint the sub-network where victim located, and meanwhile prevent superfluous and incorrect alerting. The traffic values not only display on the link and shown in the module output in real time, but also recorded into the output vector file in OMNeT, which could be traced back to analyse why and when the anomalies behaviour happens. The output vector captures traffic over time. The collected historic data demonstrate the network behaviour in terms of performance and reliability. In addition, real-time statistics are important for detailed in-depth analysis. To ensure no false alarm created by the flash crowd also the alert could be raised immediately after the attack, we embed the timer function into the link monitor. The alert won't be generated unless the traffic above threshold for the continuous reasonable interval time. This method effectively avoids the events caused by the flash crowd.

(2) LocalManagerMO configures components in the subnetwork accordingly: on high link utilization, RateLimiterMO is notified to start limiting all ingress traffic on a link to a given rate. This is the first remedial action taken, which is coarse grained but reduces the overall impact of the attack. IntrusionDetectionMO is started to commence packet level analysis of the link traffic.

(3) IntrusionDetectionMO uses a threshold-based algorithm to count incoming packets on the link. It raises an event to LocalManagerMO when it determines the victim IP address.

The anomaly detection module has been developed for the evaluation of attack detection and traffic analysis. As the LinkMonitor offers the chance to get close to the victim by raising the alarm on the sub network where victim positioned, hence could perform efficient filtering. So the anomaly detection module use the simple algorithm to identify the victim, this effectively reduce the computation complexity and cost. The anomaly detection module implemented as the cSimpleModule and built into the INET compound module Router. In OMNeT, modules communicate by

messages, which contain usual attributes as timestamp and arbitrary data. The cPacket class extends cMessage with fields to represent network packets (frames, datagrams, transport packets etc.). Simple module sends message through output gate. The output gate and input gate linked by a channel. Therefore, the message travels through the channel and arrives at the input gate of another simple module. The Compound module consists of several simple modules and transparently relaying messages between their inside and the outside world. The Router includes the modules NetworkLayer, Routing Table etc. The anomaly detection module interconnected with NetworkLayer, TCP, and UDP by incoming and outgoing gates through channel. Every packet transferred into the router will pass the anomaly detection module for processing. The anomaly detection module will transparently process different network packets in a unified way. In addition, we use a hashing table to store the objects into the IDS table, and the table could iterate through. The IDS table could be monitored in real time as the module output, and the event be created immediately after identifying the destination IP address of the victim.

(4) When LocalManagerMO receives the victim IP from IntrusionDetectionMO it notifies RateLimiterMO to now limit traffic to the victim IP only, thus allowing traffic for other IP addresses to be forwarded as normal. However, this affects both malicious and non-malicious traffic towards the victim. FlowExporterMO is then enabled and starts recording IP flows.

(5) FlowExporterMO will truncate flow records after a specific time out period, e.g., 60s or 180s, and send records to the ClassifierMO with a given sampling rate, both of which are preconfigured by LocalManagerMO.

(6) ClassifierMO, through the use of one or more machine learning classification algorithms, eventually identifies the precise nature of the flow information it is being sent, e.g., they are benign or part of a TCP SYN attack, and LocalManagerMO is then notified.

(7) LocalManagerMO now notifies RateLimiterMO to limit only the attack flows, thus permitting non-malicious traffic to reach the intended destination.

As discussed earlier, we use policies to separate the management strategy, i.e., the behavior of the resilience mechanisms from their hard-wired implementation. For example, Fig. 9 illustrates the policy which implements step 4 of our algorithm, which specifies the limiting of traffic to the target and configuration of FlowExporterMO. Policies implementing each of the other steps of our algorithm were defined in a similar manner.

```
1  on IntrusionDetectionMO.detection(IPAddress,EAccessLink)
2      do
3      {
4          RateLimiterMO.limit(IPAddress);
5
6          FlowExporterMO.setTimeOut(EAccessLink,60s);
7          FlowExporterMO.setSampleRate(EAccessLink,75%);
8      }
```

Fig. 9. Policy configuring RateLimiterMO and FlowExporterMO, in response to a detection event generated by the IntrusionDetectionMO

5.2.2 Simulating Policy-driven Resilience Strategies

The notion of a policy-driven resilience simulator [Schaeffer-Filho et al., 2011] has been proposed based on the integration of a network simulator and a policy management framework. The toolset allows the evaluation of resilience strategies consisting of instrumented mechanisms within the simulation, whose behavior can be adapted during run-time – e.g., setting flags, dropping connections, triggering or stopping monitoring sessions. Our simulation environment has two key components: the OMNeT++ simulation environment [Gamer et al., 2009] and the Ponder2 policy framework [Twidle et al., 2011]. The integration coupling the tools is based on XMLRPC2.

The behavior of the policy-enabled mechanisms in our case study is implemented using Ponder2 polices. Ponder2 supports the enforcement of both obligation (event, condition, action) and authorization (access control) policies. Policies can be dynamically loaded, enabled, disabled and unloaded to change the behavior of managed objects without interrupting their functioning. Policies are written using PonderTalk, which is a high-level control language. Ponder2 enables the dynamic reconfiguration of the managed objects, resulting in the invocation of the appropriate management

actions specified by the policies. This realizes the transitions in Fig. 10. The managed objects were implemented as OMNeT++ modules, most as extensions of the standard Router module.

The configuration of these modules is shown in Fig. 10. FlowExporterMO and IntrusionDetectionMO are positioned above the network layer implementation, and receive duplicate packets from that layer. RateLimiterMO sits between the network and physical layers, and thus has access to every incoming and outgoing packet. Finally, LinkMonitorMO was created by modifying an existing channel type, thereby allowing us to place it at any position within our network. To simulate large-scale IP networks we use the ReaSE tool [Gamer et al., 2009], which permits the creation of realistic topologies and the generation of background and attack traffic. Of particular importance for our experiments, it can generate DDoS attack traffic based on the Tribe Flood Network [Dittrich, 1999].

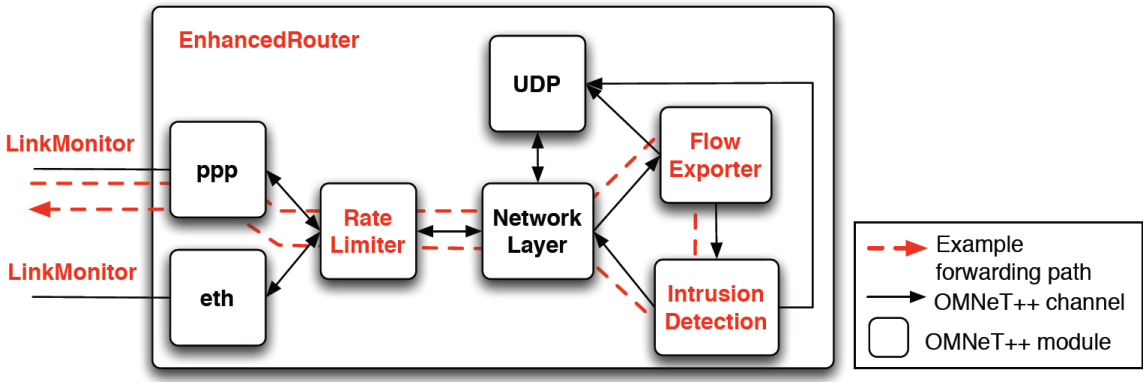
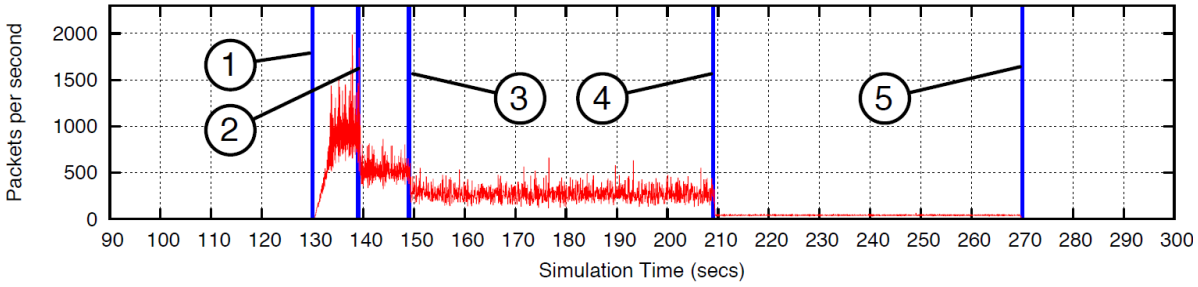


Fig. 10. Configuration of the OMNeT++ modules used to implement some of the managed objects. Channels to the UDP module from the FlowExporter and IDS modules enables communication within the simulator to other modules, such as the Classifier

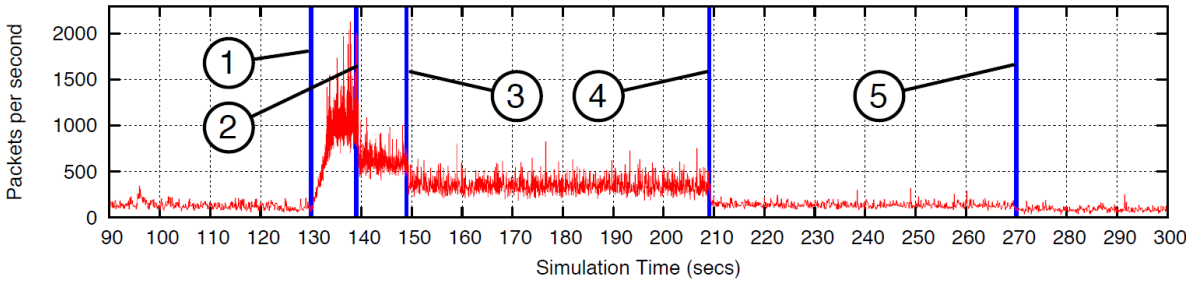
5.2.3 Results and Discussion

For our experiments, we simulated a network consisting of twenty Autonomous Systems (ASes): fourteen stub ASes connected by six transit ASes. A Web service in one of the stub ASes is configured as the victim to be attacked by thirty-nine DDoSZombie hosts across the network – this stub AS represents the network depicted in Fig.6. In addition, 1105 hosts generate background traffic to a number of other servers in the network. The various managed objects, are

activated on the ingress link from a core router to the gateway of the AS under attack. That is, the functions of monitoring, intrusion detection, rate limiting and so on are carried out at the edge of the AS network, in order to protect the AS's network. The following results show the execution of the algorithm in Fig. 8.



(a) DDoS traffic



(b) DDoS and benign traffic

Fig. 11.Initial results from simulations that implement the algorithm described. Numbered labels pertain to points in our multi-stage approach.

Fig. 11(a) shows the onset of the attack on the ingress link at approximately 130 seconds (1). The raising of an alarm by LinkMonitorMO is seen at 139 seconds (2), whereby a sustained traffic load in excess of the threshold defined in policies has been reached (currently, an increase in average incoming traffic of four times the previous average). Shortly thereafter, the effects of the initial rate limiting of the ingress link by RateLimiterMO can be observed. The filtering rate can be set by policies, as discussed earlier. In this case, we discard 70% of all incoming traffic in order to protect downstream servers and infrastructure. Results derived from the simulation show that 92% of blocked traffic during this period is malicious.

At 149 seconds (3), IntrusionDetectionMO identifies the destination IP address of the victim. This is achieved in this case by examining the destination address of each incoming packet, and raising an event when one destination accounts for 60% of all packets. RateLimiterMO is now reconfigured to drop 70% of the traffic destined for the victim only, as defined in policies. Some legitimate traffic that is not destined for the victim, which previously was blocked, is now not filtered. Results from simulation show that in this period (3-4), 95% of blocked traffic is malicious, while the proportion of legitimate traffic that is not blocked increases compared to the previous period (2-3).

Also, ClassifierMO is initiated at (3) and flow exporting from the router is started. Hereafter, ClassifierMO, receiving flow records from FlowExporterMO, attempts to identify the specific attack flows. At 209 seconds (4), rate limiting is confined just to the attack flow and legitimate traffic to the Web service can continue. After 270 seconds (5), all the malicious traffic is blocked, shown in Fig. 11(a), and the remaining traffic, shown in Fig. 11(b), pertains to normal background traffic. Fig. 12 shows how we refine mitigation by increasing the percentage of malicious traffic limited (red), and conversely decreasing the percentage of legitimate traffic limited (green).

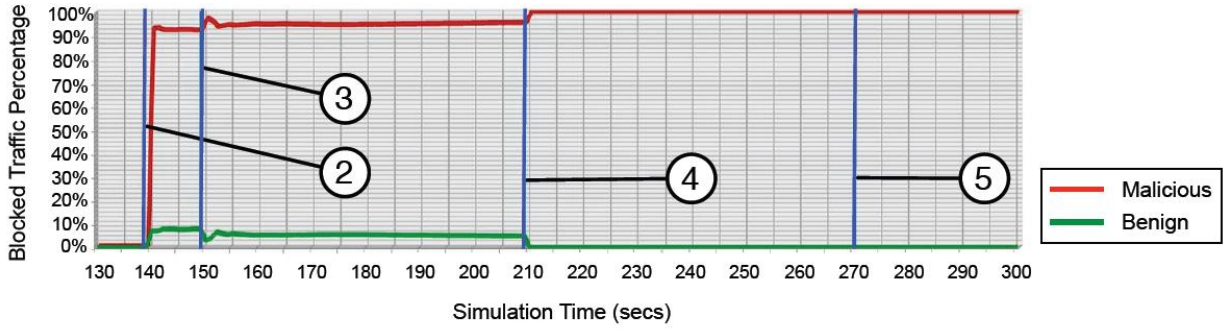


Fig. 12. The malicious and benign traffic dropped at different stages of the algorithm described in Section IV-A, shown as a percentage of the total dropped

These results provide a proof-of-concept demonstration of our approach. They show how a challenge (in this case a malicious attack) can be dealt with by initially using lightweight detection, and then progressively applying more heavyweight analysis to identify the specific challenge. In parallel, the results show the initial

application of coarse grain remediation, to minimize disruption to downstream services, which then moves towards more fine-grain, attack-specific remediation.

5.3 Discussion with flash crowd

In this section we discuss how our approach could be modified to deal with benign volume events. A DDoS attack and a flash crowd display very similar symptoms as the demand for a web site experiences a sudden increase. From the server side, it is really difficult to distinguish the DDoS attack and flash crowd event. If the flash crowd is mistakenly identified as a DDoS attack (which is the false positive), then all benign traffic to the web site will be denied, and genuine users will be blocked from accessing the website.. Also if the DDoS attack has mistakenly be detected as the flash crowd event (which is false negative), then the attacker could successfully crash the server, as well as consume the resources such as computing power and network bandwidth. So finding an efficient and accurate method to differentiate DDoS attacks and flash crowds is an important topic.

Previous research has considered this issue Rahmani et al. [2011] applied a joint-entropy scheme to detect DDoS. The DDoS attack could aggressively saturate the resources, so there is an unexpected disproportion between the received number of packets and the established number of connections. However, for a flash crowd, the increased number of packets received is always accompanied by an increased number of connection. The joint-entropy algorithm could effectively quantify the degree of disproportion to detect traffic anomalies. However, the authors mention that it is not easy to define a threshold to cope with both low rate and high rate attacks.

Prasad et al. [2012] proposed an information theoretic framework using BotNet on ITM (Internet Threat Monitoring) to model flooding attacks. They demonstrate an effective attack detection and trace-back using entropy by calculating the entropy variations between normal and attack traffic. They claim this methodology could effectively identify both low and high rate attacks. A flash crowd could be identified as a high rate attack whilst DDoS attacks could be detected as low rate attacks. Jeyanthi et al. [2011] also applied an entropy based approach to differentiate DDoS attack from flash

crowd in VOIP networks. In the paper, they list a comparison between DDoS attack and flash crowd, and find that a flash crowd is more responsive to traffic control - the traffic type mostly is web traffic and it is mostly predictable. However, a flooding attack could have any traffic type with unpredictable behaviors. The experiment has been validated by the simulation on the VOIP call processing server. The packets queued for the requesting service to the server appears to be high only for the short period whereas DDoS appears to be high for much longer. When a server suffers any overload, their proposed entropy based approach could distinguish between Benign and malicious events.

Graphic puzzles are widely used to differentiate between botnets and humans, so could possibly defend against flash crowds. But human response needs to be involved in this method, which is inconvenient to the users [Kandula et al., 2005]. Wenlei et al. [2012] demonstrate a discrimination algorithm using the flow correlation coefficient as a similarity metric among suspicious flows. They found the current flow of DDoS attacks are more similar compared to the flows of flash crowds. The flows of DDoS attacks have higher probability with similarity compare to flash crowds. They applied flow similarity features to detect flash crowds under typical botnet size and organisation. It is effective in dealing with unknown forthcoming flooding attacks. Experimentation has validated the approach using real datasets and real attack tools, but there is a need to evaluate cost and detection accuracy.

Thapngam et al. [2012] also realized that DDoS attacks have repeatable attack patterns, which are different from a flash crowd. Their proposed approach is based on Pearson’s correlation coefficient. It could extract repeatable patterns from DDoS attack but not flash crowd. Experimentation has been carried out with real datasets, but it would be interesting to see whether this could be validated in a real case in real time. Also the authors may want to improve the approach to detect faster reducing complexity and delay.

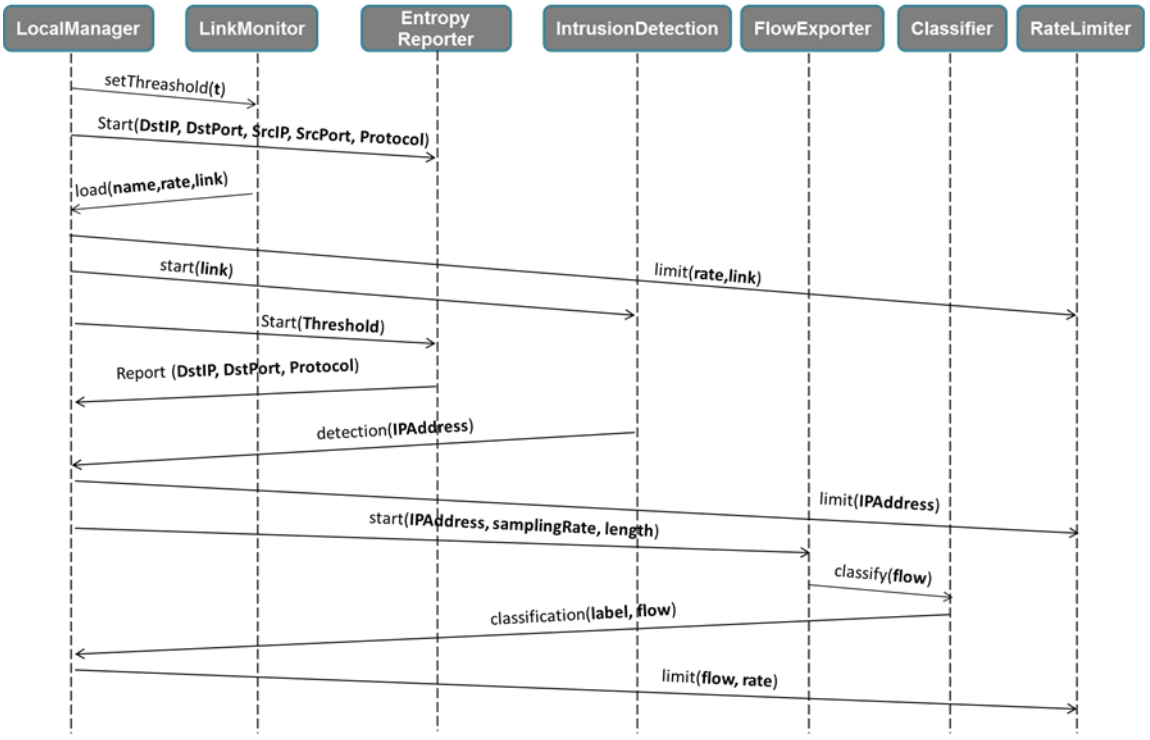


Fig. 13. Volume based challenges resilience strategy

Based on these findings we propose that we could extend our resilience strategy to identify a flash crowd. The current information theory based framework could effectively detect the DDoS attack. As introduced by Prasad et al. [2012], we develop an EntropyReporter Module (described in the next chapter) to calculate the entropy variations between normal and attack traffic. Thus a flash crowd and DDoS could be differentiated by the high or low rate attack. Figure 13 depicts and enhanced resilience strategy for volume based challenges. Compared to Figure 8, we add an entropy reporter as another managed object. After the dramatic traffic increase reported by the link monitor, the entropy reporter module could further classify it as a flash crowd or DDoS attack.

Figure 14 shows the entropy result for DDoS attack detection. With the simulated attack triggered at 130s, the entropy of the Destination IP and Destination Port turn concentrated, since the target is a single web server, whilst the entropy of Source IP and Source Port becomes dispersed. Once the entropy reporter confirms a potential

DDoS attack, then appropriate remediation strategies could be triggered.

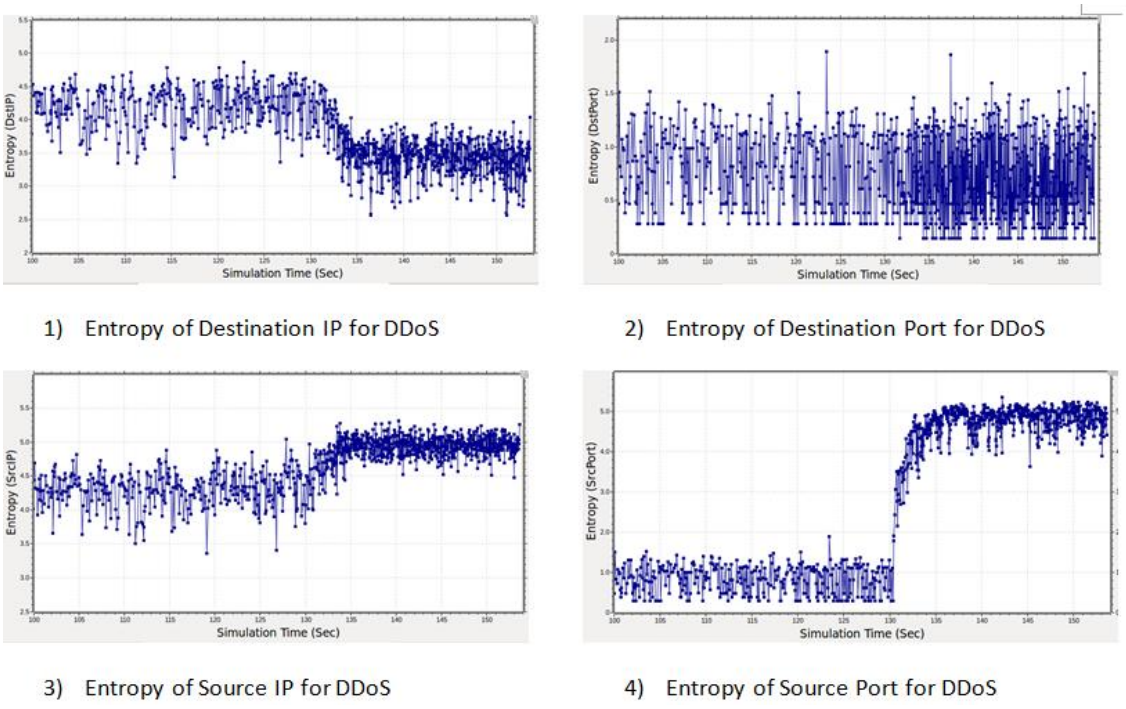


Fig. 14. Entropy result for DDoS attack detection

5.4 Summary

This chapter has presented an instance of our incremental policy-driven approach. Experimentation has demonstrated how this approach could effectively defend against volume base challenges, primarily a DDoS attack with possible extension to a flash crowd. The whole process is controlled by policies that rely on incomplete challenge and context information to progressively elaborate the configuration of the mechanisms currently deployed in the network. Our proof of concept case study, while simple, is a demonstration of the appropriateness and feasibility of our policy-based approach to challenge identification and remediation in realistic network settings. The main contribution of the approach presented in this chapter is to enable the mitigation of a challenge as early as possible, and refine the operation of resilience mechanisms as more information becomes available. We contrast this with a current, commercial solution.

Arbor is a popular DDoS protection, prevention and mitigation product used in the market, with privileged relationships with majority of world's ISPs. Peakflow is used as the flow based passive DDoS protection service. Arbor's Active Threat Feed (ATF) and Active Intelligence Feed (AIF) detects sophisticated attacks.

Arbor can detect three types of anomalies: misuse Anomalies are traffic of a certain type directed towards individual hosts that exceed what should normally be seen on a network; profiled anomalies means the threshold is automatically calculated based on the customer identified traffic performance for the last 30 days; fingerprints identifies traffic that matches a user specified signature.

A number of countermeasures are available via Arbor, such as invalid packet, IP address filter, zombie detection, TCP SYN authentication, TCP connection reset, traffic shaping, payload regular expression etc. These mechanisms can all be accommodated within our approach as Managed Objects and/or parameter settings.

With our staged approach, we attempt to eventually classify all the malicious traffic, so only the attack traffic is blocked in mitigation. Also, our aim is to automate the detection and mitigation stage with low false alarm rate and increased accuracy.

Our simulation platform can be used for further research into detection mechanisms, in order to understand how they may be combined algorithmically to analyze different challenges and the resource trade-offs involved. To explore the generality of our approach we have also implemented case studies for a variety of challenges including malware (e.g., worms and botnets) and non-malicious challenges (e.g., faults and router misconfigurations). These are described in the following chapters.

Chapter 6

Malware Attack Resilience

Network propagated malware such as worms are a potentially serious threat, since they can infect and damage a large number of vulnerable hosts at timescales in which human reaction is unlikely to be effective. Research on worm detection has produced many approaches to identifying them. A common approach is to identify a worm's signature.

However, as worms continue to evolve, this method is incapable of detecting and mitigating new worms in real time. In our work, we propose a novel resilience strategy for the detection and remediation of networked malware. Our strategy can be adapted to detect known attacks such as worms, and also to provide some level of remediation for new, unknown attacks. Advantages of our approach are demonstrated via simulation of various types of worm attack on an Autonomous System infrastructure. Our strategy is flexible and

adaptable, and we show how it can be extended to identify and remediate network challenges other than worms.

The remainder of this chapter is organized as follows: Section 6.1 provides an overview of background work on policy-based resilience management. Then outlines our multi-stage, policy-driven approach applied to the detection and remediation of worms. Section 6.2 presents results from the simulated deployment of worm resilience strategies for a number of known worm attacks, and also shows how the flexibility of our framework assists the evolution of resilience strategies to meet new challenges. Section 6.3 discuss and evaluate of our approach. Finally, Section 6.4 presents concluding remarks.

6.1 Strategy for Malware Attack Resilience

6.1.1. Strategy Overview

Worms represent a large class of networked malware. Although this type of malware has been studied for a number of years, according to recent security reports [PandLabs, 2011], worms constituted approximately 9% of the successful malware infections in 2011. Worms are self-replicating, self-propagating malware that can pose a serious threat to networks. Some worms can spread at great speed, infecting and potentially damaging a large number of hosts in a very short time, so that human reaction is unlikely to be effective. There is a need to develop new mechanisms capable of detecting and reacting to network propagated attacks in real time. Research on worm detection has produced many approaches to identify them. A common approach is to identify a worm's signature. However, in some network environments the achievement of real-time signature detection can be hampered by a lack of computational resources. Furthermore, as worms continue to evolve; this method is incapable of detecting and mitigating new worms in real time.

We characterize worm attacks, particularly in their propagation phase, as a type of network challenge. There are a number of mechanisms that can be used to meet our aim of network resilience. Detection mechanisms, such as link monitors and anomaly detectors, assist the identification and categorization of challenges such as

worms. Remediation mechanisms, such as rate limiters and firewalls, are used in the subsequent mitigation of these challenges. Recently, we have proposed a policy-based, multi-stage resilience approach [Yu et al., 2011], in which the configuration of detection and remediation mechanisms deployed in the network is dynamically refined as new information about challenges becomes available.

In this chapter, we use this approach to manage a range of resilience mechanisms to combat networked malware attacks such as worms. We use policies to control the operation of such mechanisms, and how they should be reconfigured in the face of different attack behaviors. Instead of relying on known payload attack signatures, which is the most widely deployed worm detection method, we show how our approach can embrace and adapt a range of detection and remediation mechanisms for both known and unknown attacks. In our case studies, changes in the distribution of specific traffic features are monitored, and a set of active policies determines how this information should be interpreted to contain worm propagation. Resilience strategies are evaluated using a policy-driven simulation environment [Schaeffer-Filho et al., 2011]. The primary contribution of this paper is a demonstration of the generality and benefits of our approach for dealing with an ever changing class of network challenge in the form of worms. We also show how our approach may easily be extended to deal with other forms of networked malware such as port scans.

6.1.2 Policy-driven Resilience Mechanisms

We rely on a policy-based approach to monitor and react to various network challenges. Through policies, we can decouple the “hard-wired” implementation of resilience mechanisms used to combat a specific attack from the run-time management system that defines their role in a resilience strategy. Consequently, resilience strategies can be adapted without interrupting service operation.

When defining a resilience strategy, there is trade-offs between the over-heads, timescales and accuracy of available mechanisms for challenge detection. Our assumption is that detection that yields coarse-grain findings, e.g. detect the presence of an anomaly, are more timely and have a lower overhead. Fine-grain information, e.g. details of the nature of the anomaly, allows better decisions

regarding how to mitigate a challenge, but this information is derived using mechanisms with a higher overhead. Consequently, we advocate and utilise a multi-stage resilience approach, which is based on the successive activation of mechanisms to analyse and remediate a challenge, from initial detection through to eventual identification. This approach is illustrated in Fig. 15.

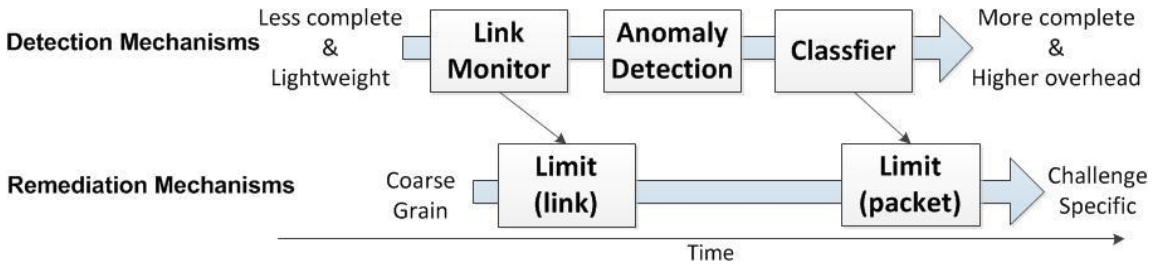


Fig. 15. Coarse to fine grain challenge identification and remediation

Initial detection is triggered by lightweight mechanisms, such as a link monitor. The detection is coarse grain. Then more sophisticated mechanisms are invoked. This approach allows the resilience mechanisms to adapt to the prevailing resources, timeliness and accuracy. This is necessary, as the simultaneous operation of a large number of challenge-specific detection techniques is too resource intensive. Through offline and/or online challenge analysis it is possible to collect network metrics and traffic information. For example, there is a significant amount of published information available that can assist understanding of (known) worms. For unknown challenges, online information needs to be gathered and analyzed. Ultimately, for each specific challenge, it is then possible to encode a complete resilience strategy into resilience patterns [Schaeffer-Filho et al., 2012], representing the policy-driven configurations of a set of mechanisms for combating that challenge.

In this section, we describe how the resilience approach can be applied to worm challenges. In our demonstration scenario the goal is to make an ISP or enterprise network - an Autonomous System (AS) - resilient to external worm attacks. It is a border protection strategy, requiring activation of resilience mechanisms at the ingress links to the AS.

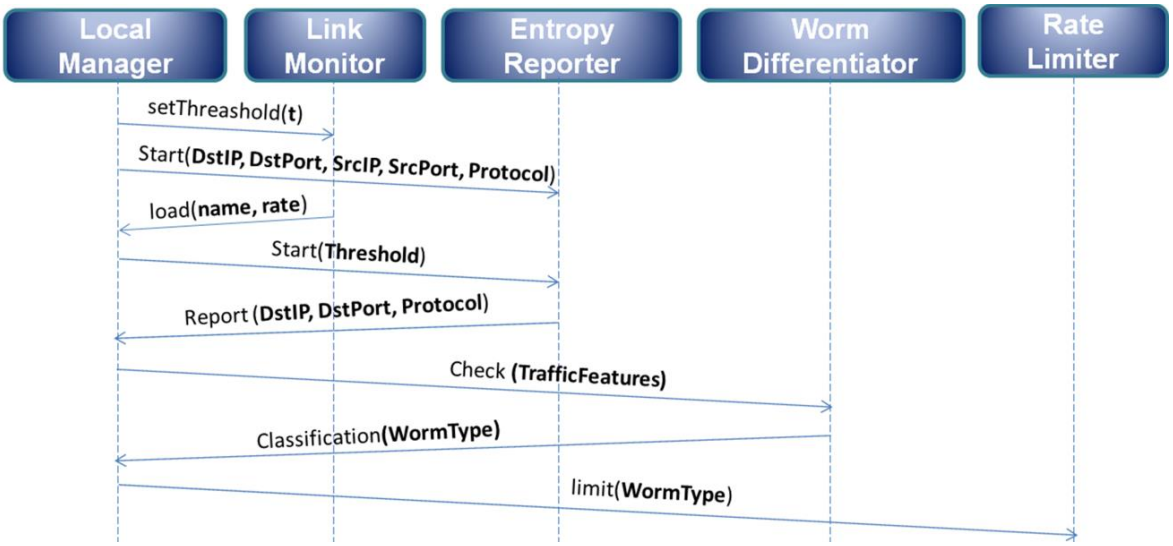


Fig. 16. Algorithm for incremental worm identification and remediation

While various mechanisms may be used to identify worm-related anomalies, we illustrate our approach using volume-based monitoring and techniques based on information theory. In particular, we quantify information using entropy [Shannon et al., 1948], which represents the uncertainty associated with the values of the different network traffic features observed. The entropy estimation for anomaly detection relies on the assumption that anomalies will disturb the distribution of certain traffic features in specific ways [Lakhina et al., 2005]. The strategy for incremental challenge identification and remediation is outlined in Fig.16.

An example configuration of some of these modules is shown in Fig. 17 to create an Enhanced Router that includes resilience functionality. This can be seen as a form of programmable router, capable of traffic monitoring as well as traffic shaping. In our implementation, the Entropy Reporter module is positioned above the network layer implementation, and receives packets from it. The Rate Limiter resides between the network and physical layers, and thus has access to every incoming and outgoing packet. As mentioned earlier, the Worm Differentiator can be used to identify known worms using entropy measures from the Entropy Reporter module. Finally, the extended channels that implement the Link Monitor can monitor both the traffic traversing the enhanced router via the PPP and Ethernet modules.

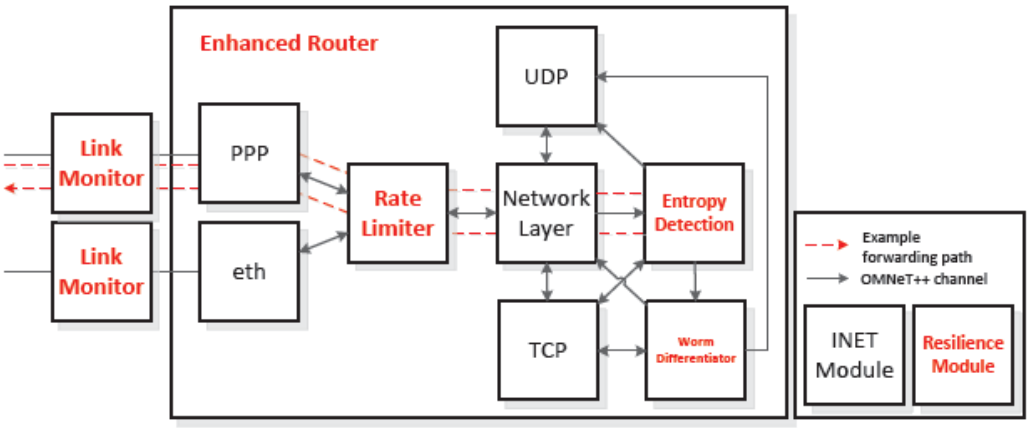


Fig. 17. Example configuration of OMNeT++ modules for network malware resilience

In the following we detail the operation of each of the resilience mechanisms, implemented as Managed Objects, which collectively realize our strategy.

Local Manager: the LocalManagerMO configures the other MOs to achieve the resilience strategy. These are event-condition-action (ECA) policies which specify activations and reconfigurations of the MOs. The policies are represented on the vertical axis of Fig. 17. At start-up the LocalManagerMO invokes two MOs. On each of the ingress links to the AS a LinkMonitorMO is activated to start monitoring link utilization, along with a threshold parameter. An EntropyReporterMO is also activated along with a list of features that it is to monitor.

Link Monitor: used for evaluating the link utilization at a given periodicity, with its threshold being set by LocalManagerMO. Slower propagating worms may escape detection by volume, which therefore require another form of early detection.

Entropy Reporter: the EntropyReporterMO continuously monitors the dispersion of traffic features using the computationally efficient Shannon entropy algorithm [Zesheng et al, 2009]. The features monitored are source IP, source port, destination IP, destination port and protocol. EntropyReporterMO recomputes entropy for the five features every 10 seconds and stores them in a vector. On notification of a volume event, LocalManagerMO sets threshold values to EntropyReporterMO, which then compares the entropy

history of each feature to see if a threshold change has been exceeded. When worms perturb the entropy values of several traffic features beyond the threshold, EntropyReporterMO generates an event.

We reference the entropy algorithm used in [Lakhina et al., 2005]. The entropy is used to capture the degree of dispersal or concentration of a distribution. An *e*mpirical histogram X = {n_i, i = 1, … , N } , means that feature i occurs n_i times. The entropy's definition is as below

$$H(X) = -\sum_{i=1}^{N} \left(\frac{n_i}{S}\right) \log_2(n_i/S)$$

Where S = $\sum_{i=1}^{N} n_i$ is the total number of observations. The same observation indicates that the distribution is maximally concentrated and the metric turns to 0. When the distribution is maximally dispersed, the entropy takes on the value $\log_2$ N.

Worm Differentiator: on notification from EntropyReporterMO the Local-ManagerMO activates the WormDifferentiatorMO along with details of the perturbed traffic features. The suspicious features are matched against a database of symptoms of known worms. These represent relatively short signatures of worms, compared to the payload-based signatures of existing detection systems. This reduces the costs of signature matching. If a match is found, an event is notified that identifies the type of worm along with a flow specification for the worm packets. However, if no match is found, for example it is a new type of worm, then alternate action needs to be taken (see discussion below).

Rate Limiter: on receiving notification from the WormDifferentiatorMO, the LocalManagerMO invokes the RateLimiterMO. This module can shape traffic according to parameters that specify the amount and types of packets to be discarded. It can limit a percentage of all packets on a link. Alternatively, it can limit at the flow level. In this case RateLimiterMO discards all packets that conform to the worm's characteristics, thus throttling all attack packets, without having to identify attacking sources.

6.1.3 Discussion

The policy-based strategy for worm resilience has advantages over traditional solutions. Firstly, through the use of an anomaly-based detection scheme we can potentially detect unknown (zero-day) worms. Secondly, through the use of policies we can quickly adapt how the system responds to attacks, since the policies are decoupled from the implementation of the managed objects and can be easily changed. Finally, the multi-stage approach introduces intermediate steps, specified by policies, when dealing with attacks. Thus, temporary forms of remediation may be put in place while the challenge is being processed, and until the root cause is reliably identified.

6.2. Experimentation

The evaluation of strategies for large-scale malware challenges is difficult. Resilience strategies often require the coordination of various monitoring and control mechanisms in different parts of the network. The use of testbeds can involve high costs of hardware and development effort, and are generally not suitable for large-scale challenges which can affect multiple autonomous systems. As an alternative, to mitigate costs and address scaling issues, we advocate the reproduction of network challenges and resilience mechanisms in a simulation environment.

Although simulators abstract some of the details of a real network deployment, our ambition is to develop high-level resilience strategies. These represent reusable configurations of resilience mechanisms that address common challenges, which are populated with mechanism instances and parameterized at deployment time. As such, we propose that simulation environment abstractions are tolerable. To evaluate the performance of resilience strategies we have developed a policy-driven resilience simulator [Schaeffer-Filho et al., 2012]. The toolset supports the simulation of a range of network challenges, such as DDoS attacks and worm propagations, and the implementation of our policy-driven approach to combat challenges. Resilience strategies for a particular challenge might use different combinations of mechanisms and policies, and the toolset enables offline evaluation of strategies.

In order to support the network resilience aim, we designed a resilience framework (illustrated in Fig. 18). For this experiment, we are using the same environment as described in the previous chapter. It allows the use of policies to define which mechanisms must be activated according to events observed. An event broker resolves event notifications in the simulation (e.g., anomaly detections, link load) to the policy framework. The policies repository will coordinate the transitions between the different stages of our process, based on the challenges observed and the mechanisms available. The policies incorporate coarse-grain and fine-grain policy repository. Coarse-grain policy repository involves the deployment of resilience patterns, which are configurations of resilience mechanisms, capable of combating a specific challenge. A challenge analysis module provides information about the challenges that are affecting a network. This is used to select which resilience patterns should be invoked. Fine-grain policy repository involves setting or adjusting the parameters of the mechanisms that are currently deployed as part of a pattern.

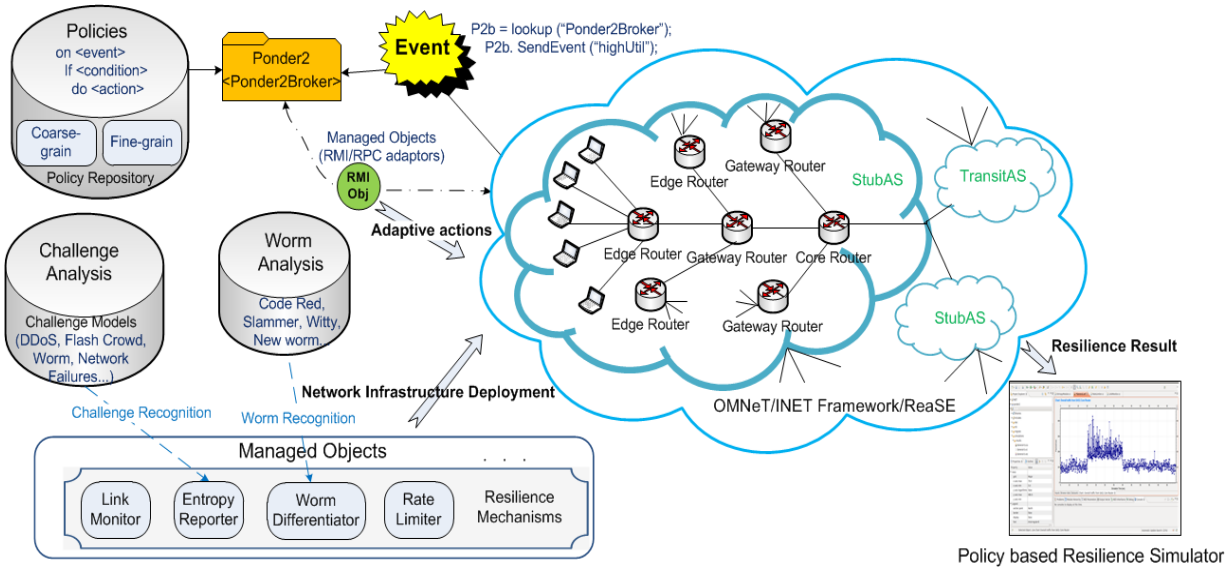


Fig. 18. Overview of the framework for worm resilience strategies

Resilience mechanisms are implemented as instrumented components. We have implemented the following mechanisms to achieve our worm resilience strategy. EntropyReporter receives duplicate packets. Based on the traffic feature monitored, it informs the type of challenge, for example, it's a DDoS or Worm.

WormDifferentiator analyses traffic features to report the type of worm (e.g. Code Red). RateLimiter is placed in-line between the network and physical layers. Finally, LinkMonitoris implemented by modifying an existing channel type, and can be placed at any position of the topology. Each instrumented object defines a management interface specifying which operations it supports. Management interfaces are used by Ponder2 for the invocation of operations on the simulated objects. Communication between Ponder2 and OMNeT++ is implemented using XMLRPC. In the remainder of this section, we evaluate a range of detection and remediation strategies using this toolset.

6.2.1. Worm Simulation

Our proposed worm resilient strategy detect various types of worms, to prove that, we did the experiment with three popular worms, code red worm, Slammer/Sapphire Worm and Witty worm. They all locate the target by blind scan, which is randomly generating the number as target IP addresses. In 2001, the well-known code red worm infected 360,000 hosts in 10 hours, it cause average 2.6 billion US dollar loss. On Jan 25th 2003, SQL Sapphire Slammer worm cause more than 1.2 billion US dollar economic damage. It could exploit the vulnerability with tens of thousands hosts in less than half an hour, where human interaction is not feasible. Its novel capability is the fast propagation speed. The full propagation speed is achieved in about 3 min (more than 55 million scans per second). Then growth rate slowed due to the worm saturate the bandwidth so that insufficient bandwidth to support more growth. Compare to code red worm, it may exploit fewer vulnerable hosts when slammer propagate speed is two orders of magnitude faster than code red. Witty worm spread in 2004, it was the first widely propagated worm result in information loss due to carried a destructive payload, attack at the integrity aspect. It targets the buffer overflow vulnerability in several Internet Security System (ISS) products. The details with these worms and the comparison among them are listed in table1.

	Code Red	Slammer/ Sapphire	Witty	New Worm
Target	Blind	Blind	Blind	Blind
Propagation	Self-carried	Self-carried	Botnet	Self-carried
Character	latency-limited	bandwidth-limited	bandwidth-limited	bandwidth-limited
Payload Size	4 Kbytes	Code size is 376 bytes, padded with its requisite headers, the payload is 404 UDP packet	Code size is 637 bytes, with data from system memory. Random UDP packet size ranging between 768 and 1307 bytes	Code size is 600 bytes
Spread Speed	Slower than slammer	Faster than code red	Faster than slammer	Faster than witty
Dst Port	port 80	port 1434	Random	Random
Src Port	Random	Random	port 4000	Random
Payload	Payload in monomorphic format and has a signature starting with "GET /default.ida?NNNNNN N."	No malicious content	Payload contains the text "(^. ^) insert witty message here (^.^),"	No malicious content
Behaviors	Spreads via many threads, each initiate connection by sending TCPSYN packet, then must wait for the target send respond SYN/ACK packet or timeout if no response. The thread is blocked during this time and cannot infect others. The response waiting occupy most of each thread's time	Overload the system and slow down traffic, exploit a buffer overflow in an MS SQL server	Target buffer overflow vulnerability in several Internet Security Systems (ISSs)	Exploit a buffer overflow and overload the system

Table 1. The Comparison between Different Worms

6.2.2. Identifying and Remediating Witty and CodeRed Worm

We use ReaSE to create topologies and generate synthetic traffic loads. The authors of this package have previously demonstrated that the simulated background traffic and the attack traffic are a realistic approximation of traffic observed in real networks [Gamer et al., 2008]. ReaSE can generate Code Red Worm propagation. We have extended the package to simulate the Witty and Slammer/Saphire worms, and a port scan.

We simulated a network consisting of 35 Autonomous Systems (ASes): 26 stub ASes connected by 9 transit ASes. 1616 hosts, 71 web servers and 21 interactive servers generate background traffic. Hosts across the network can be nominated to be zombies, which generate worm probing packets. Our strategy is simulated at a stub AS. The various resilience mechanisms are activated on the ingress link from a core router to the gateway router of this AS.

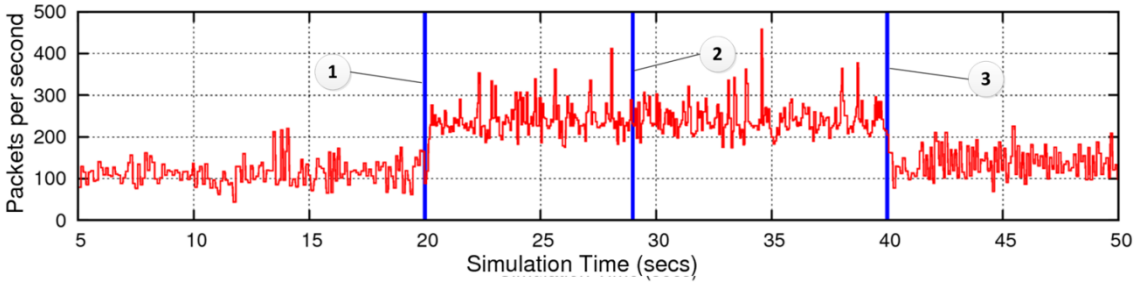


Fig. 19. Simulation results for the Worm resilience strategy

Fig. 19 shows the volume of traffic on the ingress link for a simulated, blind scanning Witty worm attack. In this scenario, the attackers sent a maximum of 8000 probing packets. At the start, LinkMonitor is activated with an alarm threshold set to an increase in average traffic on the link of twice the previous average, which is recomputed every 2500 packets. EntropyReporter is also activated to collect periodically (every ten seconds) packet-level entropy values on five traffic features: destination IP, destination port, source IP, source port and protocol. Fig. 18 shows the worm propagation starting at about 20s (1) Due to the high volume of traffic on the ingress link, an alarm is raised firstly by LinkMonitor at 29s (2) (This form of early detection is appropriate for high-volume attacks, but a different strategy would need to be deployed for low-volume attacks.)

Following this, EntropyReporter is interrogated for any significant changes in the five traffic features. The most recent entropy trend for each feature is computed and compared with the previous average entropy, with results showing that destination IP has become dispersed, destination port dispersed, source port centralised, and UDP protocol more dispersed. These results are reported back for further analysis (3). The WormDifferentiator is then invoked and can identify these changes in entropy as being a signature of the Witty worm, as described below. Consequently, at 40s (3) RateLimiter is configured to filter all probing packets, specified as all UDP packets with a source port 4000.

The features of the Witty worm are that it scans random IP addresses and destination ports using UDP, with a constant source port of 4000. Fig. 20 shows the changes in entropy of four of the features following the onset of the attack at 20s. They indicate characteristics of this specific worm attack. Furthermore, the separation between resilience policies and mechanisms permits easy adaptation of these parameters in an operational context, should conditions change.

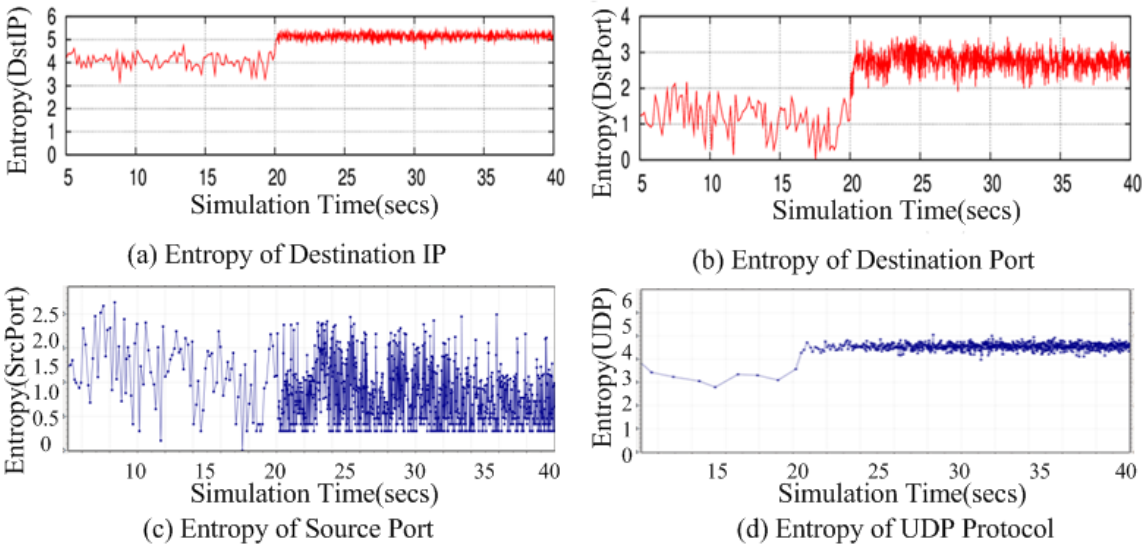
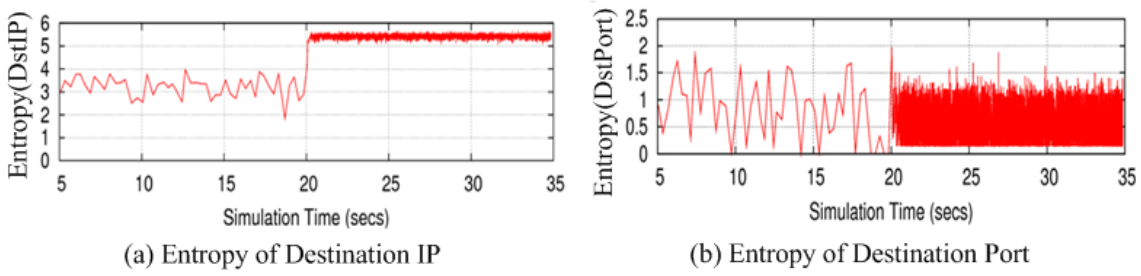


Fig. 20. Entropy changes with Witty worm

In contrast we also show partial results of a simulated Code Red attack. The scenario is identical to the Witty simulation above. Code Red also scans random IP addresses, but always to destination port

80. These features are indicated in Fig. 21 where, like Witty, the destination IP becomes more dispersed, but in this case the destination port becomes more concentrated.



(a) Entropy of Destination IP (b) Entropy of Destination Port

Fig. 21. Entropy changes with Code Red worm

6.2.3. Policy-based Adaptation of Strategies

Our simulation environment enables us to experiment with parameter settings and test the impact of adaptations of the strategies by modifying policies, which are executed by the LocalManager. As an example, worm attacks can generate high volumes of traffic which lead to denial of service. We may therefore want to consider adapting our strategy by introducing some early, interim remediation.

A simple adaptation would be for the LocalManager, at the first notification from the LinkMonitor at 29s, to also invoke the RateLimiter to shape all incoming packets. This coarse-grain remediation remains until the worm has been identified and attack specific remediation takes over. Fig. 22 shows the Witty attack scenario with such progressive remediation. Between 29s and 40s, 30% of network traffic on the link is blocked initially. At 40s, the WormDifferentiator matches the Witty signature then only the malicious traffic is blocked. Our platform provides results that show about 28% of benign traffic being blocked during the period of interim mitigation and 32% of malicious worm packets. We do not claim these to be ideal results but they are illustrative of our platform. Off-line risk analysis would determine if the costs (lost benign packets) are appropriate to the benefits (reduced flooding). Our platform provides inputs to such analysis, including the applications that are affected due to the blocking.

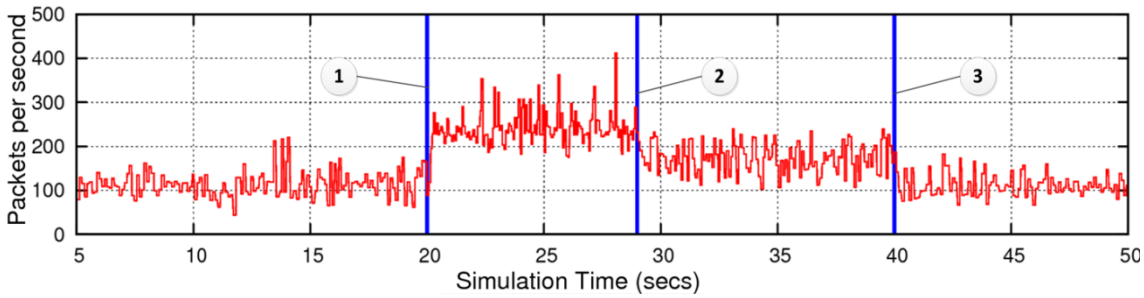


Fig. 22. Simulation results for the Worm resilience strategy with progress mitigation

6.2.4. Adapt the Resilience Strategy to Detect and Mitigate Slammer Worm

For this case study, we made use of our SQL Slammer worm implementation. As a means of infecting new hosts, this malware continuously sends 404 byte UDP packets to random IP addresses. If a malicious packet infects a new host, it will start sending probing packets. Our simulated network consists of 35 Autonomous Systems (ASes): 26 stub ASes connected by 9 transit ASes, 1700 hosts, 80 web servers and 15 interactive servers generate background traffic. A maximum of 10,000 probing packets could be sent from an infected host. A number of hosts throughout the network are initially nominated to be zombies, which generate worm probing packets.

Fig. 23 shows the volume of traffic on the ingress link for a simulated, blind scanning Slammer worm attack. At the start, the Link Monitor is activated with an alarm threshold set to an increase in average traffic on the link of twice the previous average. The Entropy Reporter module is also activated to periodically collect packet-level entropy values on five traffic features: destination IP, destination port, source IP, source port and protocol. Fig. 23 shows the worm propagation starting at about 30s (1); an alarm is generated by the Link Monitor at 37s (2) due to the high volume of traffic on the ingress link. This form of early detection is appropriate for high-volume attacks. A low-volume attack will evade detection by the Link Monitor, however, its traffic feature changes could still be captured by the Entropy Reporter module. The Entropy Reporter module is interrogated for any significant changes in the five traffic features. The most recent entropy trend for each feature is computed and compared with the previous average entropy.

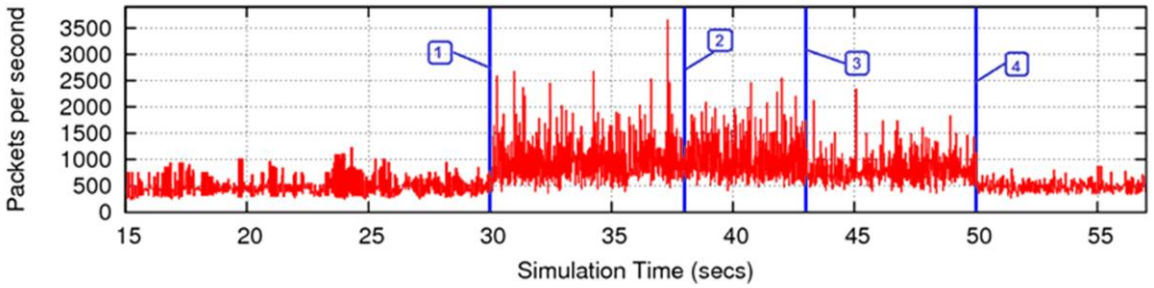


Fig. 23. Simulation results for the worm resilience strategy

Entropy changes are shown in Fig 24, which indicates that the destination IP and source port have become dispersed, and destination port and UDP protocol more dispersed. These results are reported back to further analyze the malicious traffic. Based on the traffic feature distribution, the malicious traffic is confirmed as the worm attack at 43s (3). Policies are used to specify a coarse grain remediation, actioned to initially shape 25% network traffic on the link. The Worm Differentiator is then invoked and identifies that the entropy of these traffic feature distributions match a known signature for the Slammer worm. Further analysis is performed to monitor the source IP address for all incoming UDP packets.

Source IP addresses appearing at a significantly higher than average frequency are added to a blacklist. Therefore, at 50s (4) another policy is used to reconfigure the Rate Limiter to block all probing packets, specified as all UDP packets from blacklisted sources with a destination port 1434, i.e., that used by Slammer to infect hosts.

Our platform provides results that show about 21% of benign traffic being blocked during the period of interim mitigation and 28% of malicious worm packets. Off-line risk analysis would determine if the costs (lost benign packets) are appropriate to the benefits (reduced flooding). Our platform provides inputs to such analysis, including the

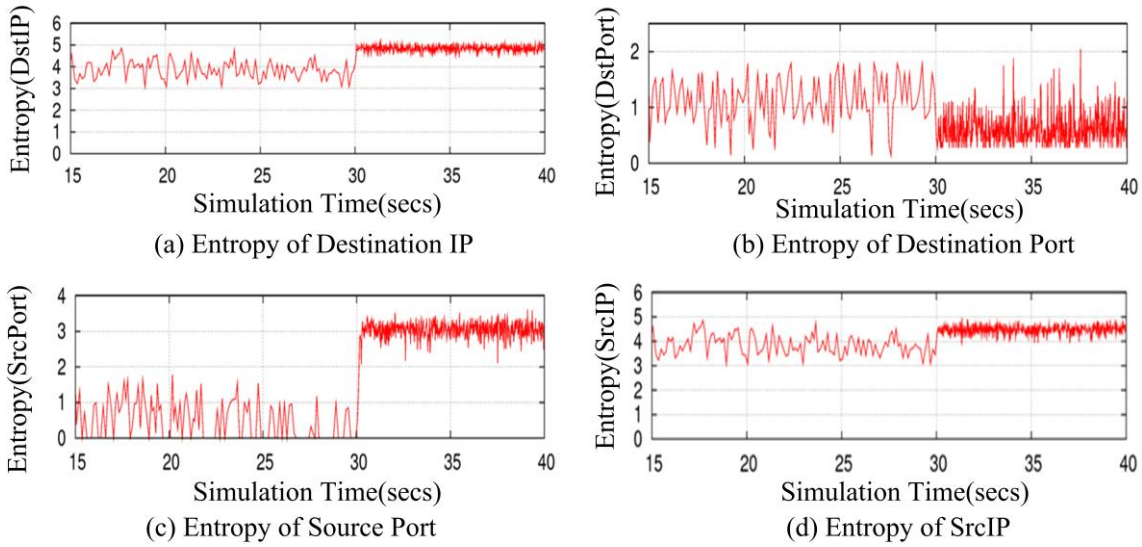


Fig. 24. Entropy changes with the Slammer Worm

applications that are affected due to the blocking. A benefit of our approach is the ease with which it enables experimentation and refinement of these trade-offs. We can develop and refine strategies through the configuration of different mechanisms via policies and the setting of parameters such as thresholds. Our platform generates results that enable evaluation of strategies. Further experimentation can identify the trade-offs between early detection and accuracy. New mechanisms can be added to analyze additional features, such as volumes at different levels of granularity or protocol.

6.2.5. Dealing with Unknown Attacks

We now demonstrate the generality of our resilience approach to combat unknown attacks. We illustrate this by fabricating a new worm. For example, a new worm might try to escape detection in two ways. Firstly it mutates an existing worm to present new traffic distribution features. In our case we mutate the Witty worm to generate random source ports in attack packets. Secondly. The worm tries to camouflage itself by hibernating for some time during its propagation. In this case, the worm propagation commences at 20s but lasts for just 4 seconds, then it hibernates before continuing at 30s. It spreads rapidly by sending UDP malicious packets with payload size 600 bytes from random source ports to random

destination ports. Fig. 25 illustrates the traffic feature changes. The destination IP becomes more dispersed during propagation. Both source port and Destination port entropy become significantly more dispersed. UDP protocol entropy is more dispersed while TCP protocol is more concentrated.

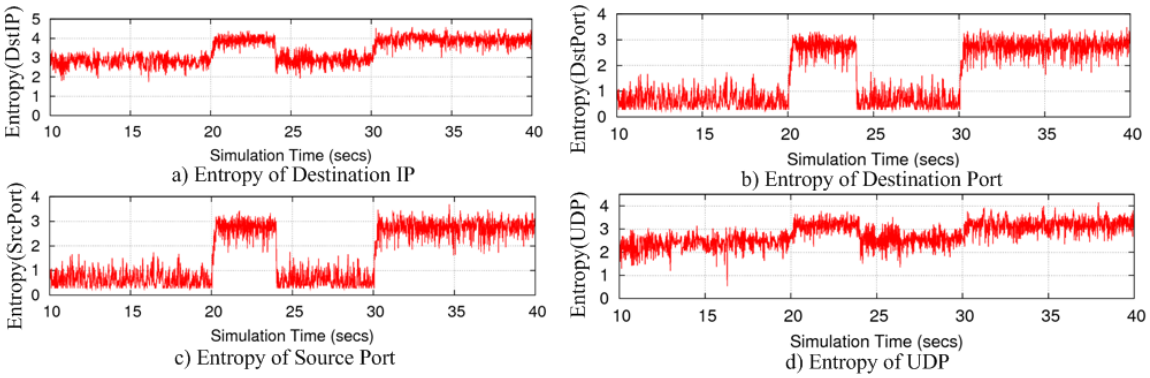


Fig. 25. Entropy changes with New worm

The new worm features are captured by EntropyReporter but the WormDifferentiator will find no matched signature for them. Then a choice of actions to be taken can be defined by policies. A human operator should be alerted in these cases with a report. Should this indeed be a new worm, then it will be a simple matter for the operator to add the new signature. Even without a signature, some automatic remediation can be implemented to protect the network.

For example, we implemented and simulated the following policy-based remediation strategy. For this unknown attack, EntropyReporter infers that the suspected worm is carried in UDP packets but from a large number of suspicious source ports and destination ports. Therefore we further analyze the frequency of suspicious packets by sender host. This could be monitored by receiving hosts. However this approach will increase computational overheads and is not easy to deploy in a large scale enterprise environment. So a better solution is to embed this strategy in the border router, and activate it once an attack is suspected. Thus, after the EntropyReporter informs that malicious packets are UDP packets, further analysis is performed to monitor the source IP address for all incoming UDP packets. Source IP addresses appearing at a significantly higher than average frequency is added to a blacklist

and RateLimiter is configured to block all packets from blacklisted sources.

Results for this initial strategy show that, with our background load of benign traffic, it produces 40% false positive rate (percentage of benign traffic blocked) while mitigation eventually achieves zero false negatives. Our platform permits further experiment to refine strategies, including adjustment of alarm and traffic shaping thresholds, and produces results whereby strategies can be evaluated.

6.2.6. Resilience Against Other Malware: Port Scan

We finally show how our worm resilience strategy can be adapted for another class of known network malware. Port scans are a reconnaissance phase of many network attacks. Typically, an attacker searches for potentially vulnerable hosts by trying to connect to the ports across random IP addresses. In order to further explore the generality of our approach, we experimented with a port scan scenario under the same topology. In this scenario the attacker scans UDP ports across the well-known port range of 0-1023, attempting to open these ports. Fig. 26 shows the attack starting at 20s, with subsequent changes to traffic features. Destination port and destination IP entropy increase substantially due to scanning across many hosts and ports. UDP entropy increases.

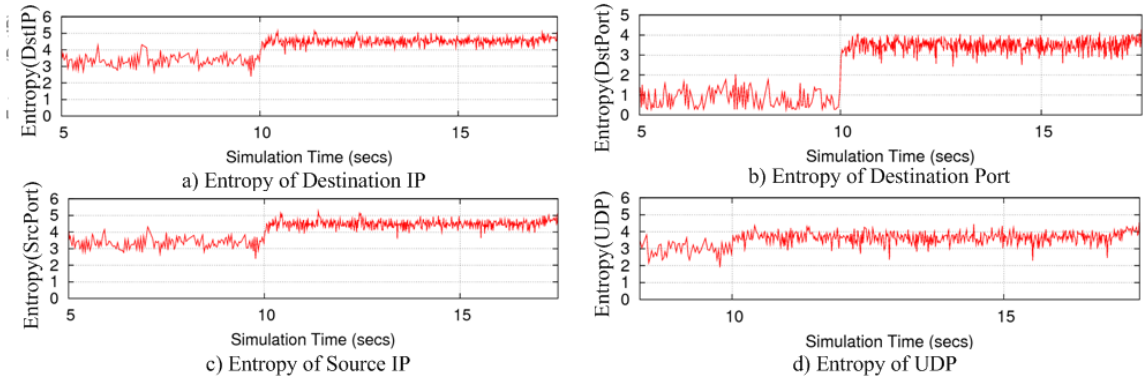


Fig. 26. Entropy changes with Port Scan

Once a potential attack is identified we can attempt to apply mitigation. In our experiment the strategy deployed is similar to the previous worm remediation. In this case, once an alarm is generated

from LinkMonitor, the LocalManager will trigger RateLimiter to initially block 30% of network traffic on the link. We then commence analysis of all incoming UDP packets and, as before successively block packets from high frequency source hosts.

6.2.7. Discussion and Evaluation of approach

A benefit of our approach is the ease with which it enables experimentation and refinement of these trade-offs. We can develop and refine strategies through the configuration of different mechanisms via policy and the setting of parameters such as thresholds. Our platform generates results that enable evaluation of strategies. For example, in Table 2 we summarize the features analyzed in our scenarios and the results of the remediation strategies deployed. Further experimentation can identify the trade-offs between early detection and accuracy. New mechanisms can be added to analyze additional traffic features, such as volumes at different levels of granularity or protocol sequences.

	Code Red	Witty Worm	Unknown Worm	Port Scan
Entropy(DstIP)	Increase	Increase	Increase	Increase
Entropy(DstPort)	Major Decrease	Increase	Major Increase	Major Increase
Entropy(SrcPort)	Increase	Major Decrease	Major Increase	Decrease
Entropy(UDP)	Decrease	Increase	Increase	Increase
Blocked benign	1%	0	40%	20%
Allowed malicious	0	0	0	0

Table 2. Performance evaluation with different malware

6.3 Summary

In this chapter, we have demonstrated our policy-based, multi-stage approach for the detection and remediation of network malware, using worm attacks as a primary example. We demonstrate how our approach could consistently be deployed to deal with different malware. In our scenarios, this approach identifies known worms on the basis of signatures that are defined as entropy perturbations of certain features. It is also capable of detection of new worms and enables partial remediation of such attacks. We have shown how we can extend the system for other forms of networked malware such as port scans.

Our approach overcomes drawbacks of existing approaches discussed under related work, by operating only lightweight detection mechanisms while a network is operating normally. More heavyweight mechanisms are only invoked once the early symptoms of a potential worm attack are detected. Thus high overhead containment is only used when there is confidence that a network is under attack. We also enable early remediation of attacks while the precise nature of the attack is diagnosed, followed by attack-specific remediation once the attack is understood. We suggest that our approach is therefore potentially more resource-efficient than existing systems, while at the same time no less accurate.

Worm detection is part of all current commercial IDS/IPS products. Different vendor's products have different actions in response to various worm propagations. For example, IBM Proventia SiteProtector manages the Proventia Network Intrusion Prevention System G/GX appliance. The core of NIPS in current ISS products is the ISS Protocol Analysis Module (PAM). It identifies and analyses 443 network protocols and data file formats. As it parses the protocols and monitors the traffic, it employs a variety of techniques to report any of 4843 suspicious events as they occur.

Different worms generate different response, for example, block worm is the response for IRC PrettyPark Worm. The Drop Packet response requests the sensor to drop the packet that triggered the signatures, such as ICQ Witty Worm, Slapper Worm, SQL SSRP Slammer Worm, TFTP MSBlaster Worm, TFTP Nachi Worm. The Block Connection response requests the sensor to drop all packets on the connection that triggered the signature, such as HTTP Nimda Worm, HTTP Spyki PhpInclude Worm, Gnutella Worm, SMB Nimda Worm, SQL Spida Worm.

Juniper Networks IPS capabilities offer several unique features that assure network security. Groups of attack signatures are identified as critical, such as HTTP Bagle Backdoor, Nimda Email Propagation, Apache Slapper Worm etc. The default action with these worms is normally to classify them as high severity and drop.

Other popular products such as Cisco IPS, McAfee M series NIPS have similar, signature-based algorithms. An innovative part of our work is that it is not only signature based but can classify suspicious

malware and put in place initial remediation until the nature of the challenge is analyzed.

We have demonstrated significant advantages of flexibility and evolvability. Through the separation of policies and mechanisms we are able to re-use and refine our resilience strategies and mechanisms. New worm types can be accommodated by micro-level adaptations such as traffic feature sets and thresholds. Our system supports the evolution of identification and remediation strategies as network contexts change. Finally, we have shown how our approach supports experimentation through simulation of resilience strategies, mechanisms and parameters prior to actual deployment.

Ongoing work would be appropriate to study how better to quantify the performance trade-offs of our approach and thus how best to optimize detection and remediation of the many challenges that networks face. We have shown the initial application of our approach to a different form of challenge, a DDoS attack, in chapter 5. We now further explore the generality of our approach by applying it to non-malicious challenges.

Chapter 7

Network Fault Resilience

Network fault resilience means the ability of the network to continue operation while a portion of the network is down. The network fully recovers when the non-operating component recovers. Fault tolerance is normally a property designed into the system to achieve this goal, which means the network could continue normal operation despite the appearance of hardware or software faults. If the network is not fault resilient, a single network fault could cause the entire network to become unavailable.

In this chapter, we extend our network resilience approach to deal with network faults. In the following, we will demonstrate our strategy for network fault resilience, and demonstrate validation via simulation and application to a real, medium sized ISP network: SWITCH. This latter work was carried out as a collaboration during my visit to ETH Zurich. The purpose is to demonstrate that the resilience strategy we

propose could not only work in a simulation environment, but could also demonstrate benefits using real ISP data.

7.1 Strategy for Network Fault Resilience

There are a number of causes of network faults, for example, the hardware fault, software error, environment challenge. Network faults could be separated into three types. Permanent faults exist in a network, where the network could not be repaired unless the root cause could be found. If the network temporarily goes down and results in minor degradation in service, this is the case for transient faults. For example, a tree contacting a power line momentarily. Intermittent faults occur on a discontinuous and periodic basis, and this could cause a degradation of service for short periods of time. Intermittent faults are now a growing problem in electronic equipment. Permanent faults are not difficult to identify, but intermittent faults are not easy to isolate, because they affect the system only part of the time, which is generally of very short duration. However, the service is workable between faults.

In order to demonstrate network resilience with our proposed system, we extend our platform to identify network failures. Fault detection is a process of capturing online indications of network disorder provided by malfunctioning devices in the form of alarms. We simulate the faults that could be triggered by non-malicious challenges, to achieve quick and efficient fault management techniques.

Our model of network failure is that it will result in the packet loss. With a broken link, the packets will be discarded until a new connection is rebuilt. Fault remediation could be used, once the network failure is identified. The overall process of failure recovery shouldn’t cause long delay in order to ensure network robustness. With the above understanding, we simulate the network failure situation. The connection failure could appear in any place within the network structure.

We designed a connection failure channel based on the OMNeT++ cDatarateChannel, which has the flexibility to be placed as a channel between any network objects. We schedule connection failure events to happen at certain simulation times. After recovery, we

schedule further failures at another time since occasional failures could occur more than once on the same connection in a real network.

We could also simulate multiple concurrent connection failure channels in the network. The monitoring system should quickly raise the alarm once a broken link is discovered: the failure detection time should be as short as possible. Then a real-time solution will be triggered.

7.2 Policy-based Network Fault Resilience Algorithm

We propose the following proof of concept algorithm to deal with network faults. Four managed object modules are used by the algorithm. The details of these modules are as follows.

Link Monitor: In OMNeT++ communication links are realized by creating “channels.” To implement a module that monitors the utilization of a link and can trigger an event if a threshold is reached, we extended the cDatarateChannel class. This allows us to place a monitoring object in arbitrary locations in a network topology. The Link Monitor is typically used to indicate the onset of a challenge that is causing anomalous traffic volumes

Entropy Reporter: entropy is the typical measure of information, could be used as the effective and efficient method to assess the changes of the traffic features. Consequently, we have implemented an entropy-based detection module, which monitors the source IP, source port, destination IP, destination port and transport protocol type for changes in entropy. The module computes the entropy of these five features using Shannon’s entropy algorithm. A threshold can be defined that triggers an event.

Entropy could be used to evaluate the uncertainty with the traffic features. When the network faults happens, some of the destination couldn’t be reached in the short term, so the entropy of the Destination IP will become more concentrated. The entropy value quantifies the degree of uncertainty with the traffic feature.

Fault Recovery: once the network fault is detected, the next stage is helping the network to fully recover. The recovery strategy will

eventually affect the reliability of the network. A large scale of network should tolerate elements/nodes failures while continuing normal operation. As the number of network failures increase, the difficulty of network recovery increases. The alarm will be raised when the network fault is identified. At this stage, either the automatic or network operator initiated fault recovery solutions are activated to address the fault and bring the network back to normal operation.

A few corresponding solutions will be proposed to correct the fault, for example, the packets could be re-routed to a backup link. The recovery stage may need to deal with complex dependencies issues between network elements. To simply replace or reset the network element may not right, since other network elements may need to be reconfigured or reset. Any mis-replacement may worsen the scenario.

The algorithm for network fault resilience is shown in Figure 27. At the beginning, the local manager will set the threshold and activate the link monitor, also the entropy reporter will start calculating the entropy value for five traffic features, which include Destination IP, Destination Port, Source IP, Source Port and Protocol. Once the traffic volume rises above the threshold, the alert will be raised in order to do further in depth analysis. Then the entropy reporter will check the entropy value changes for each of the traffic features, if both the entropy of the destination IP and source IP drop significantly, then we could assume that this is the behavior of a network fault. This is based on Lakhina [2005]'s findings. At this stage, the network operator will receive an alarm for the network fault event, and have the options to choose either automatic recovery, for example, re-route the packets another link, or manual recovery, such as manually reset/reconfigure the individual network element.

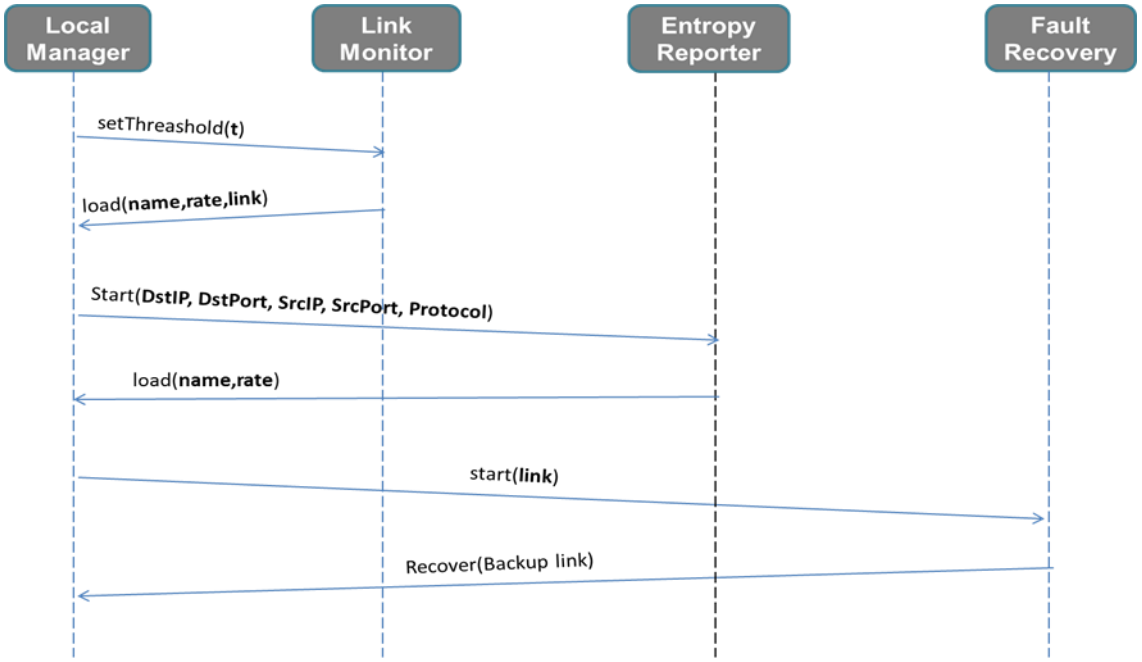


Fig. 27. Entropy based network fault resilience algorithm

7.2.1 Experiment of the network fault resilience

For the case study, our simulated network consists of 35 Autonomous Systems (ASes): 26 stub ASes connected by 9 transit ASes, 1700 hosts, 80 web servers and 20 interactive servers generate background traffic.

Our resilience strategy is simulated at a stub AS. Within a single stub AS, two links between the edge router and gateway router is down as demonstrated in Figure 28. The network fault we simulated is the intermittent fault. The connection failure happens at 10 sec and last for 5 sec. Then the network is back as usual for another 5 sec. After that, the links fails again. The various resilience mechanisms are activated on a gateway router of the AS and its ingress link from a core router. Initially, a Link Monitor module is invoked on each of the ingress links to monitor link utilization, a threshold parameter is defined, which if exceeded results in an event being generated. An Entropy Reporter module is also invoked and configured with a list of features that it is to monitor.

As mentioned earlier, the Entropy Reporter module continuously monitors the traffic features' distributions using Shannon's entropy algorithm. In the previous chapters, we show the Entropy Reporter

module could be applied to detect malware attacks, for example a worm or port scan. In this experiment, the same Entropy Reporter module is deployed to capture network faults. Based on the distribution of the traffic features, the Entropy Reporter module could differentiate network faults from other types of malicious attacks.

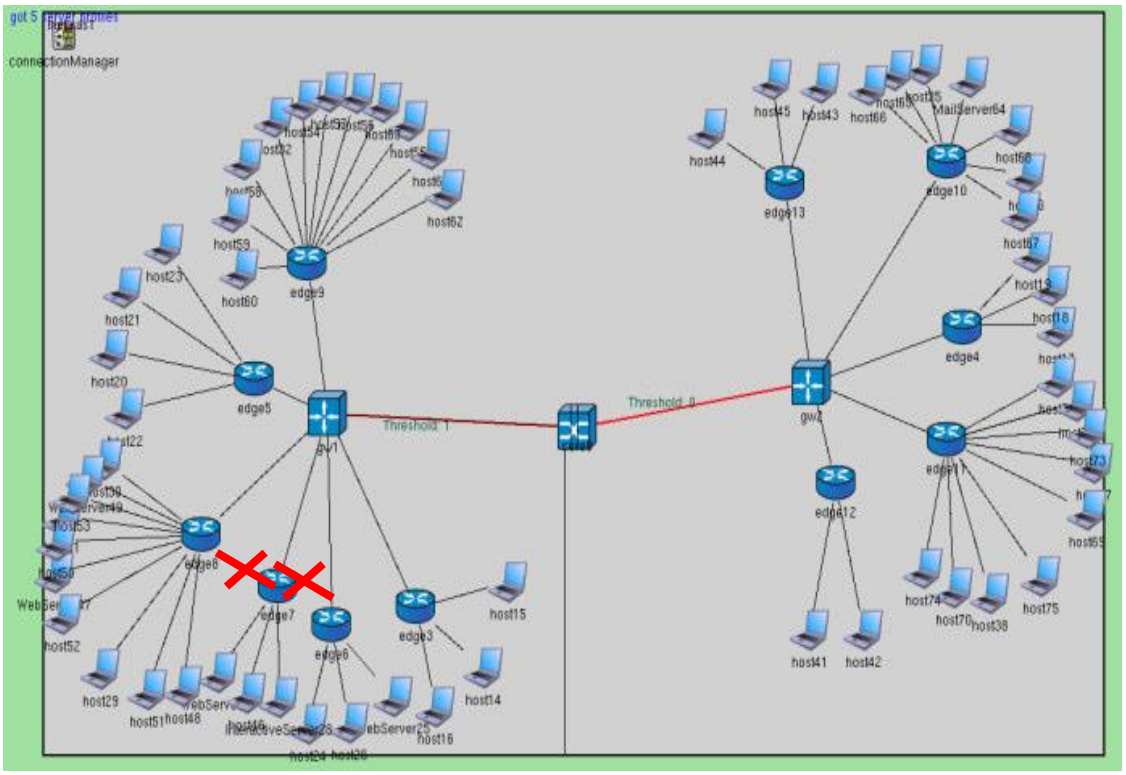


Fig. 28. AS level network topology

Figure 29 shows the volume of the traffic on the ingress link between gateway router and core router for the simulated network fault. The overall traffic drops between 10s and 15s when the connections are broken, then the network operation back to the normal after 15s. The traffic falls down again at 20s. Figure 30 and figure 31 represent the entropy of the destination IP and source IP drop correspondingly when the network fault occurs.

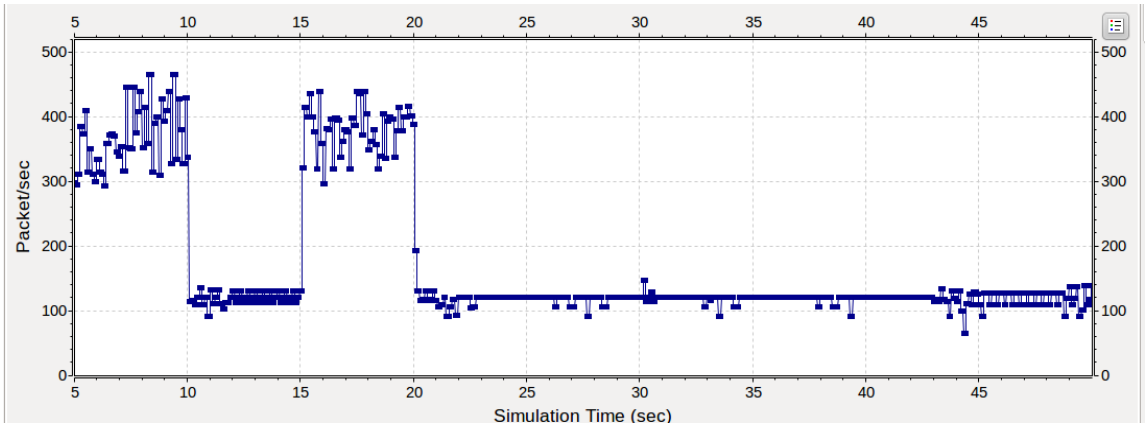


Fig. 29. Traffic changes with the intermittent network fault

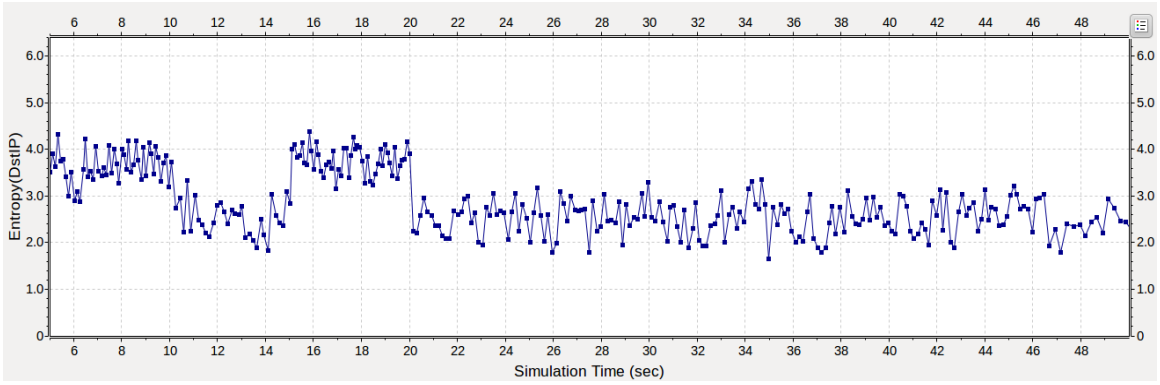


Fig. 30. Destination IP Entropy changes

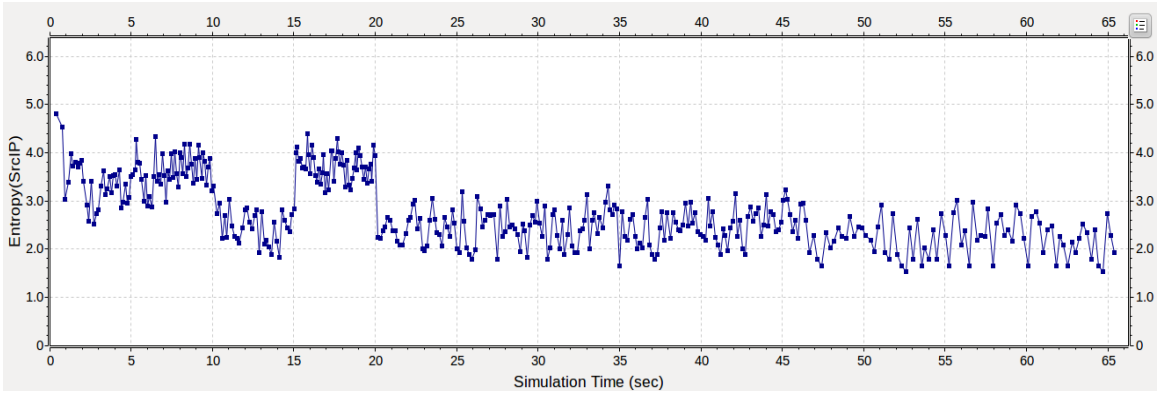


Fig. 31. Source IP Entropy changes

7.3 Network Fault Resilience Using a Classifier Approach

In order to demonstrate the flexibility and adaptability of our proposed multi-stage policy based challenge detection platform, here we propose a further solution for network fault detection.

This work has been carried out during my visit to the ETH Communication Group, which is part of a collaboration between NICTA and ETH. The classifier has previously been developed and presented [Eduard et al., 2012]. We further extend this work and deploy the classifier into our multi-stage policy based network resilience approach. In the following sections, we introduce how the classifier based multi-stage network fault resilience strategy could be applied to classify network faults in the SWITCH network.

7.3.1 Network Fault Resilience Experiment on SWITCH

SWITCH was founded in 1987, as one of the very first organizations in Switzerland to provide the internet service to the country. As the medium sized ISP, it offers internet service to approximately 30 Switzerland universities, government institutions, and research labs, e.g., IBM, PSI, CERN, and other educational institutions. SWITCHlan has some 2600 kilometers of glass fibers, built through the whole of Switzerland. it brings Swiss universities and research institutions together via secure exchange of data at 10 Gigabits/sec.

The backbone of the SWITCH network is depicted in figure 32. The SWITCH network has six border routers and traffic is collected from the Cisco border routers only. A flow data collector is deployed to collect and buffer incoming flow data. It runs software to capture the data and store it to files (e.g., one file per hour). It also generates metadata files containing the source of each data packet received (UDP) or the connection (TCP) and e.g., the timestamp, sequence number, and other things. The hourly generated files are buffered by the flow data collector. These files are compressed and available for download. Normally the data compression happens before the download with the purpose of saving processing power.

The secure flow data infrastructure is deployed to process the flow data after archiving, also performing analysis for the archived flow data. The data will be parsed and investigated at this stage. The

processing time is about real time analysis of the archived data. This process could be accelerated by massive parallelization using multiple processing nodes. The large amount of flow data will be stored into the large disk array or tape library. The current storage server has the capacity for roughly 40TiB. The raw netflow data from special events (e.g. malicious attacks, network failure) are kept on the storage servers.

In order to reduce data volume, no packet traces are gathered on the server, only the flow traces are stored, for example, NetFlow V5 and NetFlow V9. The complete flow level traffic is captured from the border router and there is no packets sampling applied. The traffic comprises both the normal and abnormal traffic happening in the daily operation network, include network faults, DDoS, network scan, alpha flows etc. The dataset captured in the SWITCH backbone network is in large data volume and multidimensional features of the traffic can be measured.

A flow record is essentially a unidirectional sequence of packets between two end points that is characterized by the 5-tuple of source and destination IP address, source and destination port number, and transport protocol, also other information could also include, for example, connection start time, connection duration, the number of packets and bytes transferred. If the flow does not have a reverse matching flow, then it will be called the one- way flow.

In the SWITCH network, the classifier is deployed to do offline analysis and the processing chain has been optimized for offline analyzes on existing/historic traffic data archives. The classifier validates on 7.41 petabytes of traffic from SWITCH boarder routers, and 2.2 million IP addresses are monitored. A massive dataset of NetFlow records was collected without sampling between 2004 and 2010.

As explained earlier, the classifier could be used to detect service outages based on the one-way to two-way flow ratio targeting a service. The classifier works as the following stages:

Fig. 32. The backbone for the SWITCHlan network [web1]

1. Collecting data

The first step is collecting the traffic data.

For the offline analysis, the flow data will be collected and YAF (Yet Another Flowmeter) could be used as a soft flow meter to create flow data files in the standardized IPFIX format. But IPFIX flow data has

to distinguish between unidirectional and bidirectional flows to be an acceptable input to the one-way flow classifier. The one-way classifier captures the count of one-way flows targeted at unreachable services

A suitable flow generator is YAF, but any flow generator that uses an IPFIX flow template including the dedicated standard information elements for describing bidirectional flows could be applied.

In the SWITCH network, the flow data is gathered from the border gateway. The SWITCH network topology will be demonstrated in section 7.3.1. All the incoming and outgoing traffic from the boarder gateway is monitored.

If it is the edge network, the traffic could also be monitored from a gateway router. If the classifier works on the one way flow, the incoming and outgoing traffic will be separated first.

For the online analysis, the traffic is collected in real time from the simulator.

2. Identify Service and P2P Host

The traffic target service and P2P hosts are monitored all the time, so that we could get the list of services and P2P hosts. A service is defined as a triple {IP, protocol, port}. Such 3-tuples registered as safe destinations and consider all flows to and from these services as benign one-way flows caused by connection failure due to any reason, e.g. congestion, outages. This analysis is run over a full observation period to leverage temporarily running services and temporary outages of services.

3. Identify unused IP addresses

For the offline analysis, because the traffic is observed over a long period of time, both the incoming and outgoing traffic are monitored. So if an IP address never shows up in a two way flow and only appears in one or two one-way flows, then it will be listed as an unused IP addresses.

4. Assign signs to one-way flows

For the offline analysis, the traffic data is collected and saved into a CSV (comma separated value) file. Each sign is listed with number of one-way flows carry it. Sign combinations that do not appear in the data are not reported to limit the size of the output data. Additionally, shorter versions of this list are provided that do not report sign combinations with a flow count below a threshold of 100fpm or 1000fpm (fpm: flows per million).

The output file could be generated to report the sign combinations of any input one-way flow. The sign sets contained in this file are aligned with the flows such that e.g. entry 100 describes the flow 100 when flows and sign sets are numbered in ascending order.

5. One-way flow classification

Following the defined rules, the flow class will be classified based on the 18 signs, if the service is classified as unreachable, then we could confirm it is network fault.

The analysis focuses on the one way traffic, as when network failure happens, the packets will send from the source host to the destination host, but it couldn’t reach the destination due to the fault. However, there are also other possibilities with one way traffic, for example peer to peer, malicious scanning, backscatter, suspected benign, bogon. Figure 33 demonstrates the one way traffic classification process. The classifier could be deployed to differentiate network failure from other scenarios. The scheme uses 17 rules and includes the novel heuristics and combine exiting algorithms. Each one way flow has been assigned with 18 different signs, to help identify the root cause of the one way flows.

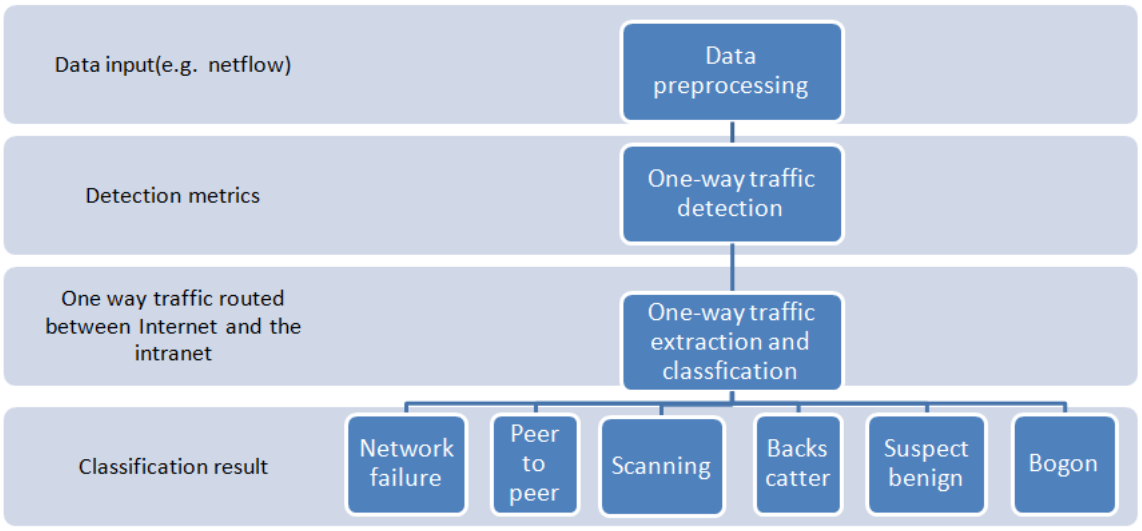


Fig. 33. One way traffic classification process

The approach could passively monitor the large dataset without inflicting network load. The time interval applied to find the reachability problem is 10 minutes in this case. The impact of different time interval has been tested on flow metrics aggregated for the whole day in [Eduard, 2012], and within 10 minutes could efficiently search for the flow pairs. Three popular services are used to recognize the connection failure including the email service, main web site access and the software distribution service. Many services could be monitored concurrently. Network failure could be confirmed if all the three services are not reachable by the defined time interval. There is a large scale of clients affected by the identified network failure. After classification, the flows data are separated into different classes. 4.8% of the total one-way flows are classified as the network failure, while malicious scanning dominates the one–way flows by occupying 83.5% of all one-way flows.

Another experiment has been carried out to apply the classifier on a week's archived NetFlow data from the Department of IT and Electrical Engineering at ETH Zurich. The coinciding outage was observed at 19:40 on the 23rd June 2011 and last more than 10 min. During this interval, the three key services are not reachable. When monitoring the time series of the volume of one way flows, it is found that the mail service has a sharp increase when the network failure happens due to the automatic retry attempt as demonstrate in Figure

34. After investigation, it is found that the failure happens due to a planned router software upgrade.

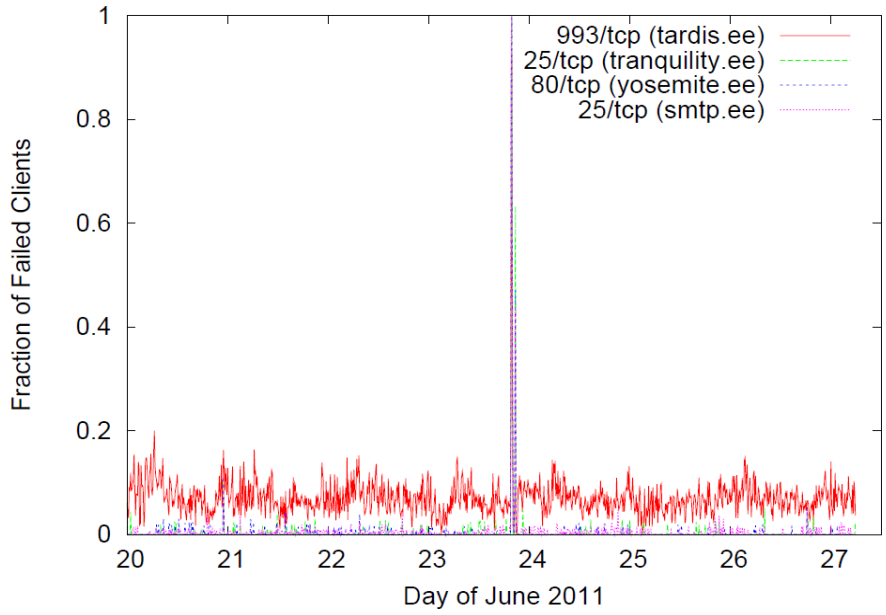


Fig. 34. Network Failure observed on the 23rd of June 2011 19:40 at ETH Zurich [Eduard, 2012]

7.3.2 Network Fault Resilience Experiment in Simulation

During the visit to ETH Zurich, we worked in cooperation with Eduard to applied the classifier to our simulation. The classifier approach had been validated by experimentation in the large dataset at petabyte level in the SWITCH network. We brought it into the simulation environment in order to differentiate network failure from other scenarios, in order to indicate that our policy based multi stage platform could be extended or adapted with different algorithms to deal with different challenges and make network more resilient.

Figure 35 depicts the classifier based network fault resilience algorithm. At the start, LocalManagerMO will set the threshold to the LinkMonitorMO, so if the network traffic drops dramatically, it will signal the LocalManagerMO. The FlowExporterMO will truncate flow records after a specific time out period, e.g., 60s or 180s, and send records to the ClassifierMO with a given sampling rate, both of which are preconfigured by LocalManagerMO. Once the FlowExporterMO

exports the flow records, they will be used as the input for the classifier.

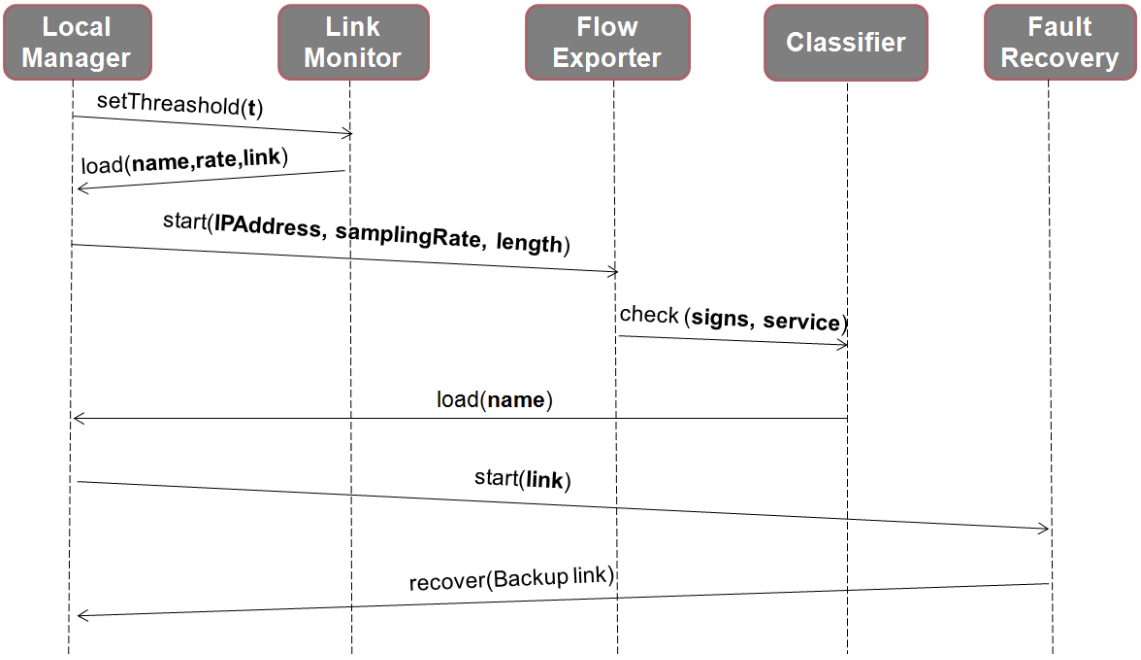


Fig. 35. Classifier based network fault resilience algorithm

Network failure could be detected when many services outage occurs. The classifier will classify the flows based on the 18 different signs, each flow will be checked against all the signs and identify the matched ones. The signs of a flow incorporate a number of well-designed rules, and they will be used to determine the equivalent class and the unknown class. 14 classification rules are included within the classifier.

These rules are applied to classify network failure from other one-way flows. Each rule associates with the specific signs, these signs need to be present or absent. Once the fault is declared, the fault recovery process will start to help the network continue normal operation. A backup path is configured in parallel with the working path. In order to minimise disruption, a quick switch over will take place.

Class	Rules
Malicious Scanning	1. {TRWscan, $\overrightarrow{HCscan}$, $\overrightarrow{PotOk}$} => Scanner 2. {HCscan, $\overrightarrow{TRWscan}$, $\overrightarrow{TRWnom}$, $\overrightarrow{PotOk}$) => Scanner 3. {TRWnom, HCscan} => Scanner 4. {TRWscan, HCscan, $\overrightarrow{PotOk}$) => Scanner 5. {GreyIP, Onepkt, $\overrightarrow{HCscan}$, $\overrightarrow{TRWscan}$, $\overrightarrow{\text{ICMP}}$, $\overrightarrow{\text{UDP}}$} => Scanner 6. {GreyIP, $\overrightarrow{TRWscan}$, $\overrightarrow{HCscan}$, $\overrightarrow{\text{Onepkt}}$, $\overrightarrow{\text{ICMP}}$} => Scanner 7. {Onepkt, $\overrightarrow{TRWscan}$, $\overrightarrow{\text{GreyIP}}$, $\overrightarrow{\text{ICMP}}$, $\overrightarrow{HCscan}$, $\overrightarrow{TRWnom}$, $\overrightarrow{Failure}$, $\overrightarrow{\text{Large}}$, $\overrightarrow{PotOk}$} => Scanner 8. {GreyIP, Onepkt, $\overrightarrow{\text{TRWscan}}$, $\overrightarrow{\text{HCscan}}$, $\overrightarrow{\text{ICMP}}$, $\overrightarrow{\text{TCP}}$ } => Scanner 9. {ICMP, $\overrightarrow{\text{TRWscan}}$, $\overrightarrow{\text{HCscan}}$, $\overrightarrow{\text{TRWnom}}$, $\overrightarrow{\text{PotOk}}$, $\overrightarrow{\text{InOut}}$} => Scanner
Network Failure	10. {Failure, $\overrightarrow{\text{TRWscan}}$, $\overrightarrow{\text{HCscan}}$} => Failure
Suspected Benign	11. { PotOk, $\overrightarrow{\text{Failure}}$, $\overrightarrow{\text{TRWnom}}$) => Benign 12. {Large; $\overrightarrow{\text{GreyIP}}$, $\overrightarrow{\text{TRWscan}}$, $\overrightarrow{\text{HCscan}}$, $\overrightarrow{\text{TRWnom}}$, $\overrightarrow{\text{PotOk}}$, $\overrightarrow{\text{ICMP}}$} => Benign 13. {TRWnom, $\overrightarrow{\text{GreyIP}}$, $\overrightarrow{\text{HCscan}}$, $\overrightarrow{\text{Failure}}$} => Benign 14. {ICMP, InOut, $\overrightarrow{\text{TRWscan}}$, $\overrightarrow{\text{HCscan}}$, $\overrightarrow{\text{TRWnom}}$, $\overrightarrow{\text{PotOk}}$} => Benign

Table 3. Rules to classify one-way flow by using the combination of signs (derived from [Eduard, 2012])

7.4 Summary

Fault management is a major component of network management suite. We need to introduce innovative concepts for fault detection, root cause analysis and self-healing architectures.

The system should implement root-cause analysis not only to detect the faults once they occur, but also to identify the source of fault for performing automatic fault recovery. Different types of network service faults need to be measured: they range from node misbehavior at different layers (MAC, routing), to software misconfigurations. We also need to consider the system's vulnerabilities include component faults, which is caused by software bugs or the deterioration of hardware.

In the datacenter or the enterprise network, the traditional methodology to monitor network failure is usually to send probing packets and/or to check server logs. In our approach, we developed two ways to perform the task: the entropy based method could be used to check the pattern of traffic features changes, and the classifier based method could exploit traffic flow data and monitor reachability. Passive measurement effectively reduces overhead compared to the traditional active probing approach. We have demonstrated further adaptability and extensibility of our approach by validating different network failure resilience strategies.

Chapter 8

Conclusion: Evaluation of Our approach and Future Work

We have described a novel solution that enables the progressive multi-stage deployment of resilience mechanisms, based on incomplete challenge and context information. Volume resilience, malware resilience and network failure resilience have been demonstrated in chapter 5, chapter 6 and chapter 7 respectively. We have illustrated the flexibility of our approach when adapting to new challenges in the form of zero-day malware. We have also shown how our approach enables experimentation and development of new resilience strategies. Here we first summarily evaluate our approach from different aspects. We conclude by discussing future work in the context of existing products.

8.1 Evaluation

8.1.1 Generality, Efficiency and Scalability

A major challenge for the automatic identification and remediation of networked malware is that these challenges display a wide array of features and are also ever changing in their characteristics in order to evade detection. Our approach allows detection strategies to be composed such that a range of Managed Objects can be applied to early detection and subsequent identification. For example, malware that propagates at high rates can initially be detected via a link monitor. More stealthy, slower propagating malware can evade volume based detection, but can possibly be detected by monitoring entropy of traffic features. In general, we propose that our approach is capable of adopting and applying a wide range of algorithms that have been developed in the literature.

Our multi-stage approach permits deployment of strategies for real-time detection without incurring the overall costs of other approaches. Always-on, real-time monitoring is relatively lightweight. While more complex analysis is only invoked at a point where there is confidence that the network is under challenge.

The approach also enables distributed detection, identification and remediation. Under the control of a policy-based local manager, managed objects can be configured and invoked at multiple vantage points in the network. This can be applied at the level of sub-networks or across an entire domain.

8.1.2 Early Detection vs Accuracy

Our method is based on packet level analysis, which has the advantage of early detection of potential attacks. Many previous approaches are based on flow analysis. However routers or switches only export flows after a ramp-up period (typically 15 seconds or more). A further period is then required for anomaly detection algorithms to analyse an attack. In this time, the whole network will be at risk [21].

Our packet based approach can detect large-scale attacks more quickly and can apply interim remediation. This may be at the expense of accuracy, resulting in high false positive rates, but this is

traded off against the protection of network assets provided by early remediation.

8.1.3 Flexibility and Evolution

Policies are deployed to coordinate different managed objects, and can be configured in a flexible and simple way via parameterization. Our platform has the flexibility to enable adaptation of strategies to fit different detection scenarios and the inclusion of new mechanisms. For example, a Managed Object to analyze packet payload content could be added as a stage in the identification of a stealth attack.

New challenges will continue to emerge in networks. We have shown how our platform can be used to develop resilience strategies against volume anomalies such as DDoS attacks in chapter 5. In doing so we easily re-used some of the managed objects described here, such as LinkMonitor and RateLimiter. We have shown that our approach can also be used to experiment with and develop resilience strategies for other forms of challenge, such as network faults. In general, our approach facilitates development and deployment of network resilience strategies, which can be further refined as new challenges emerge and/or our understanding of how to achieve resilience is enhanced.

8.2 Future Work

This thesis has elaborated some initial, proof-of-concept implementations to demonstrate our approach to network resilience. We conclude by exploring briefly how our approach could be embedded into operational networks, with reference to existing products.

The limitation of many products is that they are signature based. For example, snort uses sourcefire certified VRT rules for signatures with updates available for the registered user. The main benefit of snort is it gives users the flexibility to create new rules, new plugins and preprocessors. However if a rule hasn’t been released for a new attack or the developer hasn’t created the rule, then a new malicious attack is unlikely to be identified. Likewise the market leading product checkpoint NGX with IPS gives the engineer the option to

automatically update or manually update signatures, but cannot quarantine or mitigate a new attack.

Our approach could be integrated such that existing tools could automatically learn the traffic patterns of unknown attacks, and then provide the network operator the choice to do early network remediation. Furthermore, we could develop the approach to provide the operator with information about what particular service or application was targeted. This would help the operator to further refine the detection signatures.

A web application firewall (WAF) could be another context for further research and development. A network firewall normally protects the network from the network layer, and an IDS/IPS defends from the transport layer. A network firewall normally has policies configured for blocking or allowing specific ports/addresses. The IDS/IPS usually verifies traffic behavior via signatures. However, common web application attacks like SQL injection, cross site scripting, cookie tampering, and the top ten OWASP threats are challenging the network from the application layer. They are unlikely to be identified by a network firewall or IDS/IPS, and need to combine IDS/IPS with WAF. Future work could extend our network resilience strategies to cope with OWASP vulnerabilities and thus protect the network from the application layer.

Finally, future work could also focus on developing resilience strategies that integrate the simultaneous detection, identification and remediation of multiple challenge types including worms, DDoS, flash crowds, network faults, etc. This may introduce issues of policy conflict and resolution, and resource trade-offs. Our work enables experimental scenarios to be explored as new challenges to network resilience arise.

Bibliography

J. P. G. Sterbenz, D. Hutchison, E. K. C¸ etinkaya, A. Jabbar, J. P. Rohrer, M. Sch¨oller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," Comput. Netw., vol. 54, no. 8, pp. 1245–1265, 2010.

M. Roesch, "Snort: Lightweight intrusion detection for networks," in 13th Systems Administration Conference (LISA). USENIX Association, November 1999.

A. Hussain, J. Heidemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," in SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications. New York, NY, USA: ACM, 2003, pp. 99–110.

A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," SIGCOMM Comput. Commun. Rev., vol. 35, no. 4, pp. 217–228, 2005.

V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," ACM Comput. Surv., vol. 41, no. 3, pp. 1–58, 2009.

T. T. T. Nguyen and G. J. Armitage, "A survey of techniques for internet traffic classification using machine learning." IEEE Communications Surveys and Tutorials, vol. 10, no. 1-4, pp. 56–76, 2008.

T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the DoS and DDoS problems," ACM Comput. Surv., vol. 39, no. 1, p. 3, 2007.

Y. Yu, M. Fry, A. Schaeffer-Filho, P. Smith, and D. Hutchison, "An adaptive approach to network resilience: Evolving challenge detection and mitigation," in DRCN'11: 8th International Workshop on Design of Reliable Communication Networks, Krakow, Poland, October 2011, pp. 172 –179.

T. A. Limoncelli, "OpenFlow: a radical new idea in networking," Commun. ACM, vol. 55, no. 8, pp. 42–47, Aug. 2012.

Fessi A., Plattner, B., et al.: Resilience and Survivability for future networking: framework, mechanisms, and experimental evaluation, ResumeNet Deliverable D1.5, 2009.

A. Schaeffer-Filho, P. Smith, A.Mauthe, D.Hutchison, Y. Yu, M. Fry, A Framework for the Design and Evaluationof Network Resilience Management, in 13th IEEE/IFIP Network Operations and Management Symposium, USA, 2012.

Jung, J., Paxson, V., Berger, A., Balakrishnan, H: Fast portscan detection using sequential hypothesis testing, IEEE, pp. 211-225, 2004.

Wuhib, F., Stadler, R.: Decentralised Service-Level Monitoring Using Network Threshold Alerts, IEEE Communications Magazine, pp. 44, 2006.

Jackson, A.W., Milliken, W., Santivanez, C.a., Condell, M., Strayer, W.T.: A Topological Analysis of Monitor Placement, IEEE, pp.169 – 178, 2007.

Fry, M., Fischer, M., Karaliopoulos, M., Smith, P., Hutchison, D.: Challenge identification for network resilience, IEEE, 2010.

Labovitz, C., Ahuja, A., Bose, A., Jahanian, F.: Delayed internet routing convergence, Networking, vol. 9, pp. 293--306, IEEE/ACM Transactions, 2001.

Sethi，M. S., A. S.: A survey of fault localization techniques in computer networks, vol. 53, pp.165--194, Science of Computer Programming, 2004.

Shavitt, Y., & Shir, E., 'DIMES: Let the Internet Measure Itself', ACM SIGCOMM, 35, 5 October 2005.

Cook, E., Mortier, R., Donnelly, A., Barham, P., & Isaacs, R., 'Reclaiming Network-wide Visibility Using Ubiquitous Endsystem Monitors', Proceedings of Usenix Technical Conference, 2006.

Androulidakis, G., Chatzigiannakis, V., and Papavassiliou, S. 'Network anomaly detection and classification via opportunistic sampling', IEEE Network 23, 1, 6-12, 2009.

Algirdas, A., et al., 'Basic Concepts and Taxonomy of Dependable and Secure Computing', IEEE Transactions on Dependable and Secure Computing, 1, 1 pp 11-33, January 2004.

Beitollahi, H, Deconinck, G., 'Dependable Overlay Networks', Dependable Computing, 2008. PRDC '08. 14th IEEE Pacific Rim International Symposium, 104 - 111, 2008

Castelucio, A., Ziviani, A., and Salles, R. 'An AS-level overlay network for IP traceback', IEEE Network 23, 1, 36-41, 2009

Chandola, V., Banerjee, A., and Kumar, V. 'Anomaly detection: A survey', ACM Computing Surveys 41, 3, July 2009

Cook, E., Mortier, R., Donnelly, A., Barham, P., & Isaacs, R., 'Reclaiming Network-wide Visibility Using Ubiquitous Endsystem Monitors', Proceedings of Usenix Technical Conference, 2006.

Daniel, J., W., 'A taxonomy of computer intrusions', Massachusetts Institute Of Technology Thesis, 1998

Hansman, S., & Hunt, R., 'A Taxonomy of Network and Computer Attacks', Computers and Security, 24, pp 31-43, 2005.

Hussain, A., Heidemann, J., & Papadopoulos, C., 'A Framework for Classifying Denial of Service Attacks', Proceedings of ACM SIGCOMM'03, pp99-110, August 2003

Jackson, A.W., Milliken, W., Santivanez, C.a., Condell, M., & Strayer, W.T. 'A Topological Analysis of Monitor Placement', Sixth IEEE International Symposium on Network Computing and Applications, 169-178, 2007

Jeffrey, D., Gupta, N., and Gupta, R, 'Fault localization using value replacement'. In Proceedings of the 2008 international Symposium on Software Testing and Analysis. ACM, 167-178, July 2008

Katzela I., & Schwartz, M., 'Schemes for Fault Identification in Communication Networks', IEEE/ACM Transactions of Networking, 3, 6 December 1995

Killourhy, K., Maxion, R., & Tan, K., 'A Defence-Centric Taxonomy Based on Attack Manifestations', Proceedings of the International Conference on Dependable Systems and Networks, pp 102-111, June 2004.

Kim, B. U., Al-Nashif, Y., Fayssal, S., Hariri, S., and Yousif, M. 'Anomaly-based fault detection in pervasive computing system'. ICPS '08, ACM, 147-156, 2008

Kompella, R., Singh, S., & Varghese, G., 'On Scalable Attack Detection in the Network', Proceedings of ACM IMC'04, pp 187-200, October 2004.

Lakhina, A., Crovella, M., & Diot, C., 'Diagnosing Network-Wide Traffic Anomalies', Proceedings of ACM SIGCOMM'04, pp 219-230, September 2004.

Lazar, A., et al, 'Models and Algorithms for Network Fault Detection and Identification: A Review', ICSS, 92.

Li, X., Bian, F., Zhang, H., Diot, C., Govindan, R., Hong, W., & Iannaccone, G., 'MIND: A Distributed Multi-Dimensional Indexing System for Network Diagnosis', Proceedings of IEEE INFOCOM, April 2006.

Mirkovic, J., & Reiher, P., 'A taxonomy of DDOS Attacks and Defence Mechanisms', ACM Computer Communication Review, 34,2, pp 39-54, April 2004.

Nicholas, W., Vern, P., Stuart, S., R. C., 'A Taxonomy of Computer Worms', ACM, 11-18, 2003

Peng, T., Leckie, C., & Ramamohanarao, K., 'Information Sharing for Distributed Intrusion Detection Systems', Journal of Network and Computer Applications, pp 877-899, 2007.

Ringberg, H., Soule, A., Rexford, J., & Diot, C., 'Sensitivity of PCA for Traffic Anomaly Detection', Proceedings of ACM SIGMETRICS, pp 109-120, June 2007.

Sethi., M. S. & A. S. 'A survey of fault localization techniques in computer networks.' Science of Computer Programming, 53: 165-194, 2004

Stefan.A, 'Intrusion detection systems: A survey and taxonomy.' Technical Report 99-15, Department of Computer Engineering, Chalmers University, March 2000.

Shanbhag, S. & Wolf, T. 'Accurate anomaly detection through parallelism'. IEEE Network, 23, 1, 22-28. 2009.

Soule, A., Silveira, F., Ringberg, H., & Diot, C., 'Challenging the Supremacy of Traffic Matrices in Anomaly Detection', Proceedings of ACM IMC'07, pp 105-110, October 2007.

Ting W., Mudhakar S., Dakshi A. & Ling L., 'Learning, Indexing, and Diagnosing Network Faults', Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, July 2009.

Wang, N., Yuan, Z., and Wang, D. 'Improving Process Fault Detection and Diagnosis Using Robust PCA and Robust FDA', IEEE Computer Society, 02, 54-59, April 2009.

Wuhib, F., & Stadler, R., 'Decentralised Service-Level Monitoring Using Network Threshold Alerts', IEEE Communications Magazine, 44, 10 October 2006.

Yu, Y., Jones, J. A., and Harrold, M, 'An empirical study of the effects of test-suite reduction on fault localization', ACM, 201-210, May 2008 Cook

F. Hugelshofer, P. Smith, D. Hutchison, and N. J. Race, “Openlids: a lightweight intrusion detection system for wireless mesh networks,” in MobiCom ’09: Proceedings of the 15th annual international conference on Mobile computing and networking. New York, NY, USA: ACM, 2009, pp. 309–320.

D. Brauckhoff, K. Salamatian, and M. May, “A signal processing view on packet sampling and anomaly detection,” in Proceedings of the 29th conference on Information communications, ser. INFOCOM’10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 713–721.

W. Yu et al., “On Defending Peer-to-Peer System-based Active Worm Attacks”, IEEE Global Telecommunications Conference, IEEE Press,pp. 1757-1761, 2006 .

D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, “ Inside the Slammer worm”, IEEE Security and Privacy Magazine, vol. 1(4), 33-39, 2003.

M. La Polla, F. Martinelli, D. Sgandurra, "A Survey on Security for Mobile Devices”, Communications Surveys Tutorials, IEEE, Issue: 99, 2012 , pp.1-26

C. Pantanilla, "Worm Spreads via Facebook Private Messages, Instant Messengers”, Malware blog, Trend Micro, May 2012

"Flame worm one of the most complex threats ever discovered”, Virus bulletin fight malware and spam, May 2012

P. Li, M. Salour, and X. Su, “A survey of Internet worm detection and containment”, Communications Surveys & Tutorials, IEEE, vol. 10, no. 1, pp. 20-35, 2008.

C. Chen, Z. Chen, and Y. Li, “Characterizing and defending against divide-conquer scanning worms”, Computer Networks, vol. 54, no. 18, pp. 3210-3222, Dec 2010.

S. Chen and Y. Tang, “DAW: A distributed antiworm system”, IEEE Transactions on Parallel and Distributed Systems, pp. 893-906, 2007.

Cisco, “Cisco intrusion prevention system,” Available at: http://www.cisco.com/go/ips. Accessed in: February 2011.

IBM, “Security network intrusion prevention system,” Available at: http://www-01.ibm.com/software/tivoli/products/security-networkintrusion-prevention/. Accessed in: February 2011.

Enterasys Secure Networks, “Enterasys intrusion prevention system,” Available at: http://enterasys.com/products/advanced-securityapps/dragon-intrusion-detection-protection.aspx. Accessed in: February 2011.

A. Lakhina, M. Crovella, and C. Diot, “Diagnosing network-wide traffic anomalies,” in SIGCOMM ’04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications. New York, NY, USA: ACM, 2004, pp. 219–230.

N. Damianou et al., “The Ponder policy specification language,” in POLICY ‘01. Bristol, U.K.: IEEE Computer Society, January 2001, pp. 18–39.

E. Lupu, N. Dulay, M. Sloman, J. Sventek, S. Heeps, S. Strowes, K. Twidle, S.-L. Keoh, and A. Schaeffer-Filho, “AMUSE: autonomic management of ubiquitous systems for e-health,” Concurrency and Computation: Practice and Experience, John Wiley, vol. 20(3), pp. 277– 295, May 2008.

A. Schaeffer-Filho, P. Smith, and A. Mauthe, “Policy-driven network simulation: a resilience case study,” in 26th ACM Symposium on Applied Computing (SAC). Taichung, Taiwan: ACM, March 2011, pp. 492–497.

A. Schaeffer-Filho, “Supporting management interaction and composition of self-managed cells,” Ph.D. dissertation, Imperial College London, 2009.

Sethi，M. S., A. S.: A survey of fault localization techniques in computer networks, vol.53, pp.165--194, Science of Computer Programming, 2004

A. Schaeffer-Filho, A. Mauthe, D. Hutchison, P. Smith, Y. Yu, and M. Fry, “PReSET: A Toolset for the Evaluation of Network Resilience Strategies”, The 13th IFIP/IEEE International Symposium on Integrated Network Management(IM), Belgium, 2013

P. Smith, A. Schaeffer-Filho, A. Ali, M. Scholler, N. Kheir, A. Mauthe, and D. Hutchison, “Strategies for network resilience: Capitalising on policies,” in 4th International Conference on Autonomous Infrastructure, Management and Security (AIMS). Zurich, Switzerland: LNCS, June 2010, pp. 118–122.

T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley, “ns-3 project goals,” in WNS2 ’06: Proceeding from the 2006 workshop on ns-2: the IP network simulator. New York, NY, USA: ACM, 2006, p. 13.

Qiu, L., Zhang, Y., Wang, F., Han, M. K., Mahajan, R.: A general model of wireless interference, pp. 171--182, ACM, NY, USA, 2007

Kotz, D., Newport, C., Gray, R. S., Liu, J., Yuan, Y., Elliott, C.: Experimental evaluation of wireless simulation assumptions, Technical Report, Dartmouth College, 2004.

Fessi A., Plattner, B., et al.: Resilience and Survivability for future networking: framework, mechanisms, and experimental evaluation, ResumeNet Deliverable D1.5, 2009

Doerr, C., Smith, P., et al., Resilience and Survivability for future networking: framework, mechanisms, and experimental evaluation, ResumeNet Deliverable D2.3a, 2010

Mahoney, M. V, Chan, P. K.: An analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for network anomaly detection, In Proc. 6th Intl. Symp. on Recent Advances in Intrusion Detection, 2003

Tavallaee, M., Bagheri, E., Lu, W., Ghorbani, A.: A Detailed Analysis of the KDD CUP 99 Data Set, IEEE (2009).

Brugger, T.: KDD Cup ’ 99 dataset considered harmful, White Paper, Department of Computer Science, University of California Davis (2007)

Weingartner, E., vom Lehn, H., Wehrle, K: A performance comparison of recent network simulators, pp.1--5, IEEE, Germany (2009)

A. Varga and R. Hornig, “An overview of the OMNeT++ simulation environment,” in SIMUTools ’08: Proceedings of the 1st International Conference on Simulation Tools and Techniques. Marseille, France: ICST, 2008, pp. 1–10.

K. Twidle, E. Lupu, N. Dulay, and M. Sloman, “Ponder2 - a policy environment for autonomous pervasive systems,” in POLICY ’08: IEEE Workshop on Policies for Distributed Systems and Networks. Palisades, NY, USA: IEEE Computer Society, 2008, pp. 245–246.

D. Cavin, Y. Sasson, and A. Schiper, “On the accuracy of MANET simulators,” in POMC ’02: Proceedings of the second ACM international workshop on Principles of mobile computing. New York, NY, USA: ACM, 2002, pp. 38–43.

T. Gamer and C. P. Mayer, “Large-scale evaluation of distributed attack detection,” in Simutools ’09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 1–8.

Weingartner, E., vom Lehn, H., Wehrle, K: A performance comparison of recent network simulators, pp.1--5, IEEE, Germany, 2009

Kargl, F., Schoch, E.: Simulation of MANETs: A qualitative comparison between JiST/SWANS and NS-2, International Workshop on MobiEval, 2007

Varga, A.: OMNeT++ User Manual, http://www.omnetpp.org/doc/manual/usman.html

Mell, P., Hu, V., Lipmann, R., et al.: An Overview of Issues in Testing Intrusion Detection Systems, Technical Report, National Institute of Standard and Technology, 2003

T. Gamer and M. Scharf, “Realistic Simulation Environments for IP-based Networks”, In Proceedings of the OMNeT++Workshop,Marseille, France, March 2008.

C. P. Mayer and T. Gamer, “Integrating real world applications into OMNeT++,” Institute of Telematics, Universit¨at Karlsruhe (TH), Telematics Technical Report TM-2008-2, Feb. 2008. [Online]. Available: http://doc.tm.uka.de/2008/TM-2008-2.pdf

Cetinkaya, E. K., Jabbar, A., Mahmood, R., Sterbenz, J. P. G.: Modelling Network Attacks and Challenges: A Simulation-based Approach, EDCC, Valencia, Spain, 2010

A. Varga, “The OMNET++ discrete event simulation system,” in Proceedings of the European Simulation Multiconference. Prague, Czech Republic: SCS – European Publishing House, June 2001, pp. 319–324.

D. Dittrich, “The “Tribe Flood Network” distributed denial of service attack tool,” University of Washington, http://staff.washington.edu/dittrich/misc/tfn.analysis.txt, Technical Report, 1999.

M. Charalambides, P. Flegkas, G. Pavlou, A. Bandara, E. Lupu, A. Russo, N. Dulay, M. Sloman, and J. Rubio-Loyola, “Policy conflict analysis for quality of service management,” in Proceedings of the 6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY). IEEE Computer Society, Washington, DC, USA 2005, pp. 99–108.

M. Charalambides, P. Flegkas, G. Pavlou, J. Rubio-Loyola, A. Bandara, E. Lupu, A. Russo, M. Sloman, and N. Dulay, “Dynamic policy analysis and conflict resolution for diffserv quality of service management,” in Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS), 2006.

M. Steinder and A. S. Sethi, “Probabilistic fault diagnosis in communication systems through incremental hypothesis updating,” Computer Networks, vol. 45, no. 4, pp. 537–562, 2004.

C. E. Shannon, “A mathematical theory of communication”, The Bell System Technical Journal, vol. 27, pp. 379_423, July, October 1948.

S. Chen and Y. Tang,”DAW: A distributed antiworm system,” IEEE Transactions on Parallel and Distributed Systems”, pp. 893-906, 2007.

D. Brauckhoff, K. Salamatian, M. May, “A signal processing view on packet sampling and anomaly detection”, INFOCOM, USA: IEEE Press, pp. 713-721, 2010.

C. Zesheng et al., "An Information-Theoretic View of Network-Aware Malware Attacks", IEEE Transactions on Information Forensics and Security,pp.530-541, 2009.

PandLabs, "PandaLabs Annual Report 2011 Summary," Panda Security, Tech. Rep., 2011. [Online]. Available: http://press.pandasecurity.com/wp-content/uploads/2012/01/Annual-Report-PandaLabs-2011.pdf

E. Glatz and X. Dimitropoulos. Classifying Internet One-way Traffic. In Proceedings of ACM SIGCOMM Internet Measurement Conference, 2012.

K. Munivara Prasad ,A. Rama Mohan Reddy, K.venugopal Rao,An efficient detection of flooding attacks to Internet Threat Monitors (ITM) using entropy variations under low traffic,IEEE Third International Conference on Computing Communication & Networking Technologies (ICCCNT), 2012 , 26-28 July 2012,pages-1-11.

Devi, S. Renuka, and P. Yogesh. "Detection of Application Layer Ddos Attacks Using Information Theory Based Metrics", 2012

Jeyanthi, N. and Iyengar, N. C. S. N. , "An entropy based approach to detect and distinguish DDoS attacks from flash crowds in VoIP networks". International Journal of Network Security, 14, 257–269, 2012

Shui Yu, Weijia Jia, Song Guo, Yong Xiang, and Feilong Tang "Discriminating DDoS Attacks from Flash Crowds Using Flow Correlation Coefficient" IEEE Transactions On Parallel and Distributed Systems, Vol. 23, No. 6, June 2012

Muhammad Aamir, and Muhammad Arif, "Study and Performance Evaluation on Recent DDoS Trends of Attack & Defense", I.J. Information Technology and Computer Sciencem, Jul 2013

Jeevaakatiravan, D.Hemapriyadharshini, C.Chellapan, R.Dhanalakshmi, "A Novel Approach For Detecting Smart Camouflaging Worm", Journal of Theoretical and Applied Information Technology, Jan 2013

Bhagvan Krishna Gupta, Ankit Mundra, Nitin Rakesh, "Failure Detection and Recovery in Hierarchical Network Using FTN Approach", IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 2, No 2, March 2013

[Web1] SWITCHlan Backbone 2012, http://www.switch.ch/export/sites/default/uni/network/lan/_files/SWITCHlan-Karte2012.pdf

Appendix

A. Code for LinkMonitor MO

```
/**
 * @file LinkMonitor.cc
 * @brief An implementation of the Link Monitor Managed Object.
 * @author Tina Yu (tinayu@it.usyd.edu.au)
* */


// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU Lesser General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public License
// along with this program; if not, see <http://www.gnu.org/licenses/>.
```

```
#include "LinkMonitor.h"
#include <sstream>
#include "EventPublisher.h"
#include "MechanismExporter.h"
#include <string>
#include "IPDatagram_m.h"

Register_Class(LinkMonitor);

bool triggered;

LinkMonitor::LinkMonitor(const char *name) :
cDatarateChannel(name)
{
    batchSize = 10;
    maxInterval = 0.1;
    threshold = 100;
    intvlTime = 100;
    cnt = 0;
    value = 0;
    triggered_count = 0;
    numPackets = 0;
    numBits = 0;
    triggered = false;
```

```
    intvlStartTime = intvlLastPkTime = 0;
    intvlNumPackets = benignNumPackets = maliciousNumPackets =
intvlNumBits = 0;
}

LinkMonitor::~LinkMonitor()
{
        delete out_vectors["N"];
        delete out_vectors["V"];
        delete out_vectors["p"];
        delete out_vectors["b"];
        delete out_vectors["m"];
        delete out_vectors["b"];
        delete out_vectors["P"];
        delete out_vectors["B"];
        delete out_vectors["T"];
}

bool LinkMonitor::initializeChannel(int stage) {

        cDatarateChannel::initializeChannel(stage);

        const char *fmt = this->par("format");
        char buf[200];
```

```
char *p = buf;
std::stringstream n;
for (const char *fp = fmt; *fp && buf+200-p>20; fp++)
{
    n.str("");
    switch (*fp)
    {
        case 'N': // number of packets
            n << "Number of Packets from " << this->getSourceGate()->getOwnerModule()->getFullPath();
            this->out_vectors.insert(std::make_pair<const char*,cOutVector*>("N",new cOutVector(n.str().c_str())));

            break;
        case 'V': // volume (in bytes)
            n << "Volume from " << this->getSourceGate()->getOwnerModule()->getFullPath();
            this->out_vectors.insert(std::make_pair<const char*,cOutVector*>("V",new cOutVector(n.str().c_str())));
            break;
        case 'p': // current packet/sec
            n << "current overall packet/sec from " << this->getSourceGate()->getOwnerModule()->getFullPath();
            this->out_vectors.insert(std::make_pair<const char*,cOutVector*>("p",new cOutVector(n.str().c_str())));
```

```
                n.str("");
                n << "current malicious packet/sec from " << this->getSourceGate()->getOwnerModule()->getFullPath();
                this->out_vectors.insert(std::make_pair<const char*,cOutVector*> ("m",new cOutVector(n.str().c_str())));
                n.str("");
                n << "current benign packet/sec from " << this->getSourceGate()->getOwnerModule()->getFullPath();
                this->out_vectors.insert(std::make_pair<const char*,cOutVector*> ("b",new cOutVector(n.str().c_str())));
                break;
            case 'b': // current bandwidth
                n << "CurrentBandwidth from " << this->getSourceGate()->getOwnerModule()->getFullPath();
                this->out_vectors.insert(std::make_pair<const char*,cOutVector*> ("b",new cOutVector(n.str().c_str())));
                break;
            case 'P': // average packet/sec on [0,now)
                n << "average packet/sec from " << this->getSourceGate()->getOwnerModule()->getFullPath();
                this->out_vectors.insert(std::make_pair<const char*,cOutVector*> ("P",new cOutVector(n.str().c_str())));
                break;
            case 'B': // average bandwidth on [0,now)
                n << "AverageBandwidth from " << this->getSourceGate()->getOwnerModule()->getFullPath();
```

```
                this->out_vectors.insert(std::make_pair<const char*,cOutVector*>("B",new cOutVector(n.str().c_str())));
                break;
            case 'T': // threashold checking
                n << "Value from" << this->getSourceGate()->getOwnerModule()->getFullPath();
                this->out_vectors.insert(std::make_pair<const char*,cOutVector*>("T",new cOutVector(n.str().c_str())));
                break;
            default:
                *p++ = *fp;
        }
    }

    return false;
}
#if OMNETPP_VERSION>0x0400
void LinkMonitor::processMessage(cMessage *msg, simtime_t t, result_t& result)
{
  cDatarateChannel::processMessage(msg, t,result);
  if (dynamic_cast<cPacket*>(msg)) {

    // count packets and bits
    numPackets++;
```

```
            numBits += ((cPacket*)msg)->getBitLength();


            if (intvlNumPackets >= batchSize || t-intvlStartTime >=
maxInterval)
                beginNewInterval(t);


            intvlNumPackets++;
            string name = msg->getName();
            if ((name.compare("udp_flood") == 0) ||
(name.compare("SYN-Flood") == 0) || (name.compare("ping_flood")
== 0)) {
                maliciousNumPackets++;
            } else {
                benignNumPackets++;
                //cerr << name << endl;
            }
            intvlNumBits += ((cPacket*)msg)->getBitLength();
            intvlLastPkTime = t;


            updateDisplay();
    }
}
#else
bool LinkMonitor::deliver(cMessage *msg, simtime_t t)
{
    bool ret = cDatarateChannel::deliver(msg, t);
```

```
    if (dynamic_cast<cPacket*>(msg)) {

            numPackets++;
            numBits += ((cPacket*)msg)->getBitLength();

            if (intvlNumPackets >= batchSize || t-intvlStartTime >=
maxInterval)
                beginNewInterval(t);

            intvlNumPackets++;
            intvlNumBits += ((cPacket*)msg)->getBitLength();
            intvlLastPkTime = t;

            // update display
            updateDisplay();
    }
    return ret;
}
#endif

void LinkMonitor::beginNewInterval(simtime_t now) {
    simtime_t duration = now - intvlStartTime;

    currentBitPerSec = intvlNumBits/duration;
```

```
    currentPkPerSec = intvlNumPackets/duration;
    maliciousPkPerSec = maliciousNumPackets/duration;
    benignPkPerSec = benignNumPackets/duration;

    intvlStartTime = now;
    intvlNumPackets = maliciousNumPackets = benignNumPackets =
intvlNumBits = 0;
}

void LinkMonitor::updateDisplay()
{
    const char *fmt = this->par("format");

    char buf[200];
    char *p = buf;
    simtime_t tt = getTransmissionFinishTime();
    if (tt==0) tt = simTime();
    double bps = (tt==0) ? 0 : numBits/tt;
    double bytes;
    for (const char *fp = fmt; *fp && buf+200-p>20; fp++)
    {
        switch (*fp)
        {
            case 'N': // number of packets
                p += sprintf(p, "N: %ld", numPackets);
```

```
            this->out_vectors["N"]->record(numPackets);
            EV << "NUMBER OF PACKETS " << numPackets << "\n";
            break;
        case 'V': // volume (in bytes)
            bytes = floor(numBits/8);
            this->out_vectors["V"]->record(bytes);
            if (bytes<1024) {
                p += sprintf(p, "V: %gB", bytes);
                EV << "VOLUME " << bytes << "B\n";
            } else if (bytes<1024*1024) {
                p += sprintf(p, "V: %.3gKB", bytes/1024);
                EV << "VOLUME " << bytes/1024 << "KB\n";
            } else {
                p += sprintf(p, "V: %.3gMB", bytes/1024/1024);
                EV << "VOLUME " << bytes/1024/1024 << "MB\n";
            }
            break;


        case 'p': // current packet/sec
              p += sprintf(p, "p: %.3gpps", currentPkPerSec);
              if (currentPkPerSec >= 0 ) {
                      this->out_vectors["p"]-
>record(currentPkPerSec);this->out_vectors["p"]-
>record(currentPkPerSec);
```

```
                    this->out_vectors["b"]-
>record(benignPkPerSec);this->out_vectors["b"]-
>record(benignPkPerSec);

                    this->out_vectors["m"]-
>record(maliciousPkPerSec);this->out_vectors["m"]-
>record(maliciousPkPerSec);}


            EV << "CURRENT " << currentPkPerSec << "packet/sec
\n";

                        if ((currentPkPerSec>=3000) && triggered ==
false) {

                            triggered_count++;

                            if (triggered_count == 100) {

                                triggered = true;

                                vector<string> values;

                                values.push_back(this-
>getFullPath());

                                std::string s;

                                std::stringstream out;

                                out    <<    this->getSourceGate()-
>getNextGate()->getIndex();

                                s = out.str();

                                values.push_back(s);

                                EventPublisher publisher;

                                publisher.publish("load",values);

                            }

                        }

                    break;
```

```
        case 'b': // current bandwidth
             this->out_vectors["b"]->record(currentBitPerSec);

            if (currentBitPerSec<1000000){
                p += sprintf(p, "b: %.3gk", currentBitPerSec/1000);
                EV << "CURRENT BW " << currentBitPerSec/1000 <<
"k\n";
            }
            else{
                p += sprintf(p, "b: %.3gM", currentBitPerSec/1000000);
                EV << "CURRENT BW " << currentBitPerSec/1000000
<< "M\n";
            }
            break;

        case 'P': // average packet/sec on [0,now)
            p += sprintf(p, "Avg P: %.3gpps", tt==0 ? 0 :
numPackets/tt);
            this->out_vectors["P"]->record(tt==0 ? 0 : numPackets/tt);
            EV << "AVERAGE " << numPackets/tt<< "packet/sec \n";

            break;

        case 'T': // check threshold
             p += sprintf(p, "T: %d", value);
```

```
    this->out_vectors["T"]->record(value);
    if (currentPkPerSec > threshold){
            cnt += 1;
            if (cnt == intvlTime)
                    value = 1;
    }
    else {
            cnt=0;
            value = 0;
    }

  EV << "VALUE " << value << " \n";
  break;

case 'B': // average bandwidth on [0,now)
    this->out_vectors["B"]->record(bps);
    bubble("this is B");
  if (bps<1000000){
     p += sprintf(p, "Avg B: %.3gk", bps/1000);
       EV << "AVERAGE BW " << bps/1000 << "k\n";
  }
  else{
     p += sprintf(p, "Avg B: %.3gM", bps/1000000);
     EV << "AVERAGE BW " << bps/1000000 << "M\n";
  }
```

```
                break;

            default:
                *p++ = *fp;
        }
    }
    *p = '\0';

    getSourceGate()->getDisplayString().setTagArg("t", 0, buf);

}
```

B. Code for DDoS Resilience Strategy IDSModule

```
/**
 * @file IDSModule.cc
 * @brief An implementation of the Intrusion Detection System Managed Object.
 * @author Tina Yu (tinayu@it.usyd.edu.au)
 * @date 22/09/10
 * */

// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU Lesser General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public License
// along with this program; if not, see <http://www.gnu.org/licenses/>.

#include "IDSModule.h"
```

```
#include "IPDatagram_m.h"
#include <iostream>
#include <string>
#include "TCPSegment.h"
#include "UDPPacket.h"
#include "SCTPMessage_m.h"
#include "UDPSocket.h"
#include "IPAddressResolver.h"
#include "IDS_message_m.h"
#include <map>
#include <tr1/unordered_map>
#include "MechanismExporter.h"
#include <string>
#include "EventPublisher.h"

using namespace std;

bool report;

Define_Module ( IDSModule);

IDS::~IDS() {

}
```

```
IDS::IDS() {
        destination_ip = "0.0.0.0";
        packet_count = 1;
        seen = false;
}

typedef std::tr1::unordered_map<string, IDS> IDSTable;
IDSTable tableIDS;

//Initialise
void IDSModule::initialize(int stage) {
        EV<< "IDS initialised!\n";
        packetCount = 0;
        mechanism_exporter.registerIDS(this, this->getFullPath());
        report = false;
}

void IDSModule::handleMessage(cMessage *message) {
        //Get sim time
        simtime_t current_time = simTime();



                if (message->isPacket()) {
                        packetCount++;
```

```
                EV<< "\nTotal number of packets: " << packetCount << "\n";
                IDS new_IDS;
                //Cast to a packet
                cPacket *packet = check_and_cast<cPacket *>(message);
                //Cast to a datagram
                IPDatagram *datagram = check_and_cast<IPDatagram *>(packet);
                //Get destination IP
                IPAddress destination_ip = datagram->getDestAddress();
                new_IDS.destination_ip=destination_ip;

                string destination_string = destination_ip.str();

                string key = destination_string;

                IDSTable::iterator iter;

                iter = tableIDS.find(key);
                if (iter != tableIDS.end()) {
                        EV<< "Repeat IP: "<< key;
                        //Increase packet count
                        iter->second.packet_count++;
                        unsigned int c = iter->second.packet_count;
```

```
                        float prob = float(c)/float(packetCount);
                        if (prob > 0.7) {

                                simtime_t detection_time = simTime();

                                char * current_ip=(char *)(iter->first.c_str());
                                static char victim_ip[10];

                                if (strcmp(current_ip, victim_ip)!=0) {
                                        strcpy(victim_ip, current_ip);
                                }
                                else {

                                        if ((report == true) && iter->second.seen == false) {
                                                iter->second.seen = true;
                                                vector<string> values;
                                                values.push_back(iter->first);
                                                EventPublisher publisher;

        publisher.publish("intrusion",values);
                                        }
                                        EV << "Found Victim!\n" << "Victim IP: " << iter->first <<"\n";
                                }
```

```
                    }
                } else if (iter == tableIDS.end()) {

        tableIDS.insert( std::make_pair(key,new_IDS) );
                }
            }

        send(message,"distack");
}

void IDSModule::setReport(string action) {
        if (action.compare("on") == 0) {
            report = true;
        } else if (action.compare("off") == 0) {
            report = false;
        }
}

void IDSModule::finish() {
        EV<< "IDS finished!\n";
}
```