Available online at www.sciencedirect.com
ScienceDirect
journal homepage: www.elsevier.com/locate/cose
Computers & Security

A framework for generating realistic traffic for Distributed Denial-of-Service attacks and Flash Events

Sajal Bhatia[*], *Desmond Schmidt*, *George Mohay*, *Alan Tickle*

Information Security Discipline, Science and Engineering Faculty, Queensland University of Technology, Brisbane, Queensland, Australia

ARTICLE INFO

Article history:
Received 21 July 2013
Received in revised form
30 September 2013
Accepted 16 November 2013

Keywords:
Synthetic traffic generation
DDoS attacks
Flash Events
IP-aliasing
Testbed framework

ABSTRACT

An intrinsic challenge associated with evaluating proposed techniques for detecting Distributed Denial-of-Service (DDoS) attacks and distinguishing them from Flash Events (FEs) is the extreme scarcity of publicly available real-word traffic traces. Those available are either heavily anonymised or too old to accurately reflect the current trends in DDoS attacks and FEs. This paper proposes a traffic generation and testbed framework for synthetically generating different types of realistic DDoS attacks, FEs and other benign traffic traces, and monitoring their effects on the target. Using only modest hardware resources, the proposed framework, consisting of a customised software traffic generator, 'Botloader', is capable of generating a configurable mix of two-way traffic, for emulating either large-scale DDoS attacks, FEs or benign traffic traces that are experimentally reproducible. Botloader uses IP-aliasing, a well-known technique available on most computing platforms, to create thousands of interactive UDP/TCP endpoints on a single computer, each bound to a unique IP-address, to emulate large numbers of simultaneous attackers or benign clients.

© 2013 Elsevier Ltd. All rights reserved.

1. Introduction

In spite of considerable research conducted over the past decade, Distributed Denial-of-Service (DDoS) attacks in various guises continue to constitute a pernicious threat within the Internet community (Nazario, 2008). These attacks, using the same basic *modus operandi* as some of the earliest known attacks, still exist; the only difference being their magnitude, complexity and frequency, which have all increased dramatically. Hence, developing techniques to accurately and reliably detect DDoS attacks, and differentiate them from Flash Events (FEs) – in which a similar high-load is generated from a large number of benign clients accessing the server simultaneously – remains an active area of research. However, the requisite research in developing such techniques is hampered by the extreme scarcity of *recent* and *realistic* public domain datasets representing real traffic traces, whether attack or benign, for testing and evaluation purposes.

This paper addresses this challenge by proposing a traffic generation and testbed framework to synthetically generate a variety of realistic DDoS attacks, FEs, and other benign traffic traces. The proposed framework uses modest hardware and a software traffic generator, called Botloader, which exploits IP-

* *Corresponding author*.
E-mail addresses: s.bhatia@qut.edu.au (S. Bhatia), desmond.schmidt@qut.edu.au (D. Schmidt), g.mohay@qut.edu.au (G. Mohay), ab.tickle@qut.edu.au (A. Tickle).

0167-4048/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cose.2013.11.005

aliasing, a well-known technique available on most computing platforms for associating multiple IP addresses with a single Network Interface Card (NIC).

The remainder of the paper is structured as follows. Section 2 provides an overview of the various techniques so far proposed for obtaining or generating realistic DDoS and FE datasets, focusing on the limitations of the existing public domain datasets, and on problems intrinsic to the generation of synthetic traffic in the laboratory. Section 3 discusses the design of the traffic generation and testbed framework, consisting of a hardware testbed and software traffic generator – Botloader. Section 4 discusses the implementation details of the experimental testbed environment used for traffic generation. Section 5 presents the emulation results of a real-world DDoS attack and an FE, and evaluates how closely they mimic real traffic traces. Finally, Section 6 summarises the work and describes the future directions for this research.

2. Background and related work

Over the years DDoS attacks have evolved from a naïve low-scale to a more sophisticated large-scale problem. During this period there has been considerable research and development into attack detection strategies, but only limited research on testing these techniques against realistic data. One of the key reasons for this has been the legal and privacy issues associated with sharing captured data. As a result, the majority of published work on DDoS attack detection is evaluated by: (a) replaying publicly available datasets, (b) generating traffic using open-source traffic generators, and (c) using testbeds based on simulation, emulation or direct physical representation strategies. Each of these approaches has its own technical limitations, may not reflect current trends in DDoS attack scenarios, and may produce misleading results for the evaluation of the proposed detection techniques. Each of these approaches will now be discussed.

2.1. *Replaying traffic traces*

One obvious way to test and evaluate a workable solution for detecting DDoS attacks is to replay traces of real attacks (Ahmed et al., 2010; Buchanan et al., 2011). Although DDoS attacks have become common, real-world attack traces in the public domain are rare, because the sharing of captured live data (both attack and normal) is limited by significant legal and privacy issues.

The KDD Cup 1999 Data is one of the most referenced and possibly the only reliably-labelled dataset available in the public domain (Hettich and Bay, 1999). Use of the KDD dataset completely eliminates any legal and privacy-related issues that arise when using any *real-world* data. However, due to its age (1999) and other limitations e.g., its prime usage is for evaluating signature-based intrusion detection systems (McHugh, 2000; Tavallaee et al., 2009), it is not really suitable for testing DDoS detection and mitigation mechanisms. The set of network traces from Waikato Internet Trace Storage Project (Waikato Applied Network Dynamic Research Group) is another widely referenced dataset within the DDoS detection domain. However, these datasets have their own limitations: IP addresses are pseudonymised, packets are truncated 20 bytes after the transport header, and any retained payloads of UDP packets (except the DNS) are zeroed. A more recent addition has been the 'CAIDA DDoS Attack 2007 Dataset' containing an approximately one hour traffic trace of a DDoS attack which aimed at consuming the computing resource of the targeted server (Hick et al., 2007). However, the available traces have had their IP addresses pseudonymised using CryptoPAn and their payloads removed, thereby limiting their usability.

In the FE domain, a very limited number of datasets representing different types of FEs are publicly available. The 1998 FIFA World Cup Dataset, provided by the Internet Traffic Archive, is the most widely used and probably the only predictable FE[1] dataset available in the public domain.

In addition to these specific limitations, these datasets also share some characteristics which greatly limit their usability. First, excepting the CAIDA DDoS Attack 2007 Dataset, the other datasets are old and obsolete in the context of continuously evolving DDoS attacks with complex attack vectors. Secondly, most of the available datasets (except KDD and DARPA) are not labelled, which makes it difficult to cleanly filter out the attack from benign background traffic, and increases the training difficulty for detection strategies that rely on machine learning algorithms. Thirdly, even though the target addresses of the packets in these datasets can be altered and subsequently directed to a test machine while replaying, the real effects of the network trace still cannot be reproduced due to the non-availability of the services addressed in the original trace. Fourthly, the missing payloads completely eliminate the possibility of developing any *content-based* analysis technique. Lastly, the majority of FE datasets are web-server logs in Common Log Format, thereby making them difficult to use, e.g. it is not possible to replay them over a network to test FE detection algorithms.

2.2. *Traffic generators*

Given the limitations of direct replay of captured traffic traces, artificial generation would seem to be the only practical way to generate the required traffic, both attack and benign. Unfortunately both hardware and software traffic generators are far from being ideal tools for simulating such attacks.

The hardware traffic generators can do much the same as their software counterparts, but at higher speed and at greater cost. Hardware traffic generators like the SmartBits 600 (SmartBits, 2001) can artificially generate configurable flooding attacks, but they fail to interact with the application at the target machine. Our experiments with the SmartBits generator highlighted significant differences in how the computers and the other networking devices like switches and routers responded to hardware-generated and software-generated

[1] A predictable FE is one whose expected occurrence is known *a priori*, thus allowing network administrators to prepare for it by using various provisioning and load-balancing techniques. Some popular examples are product releases (e.g. by hi-tech companies like Apple), widely followed sporting events such as the Olympics, or online play-along websites for popular television programs. A detailed classification of FEs can be obtained in the paper by Bhatia et al. (2012).

network traffic. The number of packets being dropped by the kernel was found to be much higher for hardware-generated network traffic. On the other hand, most of the software-based traffic generators available in the public domain are based on simple client–server models and are limited by the number of different scenarios they are capable of simulating. Moreover, most of these software tools were written long ago and thus need serious modifications before they can be of any practical use.

One weakness shared by hardware traffic generators and a majority of software based traffic generators is their inability to conduct any meaningful interactions with the target. Their sole aim is to deposit packets onto the wire that conform to the given traffic profile, as expressed by parameters like traffic volume, traffic distribution, packet size, and protocol type. The following open-source software traffic generators were investigated: D-ITG (Botta et al., 2012), Harpoon (Sommers et al., 2004), fudp,[2] Hping (Sanfilippo, 2008) and curl-loader (Iakobashvili and Moser, 2009). Curl-loader, discussed below, was extensively tested and used to conduct preliminary experiments for generating datasets with the desired characteristics.

2.2.1. *Curl-loader*

Curl-loader is a C-based open-source tool designed to simulate the behaviour of a large number of HTTP/HTTPS and FTP/FTPS clients, each with its own unique source IP address. Curl-loader has the ability to generate large numbers of 'virtual clients' (VCs), each having their own valid unique source IP and shared MAC address. In order to generate synthetic network traffic, comprising both malicious and normal data on the experimental testbed infrastructure, the scalability of this approach was tested and was seen to be governed by (a) processing capacity of the individual machines (i.e. the number and speed of CPUs and the amount of memory); and (b) the number of distinct platforms available in the experimental testbed.

Curl-loader places heavy demands on the host systems' resources (CPU and memory) and proved far from an ideal tool for generating synthetic traffic originating from a large number of source IPs at the desired rate. The tool also had to be modified to incorporate randomness in the source IP addresses and to create various attack profiles. These modifications led to instabilities in performance. This, combined by the need for further modifications to incorporate different types of attacks, eventually led to the abandonment of this approach.

The investigation of Curl-loader led to the use of IP aliasing with Wget (Niksic, 1998) (both available on most computing platforms) for synthetic traffic generation, which is now discussed.

2.2.2. *IP-aliasing with wget*

IP aliasing is a well-known technique, available on most computing platforms, for assigning a large number of distinct IPs to a single hardware (Network Interface Card or NIC) address. On Linux and BSD, for example, thousands of unique IP addresses can be assigned to a single Ethernet card. IP-aliasing on the Linux platform can be achieved via the *ifconfig* utility. Using IP-aliasing, traffic originating from a single host can be divided into multiple *network-flows*,[3] each having a unique value for its source IP address, and so overall the traffic can be made to appear (to the target) to be coming from many separate machines.

GNU Wget (Niksic, 1998) is a non-interactive command line utility for retrieving files using the HTTP, HTTPS and FTP protocols. As wget is a non-interactive tool, it can be easily integrated with custom scripts, cron jobs and other command line utilities available in Linux without the support of X-windows. Therefore, when used in combination with IP-aliasing, wget can potentially create the appearance of an array of machines ('bots' for DDoS attacks and 'legitimate clients' for FEs), each with a unique source IP address, but sharing the hardware address of the physical interface. This array of machines can then be used to synthetically generate realistic DDoS attacks and FEs.

The approach of using wget with IP-aliasing provided a method for crafting application layer DDoS attacks such as the HTTP flooding attack. However, the time it took to create aliases increased with the number of requested aliases. Also, an increase in the number of aliases resulted in a corresponding increase in the host machines' CPU and memory consumption. In addition to resource consumption constraints, the approach was also limited by the variety of attacks (only HTTP based) that it was capable of generating, thereby lacking the desired flexibility. The perceived drawbacks with this technique, as with the use of curl-loader, led to the development of a customised software traffic generator called Botloader, and is discussed in more detail in Section 3.2.

2.3. *Testbed design strategies*

An essential requirement for deploying synthetic or original traffic traces is to first have an experimental setup to play it on. Obtaining and evaluating experimental results also requires some means to manage an attack and measure its effects. These requirements thus imply the existence of some kind of testbed in addition to the traffic generation software. The reviewed literature in this field suggests that there are three commonly used testbed design strategies, viz. simulation, emulation and direct physical generation.

2.3.1. *Simulation*

Network simulators like ns–2 (McCanne et al., 1997) or Opnet (Modeler) have often been used to simulate DDoS attacks, measure their effects, and evaluate the attack detection techniques (Blackert et al., 2003; Sachdeva et al., 2010; White et al., 2002). However, the realism of a method in which the attackers, targets and network devices are all simulated has recently been called into question (Mirkovic et al., 2009). This lack of realism, combined with a slower speed of traffic replay, makes simulation an unattractive proposition for addressing the DDoS attack detection techniques, particularly for High Rate Flooding attacks.

[2] fudp – http://sourceforge.net/projects/usoft/.

[3] A TCP/IP network flow is defined as a sequence of packets from a source machine to a destination machine with unique values for source IP, source port, destination IP, destination port, and protocol.

Table 1 – Comparative analysis of various traffic generation strategies.

Characteristics	Replaying traffic traces	Traffic generators	Testbed design strategies			Proposed framework
			Simulation	Emulation	Direct physical representation	
Reconfigurability	✓	✗	✓	✓	✗	✓
Low cost	✓	✓	✓	✗	✗	✓
Monitoring capabilities	✗	✗	✗	✓	✓	✓
Variety of attacks	✗	✗	✓	✓	✓	✓
Scalability	✗	✗	✓	✗	✗	✓
Interactive traffic	✗	✗	✗	✓	✓	✓

2.3.2. *Emulation*

Emulation is a step forward in realism over simulation: real machines are used as attackers and targets. This approach gained traction with the Emulab (White et al., 2002), DETER (Benzel et al., 2009) and Planetlab (Peterson et al., 2006) testbeds. The DETER and Emulab testbeds allowed users to get access to desired machines, in a central and isolated (from the Internet) facility, whereas Planetlab, being a distributed testbed set-up, offers shared access to machines, via a VM (virtual machine) software, thus allowing user isolation.

In emulation, only the network topology is recreated in software; targets and attackers are represented by physical machines with a real Operating System (OS) and applications. The network is represented by soft-routers, and by virtual LANs. Although this is more realistic than the simulation approach, and allows the facility to be shared, the high cost of constructing and maintaining testbeds of this type seems to limit their size to around 300 machines. Real world DDoS attacks on the other hand, typically comprise at least several thousand attackers (Rajab et al., 2007).

2.3.3. *Direct physical representation*

In this technique, the desired network topology is built by physically arranging a network of routers, switches, and computers. Whilst this satisfies the criteria of realism, the main disadvantage is lack of flexibility: each experiment has to be set up by hand, and the facility can't be shared (Yu et al., 2013). On the other hand, larger scale facilities can be built, although at greater cost. For example, Calvet et al. were able to create a network of 3000 virtual machines using only 98 blade servers (Calvet et al., 2010). The experimental testbed, developed as a part of this research and explained in detail in Section 4, uses this approach as it allows a decoupling of hardware and software, and allows for more realistic experiments.

The literature reviewed above thus highlights the shortcomings of various approaches used for obtaining realistic datasets, both attack and benign, to evaluate DDoS attack detection techniques. Table 1 summarises these shortcomings and presents a comparative analysis to show how the proposed framework will overcome them.

3. Testbed framework design

This section presents the design of a simple experimental traffic generation and testbed framework, comprising of the testbed facility (the hardware platform) and the software traffic generator (Botloader), which can together be used to synthetically generate realistic network traffic for DDoS attacks and FEs.

Given the limitations of the various approaches investigated for obtaining realistic datasets, as discussed in Section 2, one obvious way around this problem was to develop such a customised traffic generation and testbed framework to meet the identified requirements. This framework had to be based on available hardware and a customised software traffic generator able to emulate a wide range of network and application layer DDoS attacks and FEs. This section describes both the hardware and traffic generation components of the testbed.

3.1. *Framework design requirements*

The following design-level requirements were identified for developing the framework.

- **Reconfigurability.** The framework is required to simulate a variety of DDoS attack, at the network and application level, as well as a variety of FEs, and as a result, it must allow for reconfigurability of the network topology.
- **Modest hardware.** Given the financial constraints of the project,[4] it was necessary to develop a testbed facility using only modest hardware. While the scale of emulated attacks can be increased by using high-end networking device, many meaningful experiments can still be conducted using modest equipment and customised software, for example, by comparing the results of different configurations.
- **Maximum resource utilisation.** A single attacking host should be able to generate sufficient traffic to impede a server of similar power from servicing its clients. Thus, orchestrating a number of such host machines should have a substantial impact on a large server used as the target host.
- **Traffic load coordination.** Each attacking host should send a relatively small portion of the overall traffic. Thus, an attacker would combine the strength of all participating hosts. This feature would cater for coordinated attacks, but at the same time would require a so-called 'control machine' for remotely issuing commands to the attacking machines and synchronising their activity.

[4] A joint research project into DoS and DDoS attacks undertaken under the auspices of the Australian and Indian Governments as part of the Australia-India Strategic Research Fund (AISRF) 2008–2012.

- **Traffic aggregation.** Since each of the participating hosts would be a standard PC with limited capabilities, in order to emulate a variety of volumetric DDoS attacks and FEs, the outgoing traffic from all of the participating hosts (attackers or legitimate clients depending on the scenario) would need to be aggregated to a single network interface before arriving at the target server.
- **Monitoring capabilities.** The framework should make it possible to monitor various intermediate nodes such as switches, firewalls, and end nodes such as applications running on the target hosts in real time.

The software traffic generator, Botloader, used with the traffic generation and testbed framework, will now be described.

3.2. *Botloader: a software traffic generator*

In order to emulate various types of DDoS attacks and FEs it was decided to make use of multiple autonomous agents. At the conceptual level, Botloader, running separately on every host machine, uses the technique of IP-aliasing to create a large number of bots (or legitimate clients in the case of FEs), each with a unique source IP, and uses them to synthetically generate network traffic based on a user-specified profile.

IP aliasing is a well-known technique for associating multiple unique IP addresses to a single network interface card, and is available on most computing platforms. This use of IP-aliasing allows the creation of a large number of different data flows originating from a single physical machine, creating the impression (to the target) as if they were coming from different physical machines.

3.2.1. *Bots, clusters and modules: the building blocks of Botloader*

Each of the unique aliased-IP addresses created by Botloader is called a *bot*. The bots are referred to as ‘compromised machines’ when emulating DDoS attacks and as ‘legitimate clients’ during FE emulation scenarios. In software terms, a bot is like an ‘object’ that consumes a small amount of memory (approximately 46 bytes, plus whatever is allocated by the operating system for its socket and its alias), and performs a fixed set of functions via an Application Programming Interface (API). All bots, irrespective of their types, have the same API, thus allowing them to be uniformly controlled by the Botloader program.

A group of bots performing similar operations is called a *cluster*. All the bots within a cluster share the same user-specified configuration, such as the IP address of the target machine, the duration of the attack or FE, the data rate (kilobytes) of the attack, or the resource map to be used in case of HTTP-based DDoS attacks and FEs. The use of clusters presents two obvious advantages: first, aggregating bots with similar functionalities avoids duplication of code, and secondly it keeps the memory requirement of the Botloader program low. Each cluster is assigned a separate execution thread to facilitate the simultaneous execution of different types of attack or background traffic. The Botloader program itself has its own main execution thread and controls the various types of clusters.

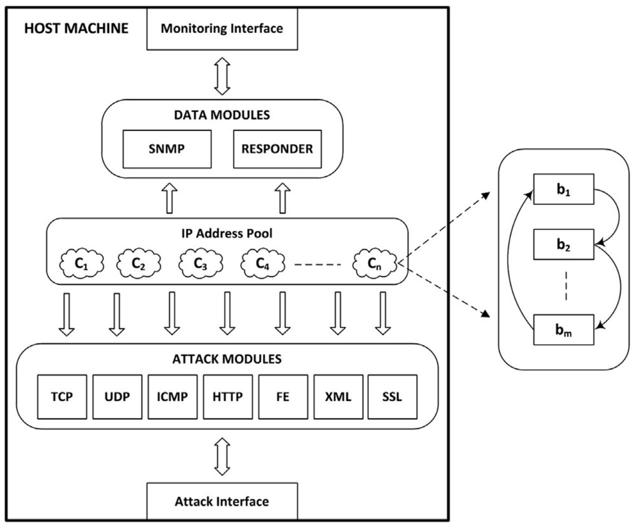


Fig. 1 – Schematics of Botloader.

A *module* is essentially a library or ‘plugin’, comprising of a set of functions that define a given attack type shared by all the bots within a cluster. Each attack type is carried out by a different module (also referred to as bot-type). On start up, Botloader reads the configuration file and searches for available modules or bot-types. Fig. 1 shows an abstract level representation of Botloader with bots, clusters, and modules. Modules come in two basic flavours: attack and data modules.

i **Attack Modules:** These modules are responsible for sending the network traffic representing a user-specified attack-type, an FE, or normal background traffic. Each bot participating in an attack uses the characteristic properties of the relevant attack module and shares it with the other bots in its cluster. However, in an attack involving multiple clusters sharing the same functionality, clusters may be configured separately, as in case of a HTTP flooding attack with background HTTP traffic at a different rate. Botloader currently caters for seven attack modules, viz., `syn_flood`, `udp_flood`, `icmp_flood`, `http_flood`, `xml_flood`, `ssl_flood`, and `fe_flood` to emulate various types of DDoS attacks and FE scenarios.

ii **Data Modules:** Apart from the modules responsible for conducting different types of attacks as specified in the configuration file, there are other modules designed for monitoring and data gathering. Two such data modules are currently implemented, viz., `snmp_bot` and `resp_bot`.
The `snmp_bot` is responsible for gathering data using the standard and custom Management Information Base (MIB) via the Simple Network Management Protocol (SNMP) protocol. The `snmp_bot` when used with other attack modules gathers information at 5 s intervals. The `snmp_bot` can gather both system-wide information such as the amount of data received on all the interfaces of the target, as well as application-specific information such as CPU and memory utilisation. The `resp_bot` is used for measuring the response time of a specific application running on the target host.

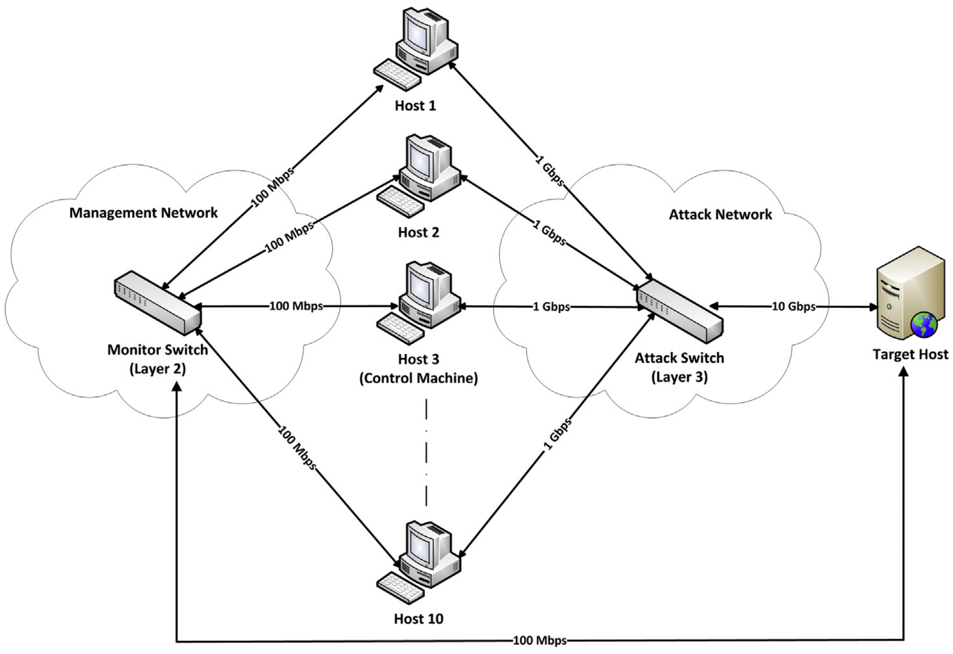


Fig. 2 – Testbed architecture.

Both of these data modules – snmp_bot and resp_bot – use a separate 'management network' to perform monitoring and data collection operations. Using a separate network channel helps to avoid any potential interference between the operation of the data and attack modules. The data gathered by the snmp_bot is saved directly on the *control machine*, in a tab-delimited spreadsheet format.

SNMP data collection during UDP flooding attacks. Even after using separate networks for attacking and monitoring the target host, UDP flooding attacks were found to interfere with the SNMP functionality, which also uses the UDP protocol for communication. In such cases a separate *monitor* program was used on the target machine. This performed the same data-gathering role as the snmp_bot, but was copied to the target at the start of the run. Once the experiment was complete, the gathered SNMP MIB data such as CPU utilisation, memory usage, etc., was copied back to the *control machine*. The monitoring program or SNMP daemon was given a higher execution priority on the target machine by increasing its 'niceness'[5] value. As the SNMP data was collected locally on the target machine, or via a separate monitoring network, any interference with the traffic on the attack network was eliminated. An insignificant amount of CPU was consumed on the target by the monitoring process which activates only once every 5 s.

3.2.2. *IP address allocation*

Based on the number of IP addresses specified in the configuration file to emulate a given traffic scenario, Botloader reduces the size of the interface's netmask and so expands the available pool of addresses (initially 254 for a class C address) to a suitably large number. For example, if the netmask is expanded to a class B network then a total of 64,770 aliased addresses can be created (65,024 for class B, less the original 254 for the class C network, so as to avoid clashes with other machines). From this pool, Botloader selects *random* IP addresses. Memory and operating system limitations currently restrict the number of aliases to around 10,000 per machine. The 'think time' or the amount of time between packets from a single IP-address can be configured by controlling the amount of data sent by each bot, and varies per attack type (Krishnamurthy and Wang, 2000).

Fig. 1 shows an instance of Botloader running on a host machine. Within each instance, the assigned IP address pool is further subdivided into n different clusters (C_i, $i \in [1, n]$) with non-overlapping IP addresses. Each of these clusters (C_i) is a collection of m bots (b_i, $i \in [1, m]$), each with a unique IP address and a network socket, and sharing the hardware address of the physical network interface. Before each data transfer cycle, bots are jumbled and then invoked in a sequential manner to send and receive traffic to/from the target. At the end of each data transfer cycle, the bots are jumbled again before repeating the cycle (see Fig. 1).

A typical traffic simulation scenario usually involves several data transfer cycles per second until the simulation times out. This ensures a *randomised* spread of source IP addresses (as seen by the target machine) that resembles the entropy of source IPs from real-world DDoS attack scenarios (Hick et al., 2007). Entropy H of a discrete random variable X is given by

$$H(X) = -\sum_{i=1}^{n} p(x_i)\log_2 p(x_i)$$

where n is the number of symbols and $p(x_i)$ is the probability of that symbol appearing in the message. The entropy H ranges

[5] Nice is program available on most Unix or Unix-like operating systems, and is used to reschedule a program with a particular processing priority. Niceness values range from −20 (highest priority) to +19 (lowest priority).

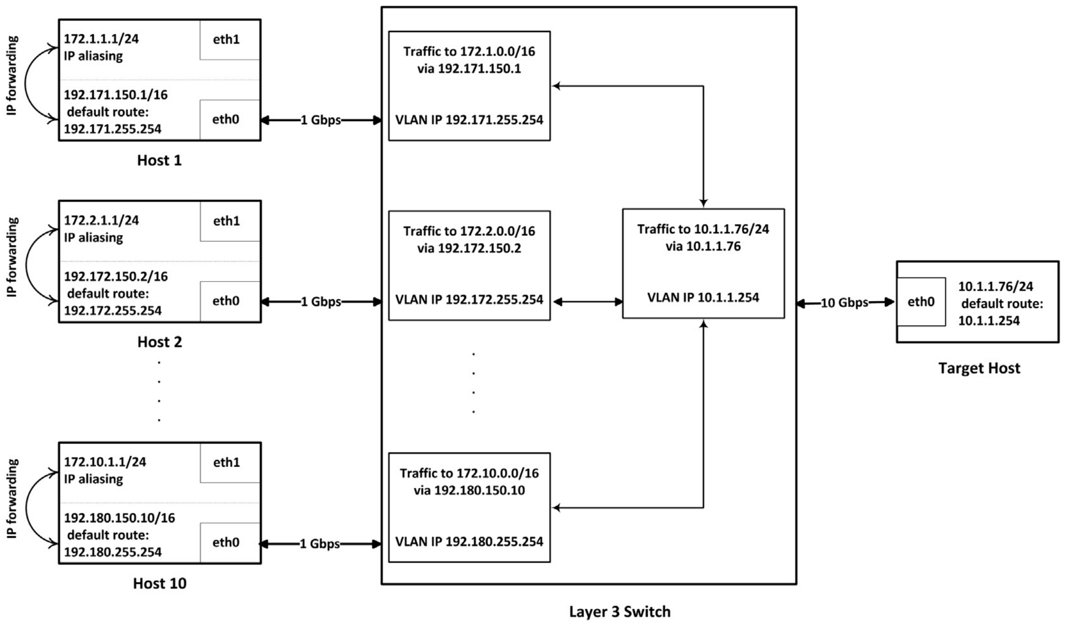


Fig. 3 – Multi VLAN with IP forwarding.

between 0 and $\log_2 n$. Dividing by $\log_2 n$ gives the normalised entropy $H_0(X)$ with values between 0 and 1, and is defined as:

$$H_0(X) = H(X)/\log_2 n$$

A normalised entropy of 1 implies that the symbol 'x_i' under consideration is purely random. Thus, $H_0 = 1$ if all bots transmit equally, and 0 if all traffic comes from a single address. Applying this formula to the source IP addresses of packets in the CAIDA DDoS Attack 2007 dataset (Hick et al., 2007) resulted in $H_0 = 0.9405$, suggesting that the experimental value of $H_0 = 1$ is realistic.

4. Testbed framework implementation

The experimental testbed environment contains two distinct networks: the *attack-network* for sending and receiving all the attack and normal background traffic, and the *management-network* for monitoring the target and collecting MIB data. The testbed architecture consists of ten dual-homed[6] host machines (Host 1–Host 10), one layer 3 switch (attack switch), one layer 2 switch (monitor switch), and a target machine. In the attack-network, all hosts machines are connected to the target via the layer 3 switch. One of the 10 host machines in the attack-network is a *control* machine used for orchestrating, configuring and managing the various DDoS attacks or FE scenarios on each host. Fig. 2 gives an abstract-level schematic of the testbed architecture.

All host machines in the attack-network use a 1 Gigabit per second (Gbps) link to connect to the attack switch, which is connected to the target machine via a 10 Gbps link. In the management-network, all connections between the target and the host machines to the monitor switch are 100 Megabits per second (Mbps). Every host machine runs an instance of the Botloader program and represents a 'group of attackers or bots' while emulating DDoS attacks, and a 'group of legitimate clients' during FEs scenarios. The control machine acts like a 'master' and issues commands to initiate the instances of Botloader in each of the participating hosts. All the traffic coming from the individual hosts is accumulated and sent through the 10 Gbps link connecting the attack switch to the target machine.

All the host machines are standard PCs with 3.0 GHz Intel Core2 processors, 4 GB of memory, an integrated 1 Gigabit (Gb) NIC, an external 100 Megabit (Mb) Ethernet card, and are running Ubuntu 10.04 Desktop as the Operating System (OS). A small number of essential services like Secure Shell (SSH), Network Time Protocol (NTP), etc. are also installed on each of the host machines. SSH was installed to copy the configuration file across all the host machines and to simultaneously initiate the Botloader instance running on each of these machines. NTP was used to perform time synchronisation across all the instances of Botloader running on every host machine. The target machine is a Dell PowerEdge R710 with six quad-core Intel Xeon 2.27 GHz processors (hyper-threaded), 32 GB of memory, and running Ubuntu 10.04 (Server) as the OS and Apache2 as the web-server. The monitor switch used in the management network is a layer 2 Cisco Catalyst 3560 switch while a Dell PowerConnect 6224 layer 3 switch is used to bind together the attack-network component of the testbed architecture.

4.1. *The ARP problem*

In order to generate realistic DDoS attacks and FEs, it is important that the emulated traffic originates from a large number of IP addresses. In the case of an FE, such as the 1st semi-final match of the 1998 World Cup, the number of IP

[6] A dual-homed host is a system equipped with two networking interfaces (NICs), generally sitting between an untrusted network and a trusted network in order to provide secure access.

addresses approaches 80,000. The problem with this requirement is that an Ethernet LAN is not designed to support that many IPs. On an Ethernet network, IP packets are sent within frames, with a source and destination MAC address. In order to identify which MAC address corresponds to a given IP address, the switch maintains an ARP table, which is of limited size. For example, the Dell switch in the testbed has a maximum ARP table size of only 1024 entries. When packets come back from the target (Fig. 2), the switch must decide which of the 10 host machines receives the reply packets. If the IP-address is not in the table it broadcasts an ARP request packet, effectively asking 'who has IP address x.x.x.x?'. While waiting for a reply, the switch has no option but to drop the packet. When the reply comes back from the correct interface card the switch has nowhere to store it if the table is full, and hence cannot forward the packet when it is later retransmitted by the target (which by default takes 2 s). So the end result is that the network fills up with ARP requests and replies rather than with transmissions of desired traffic between attackers and target.

4.2. *The work-around*

In order to avoid this flood of ARP requests, it was necessary to change the overall infrastructure from a *switching-mode* to a *routing-mode*. Fig. 3 shows the configuration of the testbed architecture. The process of turning the switch and each host machine into an effective router is achieved via the following steps:

1. The first step is to configure the main attack interface on each host machine as belonging to a separate network. On hosts 1–10, the network interfaces connected to the attack-switch (`eth0`) are assigned IP addresses in the range `192.171.150.1–192.171.150.10`, and the network is specified as Class B.
2. The second step is to re-use the secondary interface, normally used for monitoring, as the repository of all the aliased addresses. In this way they will be hidden from the switch, which can only see 11 address in total – well within the limit of switch's ARP table size.
3. The third step is to set up each host machine to act as a router by enabling IP-forwarding on every machine (on Linux). This allows the network packets on one interface (`eth1`) to be forwarded to another interface (`eth0`) and vice versa. IP-forwarding bridges the two (or several) interfaces into a single network topology and routes the packets between them extremely fast using internal pathways of the computer, without the need of any ARP (IP to MAC address mapping) information.
4. The final step is to sub-divide the switch into 11 VLANs, one of which is connected to the target, and the other 10 to the attacking host. Each attacker-VLAN has its next-hop address set to the IP address of the main attack interface. For example VLAN 1 has its IP set to 192.171.255.254 and its default route to 192.171.150.1. Likewise, the main attack interface on attacker 1 has its default route set to VLAN 1's address, 192.171.255.254. Thus all traffic between VLAN 1 and attacker 1 destined for the aliased addresses for eth1 (172.1.1.1/24) will be *routed* via the eth0 interface without the need for any ARP resolution. A similar set-up is applied to other 10 VLANs.

This solution to the ARP problem allows the creation of a large set of 'real' IP-addresses (bots or legitimate clients) that can communicate and exchange data with the target, which responds to network traffic as if from a large number of physical machines. Unlike Network Address Translation (NAT), where the source or destination IP address is modified for local delivery, the packets sent to (and received by) the target host *exactly resemble the packets sent by an external machine*. The packets have unmodified source and destination IP address and the target responds to them as if they were physically located in a real network. But all the packets remain within a controlled and isolated test environment. This set-up of the testbed has so far sustained around 80,000 different IP addresses without any ARP related issues.

5. Testbed framework performance evaluation

This section presents the performance evaluation of the traffic generation and testbed framework including the Botloader program in terms of its ability to mimic a given network trace. In order to test the effectiveness of the proposed framework, the CAIDA "DDoS Attack 2007" Dataset (Hick et al., 2007), 1998 FIFA World Cup Dataset (Arlitt and Jin, 1998), and 'Hourly Hits on Wikipedia' (Mituzas, 2011), were used as sample input traces representing a DDoS attack, predictable and an unpredictable FE respectively. These network traffic trace were preprocessed and were then given to the Botloader program in a specific format to generate similar traffic. It is acknowledged that the validation is limited by a lack of datasets representing different DDoS attacks and FE scenarios available in the public domain.

5.1. *Emulating DDoS attacks*

In order to generate network traffic, attack or benign, the Botloader program reads user specified parameters such as the duration of the emulation, the number of bots (aliased-IP addresses) to use, and the traffic or data rate (specified in packets per second and kilobytes per second), via a configuration file. So each of the attack types has its own configuration file with some features that are common across all attacks, such as the duration of the emulation and the number of bots to create, and some custom features, such as a resource file containing a list of resources to download during a HTTP flooding attack or an FE. For example, the configuration file for a HTTP flooding attack contains the following line of code, with the user specified values for various parameters:

```
http_bot 100 debug=0 time=60 port=80 res_file=web-resources.txt
rates="100" dest_ip=192.168.200.76
```

This configuration file creates `100` bots and uses them to direct HTTP traffic towards the web-server running at port `80` on `192.168.200.76`. During the emulation, the resources are downloaded at `100` kilobytes per second, spread across `60` s from the `web-resources.txt` file. In this section, emulation results for the CAIDA "DDoS Attack 2007" Dataset (Hick et al., 2007) are presented.

5.1.1. *DDoS attack dataset*

In order to emulate a real-world DDoS attack, the CAIDA "DDoS Attack 2007" Dataset (Hick et al., 2007) has been used as the base traffic model for extracting features required by the Botloader program to generate similar traffic.

The CAIDA dataset contains pseudonymised traces from a DDoS attack that occurred on August 4, 2007 for approximately one hour (20:50:08 UTC to 21:56:16). The dataset represents a type of attack in which the attacking machines attempt to block access to the target server by sending a large number of access requests, and thus consume all the available computing resources, including the bandwidth of the server's Internet connection.[7] The dataset contains the attack traces to the victim and its responses, pseudonymised using prefix-preserving CryptoPAn via a single key, and all packet payloads are removed. The entire CAIDA-dataset is divided into 5-min packet capture (pcap) files. From the complete DDoS-dataset, only the one-way attack traffic going to the victim is used for analysis.

The one-way dataset is processed to extract features such as number of source IPs, peak packet rate etc. Table 2 shows the high-level statistic of the dataset used in the emulation.

5.1.2. *Experimental results and analysis*

The incoming traffic profile (packets per second) of the CAIDA dataset is shown in Fig. 4. The one-way traffic profile clearly shows two rather distinct traffic rates, as experienced by the target victim. The first one is the packet rate during the first-half of the trace i.e. from the start of the trace until around the 1500 s mark. For emulation purposes, the packet rate during this period is referred to as the *normal packet rate*. The average packet rate for this period was approximately 385 packets per second in the original dataset. The second distinct traffic rate is seen during the second half of the trace i.e., starting from approximately 1500 s until the end of the trace. During this period, the packet rate is referred to as the *attack packet rate*, and contains approximately 125,705 packets per second on average.

In order to emulate network traffic similar to the CAIDA dataset, the normal and the attack packet rates, calculated from the original dataset, are provided to the attack configuration file of the Botloader program to generate a 5 min sample network trace. The configuration file is in the format given below where two simultaneous threads (clusters of bots) are started, with the first one for emulating normal traffic part, and the second for emulating the attack part of the original CAIDA dataset.

```
ping_bot 73 debug=0 ip_version=4 time=120 packet_size=64
rates="46200" burst_default=1000 dest_ip=10.1.1.76

ping_bot 859 debug=0 ip_version=4 time=300 packet_size=64
rates="0 0 7542300 7542300 7542300" burst_default=1000 dest_ip=10.1.1.76
```

The first thread runs for 120 s, and sends 46,200 packets during this period, thereby achieving the desired rate of roughly 385 packets per second. The second thread is started simultaneously, and runs for the entire length of the emulation (300 s). However, this thread is *dormant* for the first 120 s, as specified by the two leading 0's in the `rates` section of the configuration file. During the remaining 180 s, the threads send ICMP traffic at a rate of approximately 125,705 packets per second to the target victim (`10.1.1.76`), as specified by the `dest_ip` parameter of the configuration file.

Fig. 5 shows the results of a 5-min traffic trace synthetically generated using the traffic rates from the CAIDA dataset. The Botloader program was able to nearly match the normal and attack packet rates from the original trace, thereby effectively emulating an existing real-world DDoS attack. The emulated DDoS traffic in Fig. 5 shows a few dips at various time intervals. It is speculated that this behaviour is due to heavy traffic that the server is constantly subjected to, which leads to some packets being dropped by the kernel. However, such behaviour is also seen in the original CAIDA dataset (Fig. 4), where the incoming traffic is erratic and often drops to lower levels during the attack.

5.2. *Emulating Flash Events*

In this section, the traffic generation and testbed framework described above uses the FE model developed by Bhatia et al. (2012) to generate different types of FE scenarios. In order to generate FE traffic, the model uses information extracted from the few existing FE datasets described below, but due to their limited number and types, is limited to predictable and unpredictable FE scenarios. The emulation of synthetic FEs is also limited to traffic volume.

5.2.1. *Flash Event datasets*

Two publicly available datasets – representing a predictable FE (pFE) and an unpredictable FE (uFE) and used as base models to generate similar FE traffic – are described next.

Predictable FE (pFE) Dataset. A subset of the 1998 FIFA World Cup Dataset, provided by the Internet Traffic Archive, was used as the base pFE dataset in the evaluation (Arlitt and Jin, 1998). From the complete dataset, six hour trace of all the requests made to the 1998 FIFA World Cup websites around the 1st semi-final, was extracted and used as a pFE dataset. The dataset was first filtered to remove all traffic except HTTP GET requests. Next, only packets containing 200, 206 and 400 HTTP status codes were extracted and used as the input trace for the experiments.

In order to ensure that the same web-resources are downloaded during the emulation, as were present in the original trace, a fake directory structure was created and installed on the target machine's web-server. All the files within this directory structure used the same file name and

[7] This is a classic example of a flooding attack (ICMP flood) executed by the sending a large number of requests. This attack is different from the 'ping of death' attack which is a semantic attack executed by sending oversized ICMP (or ping) packets and thereby causing memory related issues (buffer overflow) on the target system.

Table 2 – Characteristic features of the DDoS datasets used for emulations.

Parameters	CAIDA dataset
Duration	66 min
Attack Type	Ping flood
Packet Type	ICMP
Number of Packets	359,655,765
Number of source IPs	9311
Peak packet rate (packets per second)	175,010

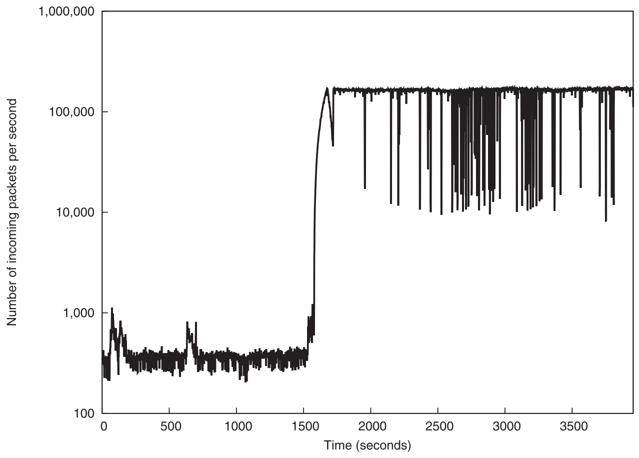


Fig. 4 – Incoming traffic profile for the CAIDA dataset.

contained the same amount of data as in the original traffic trace, although the contents were junk data.

Unpredictable FE Dataset. The 'Hourly Hits on Wikipedia' dataset containing records following Steve Jobs' death is used as the base uFE dataset for the analysis (Mituzas, 2007). This dataset was provided by Domas Mituzas, who maintains a repository of page view statistics for various Wikimedia projects. The dataset provides hourly aggregates of requests and covers a period of 43 h (starting 16:00 h UTC, 5th October 2011).

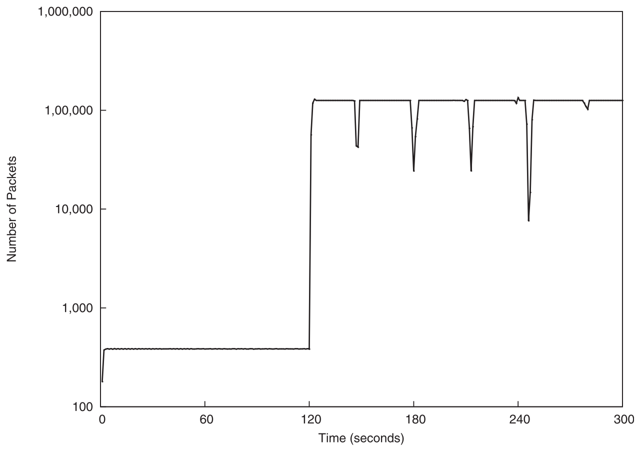


Fig. 5 – Incoming traffic profile for an emulated DDoS attack.

Table 3 – Characteristic features of the FE datasets used for emulations.

Parameters	Predictable FE (1st Semi-final)	Unpredictable FE (Wikipedia hits)
Duration	6 h	43 h
Number of requests	29,662,465	7,417,070
Number of unique source IPs	79,033	N/A
Number of unique resources accessed	11,885	N/A

Table 3 shows the high-level characteristics of the two FE datasets used in the emulations.

5.2.2. *Experimental results and analysis*

In this section, first, the parameter values of the FE model proposed by Bhatia et al. (2012) are extracted for different FE datasets. These values, summarised in Table 4, are then used to generate the configuration file loaded to the Botloader program (running on each host machine within the testbed architecture) to synthetically generate the traffic. This input file consists of tab separated values for the number of packets (HTTP GET requests) to be sent, the number of unique source IPs to use, the number of unique resources to access, and the resource entropy to emulate. All of these values are supplied on a per-sampling-interval basis to the Botloader program.

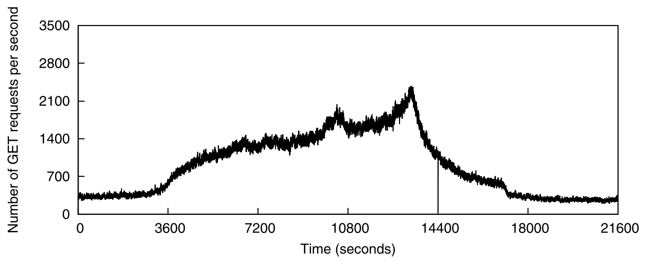
All the parameter values in Table 4 are computed using 'seconds' as the time unit for emulating the pFE and 'hours' for emulating the uFE. Therefore for the pFE dataset, R_n and R_p denote respectively the average number of incoming requests per second and the maximum number of incoming requests per second observed over a period of six hours. Values for t_o, t_f, and t_d denote the instances of time (in seconds) which mark the start and end of the flash-phase and the decay-phase. Similarly, the values for the uFE are computed using 'hour' as the time resolution. Different time resolutions were chosen for the pFE and the uFE because the original datasets were only available with different time resolutions.

Fig. 6a shows the incoming traffic profile for the original pFE dataset i.e. six hour traffic around the 1st semi-final match of the 1998 FIFA world cup. The predictable FE traffic, based on the FE model (Bhatia et al., 2012), is shown in Fig. 6b. The emulated traffic is able to mirror the shape of the actual

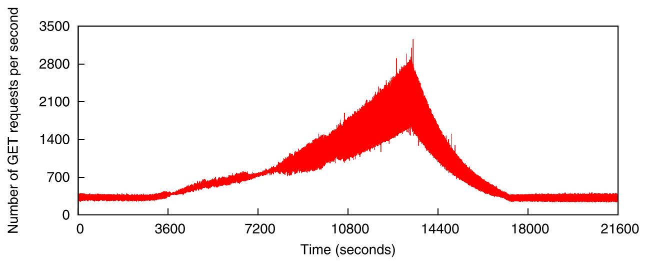
Table 4 – Parameter values of the FE model developed by Bhatia et al. (2012) for different FE datasets used in emulations.

FE datasets	R_n	R_p	t_o	t_f	t_d
pFE (1st Semi-final)	391	2376	2880	13,320	17,280
uFE (Wikipedia hits)	1826	1,063,665	8	10	43

where:
R_n: average normal incoming request rate.
R_p: peak incoming request rate.
t_o: start time of the flash-phase.
t_f: end time of the flash-phase or start time of the decay-phase.
t_d: end time of the decay-phase.



(a) Original predictable FE traffic.

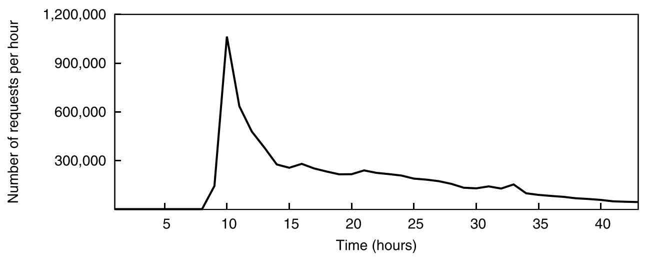


(b) emulated predictable FE traffic using the FE model.

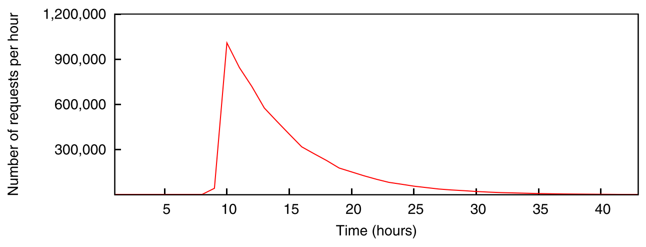
Fig. 6 – Incoming traffic profile for the predictable FE.

dataset, with exponentially increasing flash and decreasing decay phase. However, there are variations in the traffic that are noticeably different from the original data. The time-synchronisation of the 10 host machines, each sending 1/10th of the total traffic, is speculated to be the reason for these variations. Such variations can also be seen in the emulated traffic of the original FIFA dataset in Fig. 6a. A future work in this direction is to further investigate the possible reasons for such variations in the emulated pFE traffic.

Fig. 7a shows the incoming traffic profile for the original uFE dataset i.e. hourly hits on Wikipedia following Steve Jobs' death. The emulated uFE traffic using the FE model is shown in Fig. 7b. The emulated traffic is able to closely mimic the original traffic.



(a) Original unpredictable FE traffic.



(b) emulated unpredictable FE traffic using the FE model.

Fig. 7 – Incoming traffic profile for the unpredictable FE.

6. Conclusion and future work

Research associated with network anomaly detection, particularly DDoS attack detection and FE separation, often suffers from lack of accurate evaluation, mainly because of the scarcity of recent and realistic datasets available in the public domain. The majority of the publicly available datasets are either too old to reflect current network trends, or are heavily anonymised to eliminate any associated privacy and legal concerns, and so cannot provide useful information about the traffic contents or origins, or are available only in a format such as Common Log Format (CLF), which limits their usability, since they cannot be replayed over the network. Commonly used off-the-shelf hardware and software traffic generators have their own limitations in generating realistic network traffic, as do simulation programs such as ns−2.

The research presented in this paper addresses the aforementioned challenges in obtaining realistic network traffic datasets, and in the process makes two related contributions:

- the development and evaluation of a traffic generation and testbed framework, based on only modest hardware and using a software traffic generator, and
- the development of techniques to realistically emulate real-world DDoS attacks and FEs.

These contributions provide a cost-effective and dedicated test environment which can emulate large-scale DDoS attacks, both at the network and application layers, and FEs, and to monitor them in real time, at a scale not previously possible with existing emulation testbeds.

It is acknowledged that the evaluation of the proposed traffic generation and testbed framework is limited by the availability of datasets representing different DDoS attacks and FE scenarios in the public domain. This limitation is mainly due to the lack of parameter values such as number of bots, traffic rate, etc., required by the Botloader program in order to emulate a given trace. It is also to be noted that using similar values for these parameters and using other protocols such as TCP, UDP, HTTP and SSL, different attacks such as TCP SYN flood, UDP flood, HTTP GET flood and SSL flood were also simulated. However, due to non-availability of real-world traces for these attacks in the public domain, the simulated traces could not be evaluated and hence the results presented in the article are based only on publicly available datasets.

Future work will be directed towards further investigating the potential reasons for the variations in the emulated predictable FE traffic. This would involve improving the FE module to more closely mimic the traffic profile of the predictable FE, and also the 'new source IPs' and 'access resource pattern' characteristics of the original FE dataset. The current implementation of the Botloader program tends to use all the available bots (aliased IPs) at the beginning, and reuses them during the course of the emulation. Changing the traffic generation to get around this would result in a better and more controlled emulation overall.

Future work will also add a separate traffic capture machine to the existing testbed. In the current setup, experiments that measure memory usage must be run twice, once with Tcpdump to capture the traffic and once without, since the Tcpdump program increases memory usage, and so introduces an experimental confound. The use of a separate traffic capturing machine would allow the transparent mirroring of all network traffic between the attack switch and the target. However, this would require the purchase of an additional 10 Gb network interface module for the attack switch, and a 10 Gb interface card for the traffic capture machine, since the current testbed already uses a 10 Gb module to connect the attack switch to the target.

Acknowledgement

This research was supported by the Australia-India Strategic Research Fund (AISRF) 2008–2012.

REFERENCES

Ahmed E, Mohay G, Tickle A, Bhatia S. Use of IP addresses for high rate flooding attack detection. In: Proceedings of 25th international information security conference (SEC 2010): security & privacy – silver linings in the cloud, Brisbane, Australia 2010.
Arlitt M, Jin T. 1998 world cup web site access logs http://www.acm.org/sigcomm/ITA/; 1998 [Online; accessed 09.06.12].
Benzel T, Braden B, Faber T, Mirkovic J, Schwab S, Sollins K, et al. Current developments in DETER cybersecurity testbed technology. In: Conference for Homeland security, 2009. CATCH'09. Cybersecurity applications & technology. IEEE; 2009. pp. 57–70.
Bhatia S, Mohay G, Schmidt D, Tickle A. Modelling web-server flash events. In: Network computing and applications (NCA), 2012 11th IEEE international symposium on. IEEE; 2012. pp. 79–86.
Blackert W, Gregg D, Castner A, Kyle E, Hom R, Jokerst R. Analyzing interaction between distributed denial of service attacks and mitigation technologies. In: DARPA information survivability conference and exposition, 2003. Proceedings, vol. 1. IEEE; 2003. pp. 26–36.
Botta A, Dainotti A, Pescapé A. A tool for the generation of realistic network workload for emerging networking scenarios. Computer Networks 2012;56(15):3531–47.
Buchanan WJ, Flandrin F, Macfarlane R, Graves J. A methodology to evaluate rate-based intrusion prevention system against distributed denial-of-service (DDoS) 2011.
Calvet J, Fernandez J, Bureau P-M, Marion J-Y. Large-scale malware experiments why, how, and so what?. In: Proceedings of virus bulletin conference 2010. pp. 241–7.
Hettich S, Bay SD. The UCI KDD archive. University of California, Department of Information and Computer Science; 1999. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html [Online; accessed 26.09.12].
Hick P, Aben E, Claffy K, Polterock J. The CAIDA "DDoS attack 2007" dataset http://www.caida.org/data/passive/ddos-20070804_dataset.xml; 2007 [Online; accessed 09.06.12].
Iakobashvili R, Moser M. Welcome to Curl-loader http://curl-loader.sourceforge.net/; 2009 [Online; accessed 22.11.12].
Krishnamurthy B, Wang J. On network-aware clustering of web clients. In: Proceedings of the conference on applications, technologies, architectures, and protocols for computer communication. ACM; 2000. pp. 97–110.
McCanne S, Floyd S, Fall K, Varadhan K. Network simulator ns–2; 1997.
McHugh J. Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln laboratory. ACM Tran Inform Syst Secur (TISSEC) 2000;3(4):262–94.
Mirkovic J, Fahmy S, Reiher P, Thomas R. How to test DoS defenses. In: Conference for homeland security, 2009. CATCH'09. Cybersecurity applications & technology. IEEE; 2009. pp. 103–17.
Mituzas D. Wikipedia page counters http://dumps.wikimedia.org/other/pagecounts-raw/; 2007 [Online; accessed 09.06.12].
Mituzas D. In numbers http://dom.as/2011/10/07/steve-jobs/; 2011 [Online; accessed 26.09.12].
Modeler O. OPNET Technologies Inc, Bethesda, MD. URL: http://www.opnet.com.
Nazario J. Political DDoS: Estonia and beyond (Invited Talk). In: USENIX security, vol. 8; 2008.
Niksic H. GNU wget. Available from the master GNU archive site prep.ai.mit.edu, and its mirrors, 1998.
Peterson L, Bavier A, Fiuczynski ME, Muir S. Experiences building Planetlab. In: Proceedings of the 7th symposium on operating systems design and implementation. USENIX Association; 2006. pp. 351–66.
Rajab M, Zarfoss J, Monrose F, Terzis A. My botnet is bigger than yours (maybe, better than yours): why size estimates remain challenging. In: Proceedings of the first conference on first workshop on hot topics in understanding botnets. USENIX Association; 2007. p. 5.
Sachdeva M, Singh G, Kumar K, Singh K. Measuring impact of DDoS attacks on web services 2010.
Sanfilippo S. Hping–Active network security tool http://www.hping.org/; 2008 [Online; accessed 22.11.12].
SmartBits. Spirent 600 series traffic generators. http://www.spirent.com; 2001 [Online; accessed 26.09.12].
Sommers J, Kim H, Barford P. Harpoon: a flow-level traffic generator for router and network tests. In: ACM SIGMETRICS Performance Evaluation Review, vol. 32(1). ACM; June 2004. 392–392.
Tavallaee M, Bagheri E, Lu W, Ghorbani A. A detailed analysis of the KDD Cup 99 data set. In: Proceedings of the 2009 IEEE symposium computational intelligence for security and defense applications (CISDA 09). IEEE Computer Society; 2009.
Waikato Applied Network Dynamic Research Group, http://wand.cs.waikato.ac.nz/wits/auck/8/ [Online; accessed 26.09.12].
White B, Lepreau J, Stoller L, Ricci R, Guruprasad S, Newbold M, et al. An integrated experimental environment for distributed systems and networks. ACM SIGOPS Operat Syst Rev 2002;36(SI):255–70.
Yu J, Kang H, Park D, Bang H-C, Kang DW. An in-depth analysis on traffic flooding attacks detection and system using data mining techniques. Journal of Systems Architecture 2013;59(10):1005–12.

Sajal Bhatia holds Bachelor of Technology degree in Communication and Computer Engineering from the LNM Institute of Information Technology, awarded in 2008. He is currently a PhD student within the Science and Engineering Faculty at the Queensland University of Technology. His research interests include network security, intrusion detection and critical infrastructure protection.

Dr. Desmond Schmidt has a PhD in Classics (Cambridge, 1987) and Information Technology (University of Queensland, 2010). He has worked as an IT Manager and software engineer in the UK and Australia, including work on Leximancer, a concept mining tool, and on a secure licence-management system for software installation. He worked on the joint India-Australia project investigating various aspects of Denial of Service from 2009 to 2012.

Dr. George Mohay has been an Adjunct Professor in the previous Information Security Institute at the Queensland University of Technology, Brisbane, Australia. Prior to this he was Head of the School of Computing Science and Software Engineering from 1992 to 2002. His research interests have been in the areas of computer security, intrusion detection, and computer forensics. He has worked as a visiting researcher while on sabbatical leave at Stanford University in 1981, Loughborough University in 1986, Bristol University in 1990 and the Australian National University in 2000. He graduated BSc (Hons) (UWA) in 1966 and PhD (Monash) in 1970.

Dr. Alan Tickle is an Adjunct Professor within the Science and Engineering Faculty at the Queensland University of Technology. His current research interests include the application of machine-learning techniques within information security and investigating strategies for improving learning outcomes in mathematics and physics for students from a non-English-speaking background.