Detecting TCP SYN Flood Attack based on Anomaly Detection

S.H.C. Haris, R.B. Ahmad, M.A.H.A. Ghani
School of Computer and Communication Engineering,
Universiti Malaysia Perlis, P.O Box 77,
d/a Pejabat Pos Besar, 01000 Kangar, Perlis, Malaysia.
shajar_charis@yahoo.com, badli@unimap.edu.my, alifhasmani@unimap.edu.my

Abstract- **Transmission Control Protocol (TCP) Synchronized (SYN) Flood has become a problem to the network management to defend the network server from being attacked by the malicious attackers. The malicious attackers can easily exploit the TCP three-way handshake by making the server exhausted and unavailable. The main problem in this paper is how to detect TCP SYN flood through network. This paper used anomaly detection to detect TCP SYN flood attack based on payload and unusable area. The results show that the proposed detection method can detect TCP SYN Flood in the network through the payload.**

I. INTRODUCTION

The network security management confronts a big problem to detect threats since the past few years since computer viruses are widely recognized as a significant computer threat. The increase rate of new threats is high due to technology improvements especially in the internet can accelerate the threats spreading through the network by intruders. Since the intruders are not giving up to penetrate the network, network security managers is in the hopeless situation of trying to uphold a maximum of security, as requested from management, while at the same time an obstacle way of developing and introducing new applications into business and government network environments have to be considered.

There are many types of threats that can cause computer to be infected such as phishing, hackers, worms, virus that being spread out especially in the internet and network without being realize by the users. This paper focus on TCP SYN Flood attack detection by using the Internet Protocol (IP) header and TCP header as a platform to detect threats especially in the IP protocol and TCP protocol. This detection method checks the data (packets) using rate based anomaly detection system which has many advantages, and applied it under the Operating System (OS) GNU Not UNIX (GNU)/Linux.

To detect TCP SYN flood, anomaly detection is one of the most frequently suggested approaches to detect attack variants, which looks for deviations from normal behavior, signaling a possibly attack. This paper focusing on detecting threats in the local network in File Transfer Protocol (FTP) by monitoring all the packets that goes through the networks.

The rest of the paper is organized as follows. Section 2 discusses related work in detecting TCP SYN Flood. In Section 3, explain what is TCP SYN Flood and Section 4 explains types of the anomaly detection technique. Section 5 presents the algorithm for detection techniques. Next in section 6 the experiment setup followed by results and discussion in Section 7. Lastly, Section 8 concludes the paper.

II. RELATED WORKS

Several methods for detecting TCP SYN Flood attacks have been proposed**.** Some of the methods are explained as the following:

Y. Ohsita [1] had proposed a mechanism for detecting SYN flood by taking into consideration the time variation of arrival traffic. Packet had been divided into five groups according to its flow using the TCP flags; first group that complete the three-way handshake, second group is flows terminated by a Reset (RST) packet and etc. This method can detect attacks quickly and accurately regardless of the time variance into consideration but cannot detect attack with lower rate because the traffic having the lower rate attacks still follows the normal distribution. The analytical results show that the arrival rate of normal TCP SYN packets can be modeled by a normal distribution and that proposed mechanism can detect SYN Flood traffic quickly and accurately regardless of time variance of the traffic.

H. Wang [2] proposed a simple mechanism that detect the SYN flooding attacks at leaf routers that connect end hosts to the Internet, instead of monitoring the ongoing traffic at the front end (like firewall or proxy) or a victim server itself.This detection mechanism lies in its statelessness and low computation overhead, which make the detection mechanism itself immune to flooding attacks. It is based on the protocol behavior of TCP Synchronized-Finish (SYN–FIN) RST pairs, and is an instance of the Sequential Change Point Detection [3] and a non-parametric Cumulative Sum (CUSUM) method. This mechanism not only sets alarms upon detection of ongoing SYN flooding attacks, but also reveals the location of the flooding sources.

W. Chen [4] proposed a simple and efficient method to detect and defend against TCP SYN flooding attacks under different IP spoofing types, including subnet spoofing. The method makes use of storage-efficient data structure using Bloom filter which required a fixed-length table for recording relevant traffic information, and change-point detection method (CUSUM) to distinguish complete three-way TCP handshakes from incomplete ones. Simulation experiments consistently show that both efficient and effective in defending against TCP based flooding attacks under different IP spoofing types.

M. Mahoney [5] proposed anomaly detection based on packet bytes, Network Traffic Anomaly Detector (NETAD). NETAD used two stages of anomaly detection system to identify suspicious traffic. First stage, NETAD filtered the incoming server request by removing the uninteresting traffic such as TCP connection start with SYN-ACK packets and in next stage, the packet bytes value is being observed according to each protocol. From the experimental results showed this technique can reduce false alarm rate of any Intrusion Detection System (IDS), because multiple detections of the same attack are counted only once, but false alarm is counted separately.

In this paper, the packets are divided into *two main ways of detection methods based on IP Header and TCP Header payload.* These detection methods also considered the port, flags, and the mechanism of TCP three-way handshake for TCP protocol. Every port is scanned to identify sending and receiving application end-points on a host as well to check for threats. Each flag is checked to detect three-way handshake that are not completed and packet that have threats. At the same time the IP address is also checked for IP spoofing and the flow of the packet is being monitored. The protocol behavior also checked in order to know that the packet traffic is normal. If the flow is not as usual, the detection software will send an alarm to the administration.

This research is different from H. Wang and W. Chen that used CUSUM method to detect TCP SYN flood but it used TCP flags in order to detect TCP SYN flood. This research also used same technique as NETAD which used packet filtering according to each protocols but different in packet filtering rules. This detection method is simple but effectively detect TCP SYN flood in the network once the abnormal behavior is detected in the payload. The analysis of the payload behavior in this research is explained in Section 5.

III. TCP SYN Flood

A normal TCP connection usually starts a transmission from client by sending a SYN to the server, and the server will allocate a buffer for the client and replies with a SYN and Acknowledge (ACK) packet. At this stage, the connection is in the half-open state, waiting for the ACK reply from the client to complete the connection setup. When the connection is complete, it called three-way handshake and TCP SYN Flood attack manipulate this three-way handshake by making the server exhausted with SYN request.

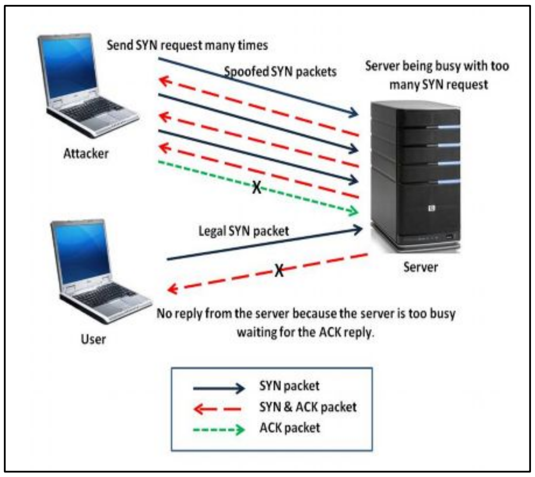

Send SYN request many times
Spoofed SYN packets
Server being busy with too many SYN request
Attacker
Legal SYN packet
Server
No reply from the server because the server is too busy waiting for the ACK reply.
User
SYN packet
SYN & ACK packet
ACK packet

Fig. 1: TCP SYN Flood

The TCP SYN Flood happened when the three-way handshake had been exploited by the intruders without being realized by the server. The steps are as below.

1. An intruder uses spoofed address to send a SYN packet to victim server.
2. A victim server sends back a request SYN and ACK packet to the client or spoofed address and wait for confirmation or timeout expiration of SYN packets.
3. If the client does not send back the final ACK packet, the server's resources can be easily exhausted.
4. At this time the SYN flood attack occurred because too many SYN packet request from other client.

The illustration of the TCP SYN flood attack as shown in Figure 1. The attack succeeds because the number of half-open connections that can be supported per TCP port is limited. When the number of half-open connections is exceeded the server will reject all subsequent incoming connection requests until the existing requests time out, creating a Denial of Service (DoS) condition.

IV. Detection Methods

There are two types of network intrusion detection system which are signature based detection or anomaly based detection. Signature detection is a technique often used in the Intrusion Detection System (IDS) and many anti-malware systems such as anti-virus and anti-spyware. In the signature detection process, network or system information is scanned against a known attack or malware signature database. If match found, an alert takes place for further actions [6].

This paper is focusing on anomaly based detection which infers malicious activity in a network by detecting anomalous network traffic patterns [7, 8]. There are three types of network analysis behavior:

- It used protocol to detect packets that are too short which violate specific application layers protocol.
- Rate-based detection which detects floods in traffic using a time-based model of normal traffic volumes especially DoS attacks.
- It detects through the behavioral or relational changes in how individual or groups of hosts interact with one another on a network.

In this paper, the anomaly detection is used in checking the IP Header and TCP Header using the payload. IP Header includes the fields such as: IP Header Length (IHL), Type of Service (ToS), Identification (ID), Flags and etc. Each field has rules such as ToS must be zero and IP header length must be equal to 20.

In TCP/IP protocol, the entire packet must go through the data link layer first before it goes to the next layer where the other protocols work. Figure 2 shows TCP Header that built on top of IP header, which is unreliable and connectionless. TCP header occupies 20 bytes but TCP header format has some limitations in header length. A normal TCP header is 20 bytes but TCP can have another 40 bytes for option. So the header size is limited to 60 bytes [9]. The sequence number is essential in keeping the sending and receiving datagram in proper order. Window size is advertised between communication peers to maintain the flow control.

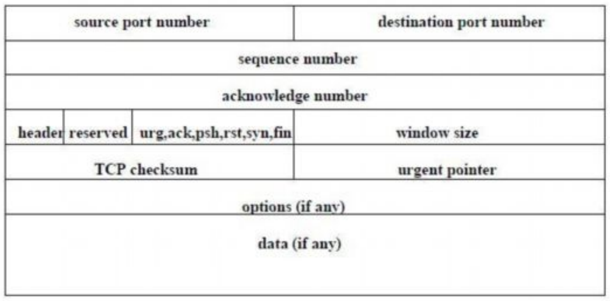


Fig. 2: TCP Header Format

Most of nowadays internet services relay on in TCP such as mail that using Simple Mail Transfer Protocol (SMTP) on port 25, FTP on port 21 and also the Hypertext Transfer Protocol (HTTP) on port 80. In this paper, FTP is the main port to be attacked by TCP SYN Flood.

V. Packet Filtering and Monitoring System

The anomaly detection technique proposed in this paper is focusing on payload and unusable area in TCP protocol. This technique focused on monitoring the traffic and filtering the packet using Tcpdump [10]. Thus, monitoring the performance of the network is a way to detect an abnormal flow that can be caused by TCP SYN Flood. The filtering algorithm is shown in Figure 3.

Algorithm for Packet Filtering System

```
for each packet arrival do
    check the IP Header Length
        if IP Header Length = 20 then
            choose the protocol = TCP
            check the payload packet
               if payload normal
                  goto destination
               else distinguish the packet for analysis
                  if TCP SYN Flood then
                     report to administrator
                  else analysis for other threats
                  end if
               end if
            other protocol
            goto destination
    end if
end for
```

Fig. 3: Packet Filtering Algorithm

The packet filtering algorithm filtered the packet that had been captured through Tcpdump. In order to filter the packets, rules must be setup for filtering it. Each field has rules such as ToS must be zero and IP header must be equal to 20. If the IP header is not equal or more than 20 bytes, the packet is send for analysis and check for any threat. On the other hand, the payload and unusable area also checked and unusable area must be zero.

In order to check TCP SYN Flood, flags in TCP are considered. TCP have SYN, ACK, Finish (FIN), Reset (RST), Urgent (URG) and Push (PSH) flags in its protocol and each of these flags have function in order to make a connection establishment, termination and control purposed. Since TCP SYN Flood attack will flood the network with request packet when in the half-open condition, so the SYN packets are investigated. If the packet contains a TCP SYN flood packet, distinguish packet for analysis to confirm whether the packet is truly comes from attackers. Otherwise, if the packet is a normal packet, it will go through sending the data to the destination. Then send the report to administrator for further action. This algorithm is applied in the experiment in the next section.

This filtering algorithm is used in the next section for the packet filtering and combination with monitoring the network system. In monitoring system, all the system usage is monitored by the administrator such as receiving and sending data, CPU utilization and also memory cache.

VI. Experimental Setup

This experiment used three host machines and switch as a testbed that based on operating system GNU/Linux Ubuntu version 9.04 and Ultimate Edition 2.5. Linux is used because it is capable to act either client or server and can be change whenever management want it due to its application [11]. These three host machines act as client (Host A), server (Host B) and attacker (Host C) using the same switch. Each host have different processor configuration as shown in Table 1.

Table 1: Host Configuration

Host	Hardware Configuration
Attacker Machine (Host A)	OS Ubuntu 9.04 Intel Core 2 Duo P7350 Processor speed 2.0 GHz
FTP Server Machine (Host B)	OS Ubuntu 9.04 Intel Pentium D CPU Processor speed 3.4 GHz
Client Machine (Host C)	OS Ultimate Edition 2.5 Intel Core 2 Quad CPU Q6600 Processor speed 2.4 GHz

In this testbed, Host C download file from Host B and at the same time Host A attack Host C. Host B is an open FTP server for any client to download file from it. FTP is the main port to analyze in this paper which is in port 21. The TCP SYN Flood program is run for ten minutes for attacking Host C and the packets being captured every 10,000 packets using Tcpdump, and being filtered by packet filtering algorithm. On the other hand, the network system being monitored to see any changes in the network flows. Both of monitoring system and packet filtering is done in a real time.

A. Traffic Monitoring

Monitoring the flow of packets using the system monitor can detect the abnormal behavior or flow in the network. There two main analyses for this traffic monitoring:

1. CPU utilization for normal network and attacked network.
2. Network history in receiving data during the normal situation, downloading file and attacked situation.

This entire situation is important to show the network traffic and to differentiate the infected network and normal network. This monitoring is analyzed every 60 seconds.

B. Packet Filtering

So, as to do the packet filtering, many factors will be considered. There are three main factors in this paper:

1. The traffic filtered each packet to each protocol such as TCP, UDP and ICMP. Since TCP packet is the main protocol in this research, other protocol is distinguished.
2. TCP flags SYN, ACK, RST, FIN, PSH, and URG are divided to each group to check the three-way handshake is complete or not. Usually, when there are a lot of SYN packets or RST packets, the analysis is focusing to these packets and assumed there is an attack.
3. IP address is valid and not a spoofed address. Since this experiment is done in local network, unrecognized IP address is considered an attack.

These factors are important to detect method to recognize which packets are the infected packets. These infected packets can show the character of TCP SYN Flood and this character is discussed in next section.

VII. Results and Discussion

From the experiments, both techniques monitoring and filtering system shows the differences for the normal flow and infected flow. Each techniques have detected the abnormal behavior and it show in the figures.

A. Traffic Monitoring Analysis

From the traffic monitoring, the CPU utilization and network history of downloading data had been analyzed. Figure 4 shows the network history when in the normal situation where there is no activity done in Host C between 40 seconds to 60 seconds. All the receiving data are below than 1.0 KiB/s. This shows that the network is not full of activity at all.

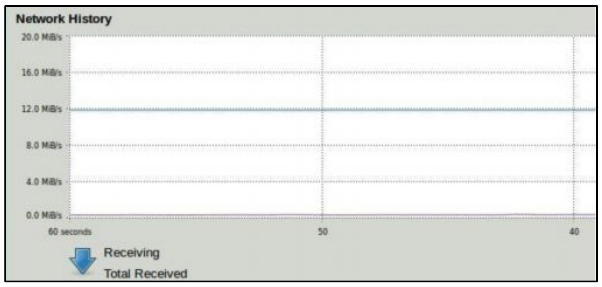


Fig 4: Network History of Receiving Data in Normal Condition

Next situation shows the network history when downloading a file from Host B. Figure 5 shows a constant rate of 12 MiB/s when receiving the data and took only one minute to finish the download of 530 MB file.

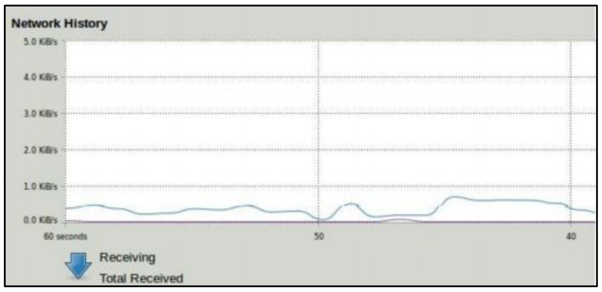


Fig 5: Network History of Receiving Data when Downloading Condition

This is totally different in when compared to Figure 6. The graph shows there is a drastically increase in receiving data up to 22 MiB/s. This type of graph shows an abnormal behavior that can be concluded as infected network. The downloading process also took nearly ten hours even though it is exactly the same file that had been downloaded.

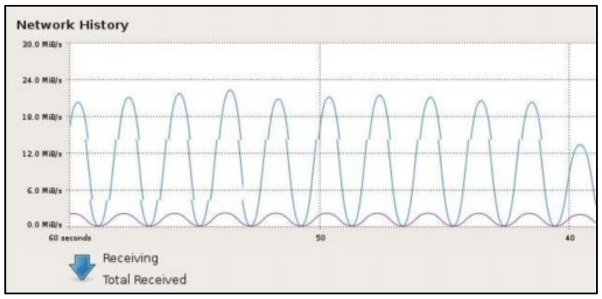


Fig 6: Network History of Receiving Data when Downloading and Attacked by TCP SYN Flood

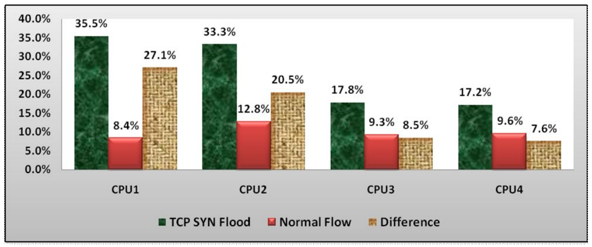


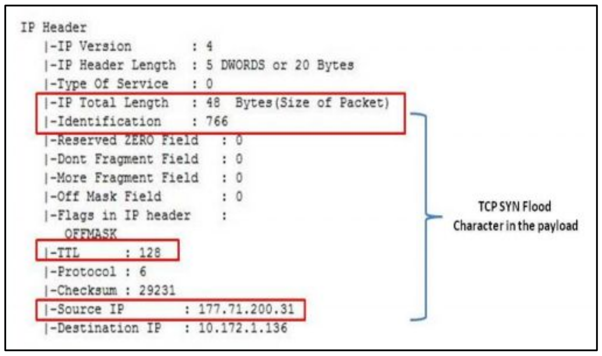
Fig. 7: CPU Utilization for every network flow.

The CPU utilization also shows big differences where every CPU shows an increasing in usage especially in CPU1. CPU1 have 27.1% of different when comparing between TCP SYN Flood infected network and normal network. This is because CPU1 is the main CPU that being used in this experiment.

B. Packet Filtering Analysis

This packet filtering analysis is focusing on payload where the entire payload especially in IP Header and TCP Header had been checked. Each packet being analyzed by comparing normal characters to infected packets. Each packet checked beginning from the IP Header, then go to TCP Header.

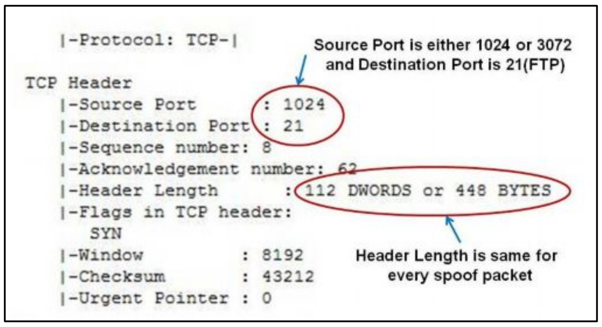
Figure 8 shows the abnormal IP Header where the field that had been highlighted is repeating many times but the source IP address is different for each packet since it is a spoof IP address.



```
IP Header
   |-IP Version        : 4
   |-IP Header Length  : 5 DWORDS or 20 Bytes
   |-Type Of Service   : 0
   |-IP Total Length   : 48  Bytes(Size of Packet)
   |-Identification    : 766
   |-Reserved ZERO Field   : 0
   |-Dont Fragment Field   : 0
   |-More Fragment Field   : 0
   |-Off Mask Field        : 0
   |-Flags in IP header    :
      OFFMASK
   |-TTL      : 128
   |-Protocol : 6
   |-Checksum : 29231
   |-Source IP        : 177.71.200.31
   |-Destination IP   : 10.172.1.136
```

Fig. 8: Infected IP Header Character

Next, the TCP Header is checked for abnormal behavior. This proposed techniques show that TCP SYN flood had character that can be recognized in this analysis by its source port and header length. From the analysis, the source port either from 1024 or 3072 and the header length for infected packet is 448 bytes.



```
   |-Protocol: TCP-|

TCP Header
   |-Source Port      : 1024
   |-Destination Port : 21
   |-Sequence number: 8
   |-Acknowledgement number: 62
   |-Header Length      : 112 DWORDS or 448 BYTES
   |-Flags in TCP header:
     SYN
   |-Window           : 8192
   |-Checksum         : 43212
   |-Urgent Pointer : 0
```

Source Port is either 1024 or 3072
and Destination Port is 21(FTP)
Header Length is same for
every spoof packet

Fig. 9: Infected TCP Header Character

VIII. CONCLUSION

The analysis for the packets is to detect threats that attack through the network. Threats are detected due to the IP Header (payload and unusable area) and TCP Header using the FTP. From the results, the suggested detection method can detect TCP SYN flood in the network. The parameters that had been observed in the CPU utilization and receiving file data rate had shown a drastically changed during downloading file as comparing before and after the attack.

By analyzed every packet to each category in TCP protocol (port, flags, and TCP three-way handshake) and IP header, the TCP SYN Flood is easier to detect once we know the behavior of an attack. In order to detect TCP SYN Flood, the normal payload characters must be understand first unless the analysis will take times for those that are not expert in payload characters.

In the experiment, the main threats in this paper, TCP SYN Flood attack had been traced even in the network. The detection method of attacks can be improved in order to make the detection faster and effective, and alarming the security administration department whenever there is an attack or abnormal behavior in the flow of the traffic.

REFERENCES

[1] Y. Ohsita, S. Ata, and M. Murata, "Detecting Distributed Denial-of-Service Attacks by Analyzing TCP SYN Packets Statistically," *Proceeding of the IEEE Communications Society Globecom*, pp. 2043-2049, 2004.

[2] H. Wang, D. Zhang, and K. G. Shin, "Detecting SYN Flooding Attacks," *Proceeding in the INFOCOM IEEE Communications Society,* 2002.

[3] M. Basseville, and I. V. Nikiforov, *Detection of Abrupt Changes: Theory and Application*, Prentice Hall, 1993.

[4] W. Chen, and D.Y. Yeung, "Different Types of IP Spoofing," *Proceeding of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL)*, 2006.

[5] M. Mahoney, "Network Traffic Anomaly Detection Based on Packet," *Proceeding of the Symposium on Applied Computing,* ACM-SAC, 2003.

[6] "Information, Computer and Network Security Terms Glossary and Dictionary,"http://www.javvin.com/networksecurity/SignatureDetection.html

[7] D. Whyte, E. Kranakis, and P. Van Oorschot, "DNS-based Detection of Scanning Worms in an Enterprise Network," *Proceeding of the Network and Distributed Systems Symposium (NDSS),* 2005.

[8] P. Barford, J. Kline, D. Plonka, and R. Amos, "A Signal Analysis of Network Traffic Anomalies," *Proceeding of the ACM SIGCOMM Internet Measurement Workshop*, Marseilles, France, November 2002.

[9] "TCP Header Format (3),"http://www.soi.wide.ad.jp/class/20020032/slides/11/15.html

[10] "TCPDUMP/LIBPCAP Public Repository,"http://www.tcpdump.org/, April, 2010.

[11] "Why Linux?," http://www.seul.org/docs/whylinux.html

Port Scan Detection

Jayant Gadge and Anish Anand Patil

***Abstract*- Port scanning is a phase in footprinting and scanning; this comes in reconnaissance which is considered as the first stage of a computer attack. Port scanning aims at finding open ports in a system. These open ports are exploited by attackers to carry out attacks and exploits. There are a number of tools to scan for open ports. However, very few tools are present to detect port scanning attempts.**

The goal of this project is to identify port scan attempts and find out information about the machine from where port scan attempts were made. If an attack takes place after the port scan, the collected information would help in bringing the criminal to justice. We hope that this work will add an additional layer of defense by identifying port scan attempts thereby indicating that an attack may follow.

I. INTRODUCTION

With an increase in computer literacy people are becoming aware about the loopholes present in the Operating Systems, networking protocols, software applications which are used on a daily basis. Many easy-to-use tools are freely available on the Internet which can take advantage of these loopholes to gain unauthorized access to a system. To further complicate the things most of us do not follow good security practices, making the job of computer criminals even easier.

Computer crimes have increased over the years. They are not limited to trivial acts such as guessing the login password of a system, they are much more dangerous. Studies indicate that the first stage of an attack is reconnaissance [5]. In this stage the prime objective is to get information about the target system. One critical piece of information is the list of open ports of the system. Open ports of a system can be exploited in a number of ways. To identify open ports a number of tools are available [4].

Currently a number of solutions are in place to deal with attacks. However, most of the solutions such as Antivirus and Intrusion Detection Systems indicate occurrence of an attack or an un-authorized activity when it happens. Having a system which predicts occurrence of attacks in the near future is advantageous. As port scans are usually performed before an actual attack, identification of port scan attempts gives precautionary indication that attacks might follow in the near future.

The goal of this project is to identify port scan attempts. This would make it possible to take precautionary steps to strengthen the defenses of the system. It would be very useful to have information about the machine from where the scans are coming. Information such as the Operating System being used, the possible location from where the scan came, information from WHOIS database and Traceroute would help in providing clues about the scanner. This information can be used against him if an attack takes place in future.

II. SCAN DETECTION METHODOLOGY

An attack typically goes through the following phases:

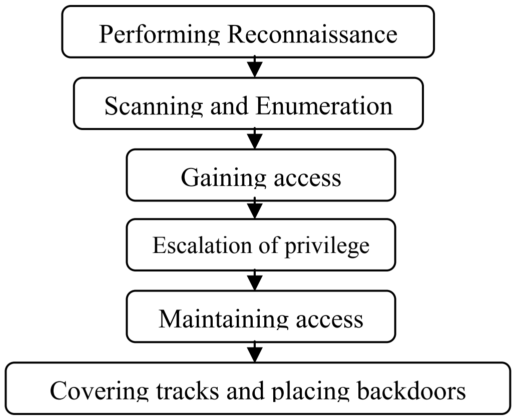


Figure 1. Steps in an attack.

Reconnaissance is considered the first pre-attack phase and is a systematic attempt to locate, gather, identify, and record information about the target. The hacker seeks to find out as much information as possible about the victim. This first step is considered a passive information gathering. This involves activities such as information gathering, determining the network range, identifying active machines, finding open ports and access points, OS fingerprinting, fingerprinting services and mapping the network.

The port scanners are mainly of two types.

- Brute force scanners.
- Stealth scanners.

Brute force scanners essentially perform scans in an aggressive manner by scanning one port after another for the specified range. They establish a full connection to the target machine and inspect whether the port is open. Owing to the full connection establishment, it is possible to detect their presence. Thus when a large number of SYN packets arrive to request for a connection from a single IP address at multiple ports of the target machine, it indicates that a brute force scanner is being used to look for open ports.

Stealth scanners get their name from their pattern of not establishing a full connection with the target. They send a single packet with a particular flag set at the target, based on the response it can be understood whether the ports are open or not. There are various types of scans; patterns for each scan can be identified as follows:

A. SYN Scans

In this scan a large number of packets with only the SYN flag set arrive at the destination. This scan does not complete the 3-way TCP connection establishment handshake and tears down the connection after the victim replies with a SYN/ACK indicating an open port. This scan can be easily identified if there are a large number of packets with the SYN flag set in them coming from a single host.

B. TCP Connect Scan

In this scan a large number of connections are established with the victim at different ports. Establishment of a connection at a port indicates that the corresponding port is open. Once the connection is established and the open ports identified, the connection is closed. As in this scan complete connection is established, TCP options such as timestamp and sequence acknowledgement are present. Thus if from a particular host a large number of connection are established at multiple ports in a very short span of time it can be inferred that a TCP CONNECT scan is coming from that machine.

C. ACK Scan

In this scan a large number of packets with only the ACK flag set arrive at the destination. This scan does not complete the 3-way TCP connection establishment handshake and tears down the connection after the victim replies with a SYN/ACK indicating an open port. This scan can be easily identified if there are a large number of packets with the ACK flag set in them coming from a single host.

D. FIN Scan

In this scan a large number of packets with only the FIN flag set arrive at the destination. If the victim replies with a RST it indicates that the port is closed, open ports simply ignore these packets. This scan can be easily identified if there are a large number of packets with the FIN flag set in them come from a single host.

E. NULL Scan

In this scan a large number of packets with no flags set arrive at the destination. The open ports ignore these packets whereas closed ports reply back with a RST. This scan can be easily identified if there are a large number of packets with the no flag set in them coming from a single host.

F. XMAS Scan

In this scan the flags FIN, PSH and URG are set. Open ports ignore these packets whereas closed ports reply with a RST. This scan can be easily identified if there are a large number of packets with the FIN, PSH and URG flag set in them coming from a single host.

G. UDP Scan

In this scan a large number of UDP packets arrive at the destination. This scan does not complete the 3-way TCP connection.

H. ICMP Scan

This includes sending ICMP echo request to the specified IP addresses to see if they are alive.

I. Fragmentation Attack

In order to overcome rules set by firewalls, attackers split the packets into small fragments and send these individual pieces over the network. These packets pass the firewall as rules meant for these individual pieces are not present This can be detected if there are a large number of packets with a 'header too short' string in them.

III. FRAMEWORK OF THE SYSTEM

The system is designed to help detect a possible port scan, getting additional information about the scanner such as his probable location, Operating System being used by attacker would help in uncovering the identity of the scanner.

The following figure shows the framework of the proposed system.

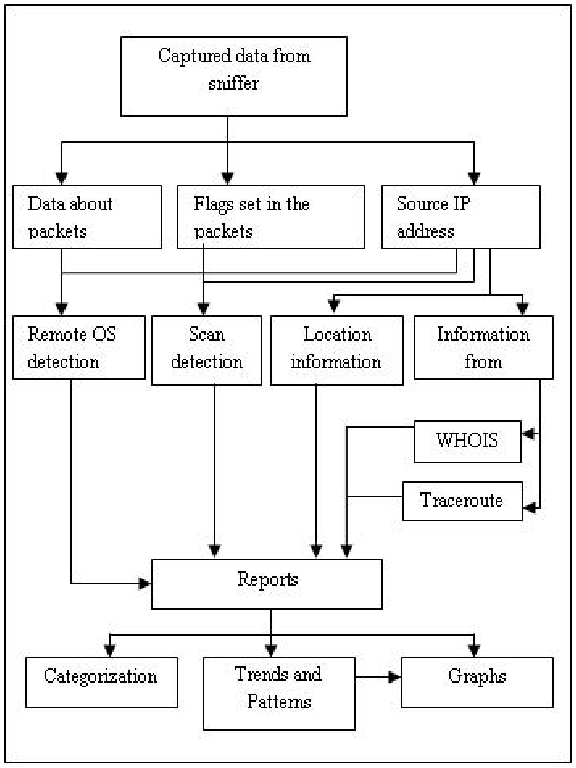


Figure 2. Framework of the System

For an administrator it would be very helpful to have knowledge that someone is performing a port scan over a system. This gives a hint that some sort of attack might follow. Having an understanding of which ports are being scanned, it is possible to predict what kind of attack may follow. This helps the Administrator in taking precautionary steps before an attack takes place.

A. Scan Detection

Using the information provided by the sniffer about the incoming packets, in particular the TCP flags present, patterns similar to general and stealth scans are found within the incoming packets. If a patterns is identified it will be marked. If the number of marked entries reaches a pre-specified threshold, it indicates that a scan is being performed.

Filters are used to filter out the unnecessary traffic and concentrate only on packets which might indicate a port scan attempt. For instance, a filter ' tcp[tcpflags] (tcp-syn|tcp-fin|tcp-ack) !=0 ' captures data about packets which have Ack, Syn and Fin flags set. Thus, using appropriate filters enables capturing of relevant packets with regards to different types of scans.

Packets may be coming from many different sources; every packet is associated with the machine from which it is coming with the help of IP address.

Online activities such as checking the E-mail, internet messengers and surfing web-pages generate packets which might be captured by the filters used. But the number of packets captured within a short span of time for these activities are very few compared to those captured when a scan is coming from a port scanner. Moreover, port scanners generally send a large number of packets with a particular flag set to a large number of ports on the target machine; this further distinguishes general online activity from port scans. This approach allows detection of scans coming from brute force scanners as well as stealth scans.

IV. INFORMATION GATHERING

A packet sniffer provides information about incoming packets. The incoming packets coming from a machine give a very crucial piece of information, the IP address of the machine from where the scans came from. This IP address now acts as the identity of the scanning machine. Apart from detecting scans, it is advantageous to have a system which provides information about the machine from where scans come from. This information gives clues which if put together may be sufficient enough to uncover the real identity of the person performing the scans.

Some of the information which can be found about the remote machine using IP address is as follows.

A. Operating System Being Used

Operating System being used on a remote machine can be guessed once the IP address of the corresponding machine is known. Every Operating System has default values for certain parameters such as time to live (TTL), type of service (Tos) and window size (Wsize). For packets coming from a machine if we know the values for these parameters, the Operating System being used can be understood. The values considered for the system developed are as follows:

Type Of Service (TOS)

The TOS bits specifies how the network should make trade-offs between throughput, delay, reliability, and cost.

Don't Fragment Flag

This flag can be set to 1 by a transmitting device to specify that a datagram not be fragmented in transit. This may be used in certain circumstances where the entire message must be delivered intact as pieces may not make sense.

Window Size (Wsize)

Window size specifies the maximum amount of received data, in bytes, that can be buffered at one time on the receiving side of a connection. The sending host can send only that amount of data before waiting for an acknowledgment and window update from the receiving host.

Time To Live (TTL)

Time to Live originally involved a sense of time. It is now used as a simple, but very effective; count to prevent routing errors and loops. Every router that handles the packet decrements the TTL value and if it reaches zero the packet is returned with an ICMP Time Exceeded message.

Maximum Segment Size (MSS)

Maximum segment size has to do with defining the largest amount of data that a computer system or other type of communications device can efficiently deal with without breaking the data into smaller components. Generally, the maximum segment size is calculated as the number of bytes that the device can handle at one time.

Timestamp

Timestamps were conceived to assist TCP in accurately measuring round trip time (RTT) in order to adjust retransmission time-outs.

Sequence Acknowledge (SackOK)

Prior to SACK, a receiver could only acknowledge the latest sequence number of contiguous data that had been received, or the left edge of the receive window. With SACK enabled, the receiver continues to use the ACK number to acknowledge the left edge of the receive window, but it can also acknowledge other blocks of received data individually.

No Operation (NOP)

The nop TCP Option means "No Option" and is used to separate the different options used within the TCP Option field. The implementation of the nop field depends on the operating system used. Nop option occupies 1 byte.

Signatures can be prepared for different Operating Systems based on values for these parameters. For instance, the signatures for Operating Systems such as Windows and Linux families are as follows:

Operatign System : TOS, DF, Wsize, MSS, Timestamp, SackOK, nops.
Windows : 0x0, [none], 30000-90000, 1, 0, 1, 2.
Linux : 0x0, [DF], 5000-9000, 1, 1, 1, 1.

These values are present in the packets itself, as a result this information is provided by the sniffer. However, value for parameters such as Timestamp, Nop, MSS are present in a packet only if a TCP connection is established between the sender and receiver. These parameters are collectively called TCP Options. These options are not present in packets which do not establish a complete connection. Thus, for packets belonging to stealth scans, these parameters are not present. As a result, Operating System detection would be done on the basis of remaining parameters.

For the system developed, two approaches have been used to detect the remote Operating System.

- Active Operating System detection
- Passive Operating System detection

Active Operating System Detection

In this method, services such as ping, telnet and file transfer protocol (FTP) are used to generate a response from the remote machine. From the incoming response packets, values for the different parameters are obtained. Once the values are obtained a simple correlation with signatures for different Operating Systems provides the possible Operating System being used on the remote machine.

This method may not be successful each time as it entirely depends on the reply packets coming from the remote machine. Good security measures such as a firewall with robust rules would block the ping and telnet packets sent to the remote machine. As a result no replies would be generated.

Passive Operating System Detection

In this method, values for the parameters are obtained from the incoming packets captured from the sniffer. The 'Verbose' option of TCPDump provides more detailed information about the packets. Among the information provided about the packets, values for the parameters required to detect remote Operating System are also provided. As a result, in contrast to the active method of remote Operating System detection, there is no dependency on reply packets from the remote machine. Operating System detection is achieved from the initial packets coming from the remote machine.

Being completely passive in nature, there is very little chance that the person on the other side ever comes to know that the Operating System being used by him is being detected. This method is more reliable compared to the active method as there is very little dependency.

B. *Probable Location from IP Address*

A number of websites on the internet provide information about an IP address, specifically the location of the machine to which this address belongs and the service provider. The system developed uses the services of such websites to provide the user with the probable location of the machine. Earlier such websites used to give information about the location in plain text and numbers signifying latitude and longitude. But over the years they have become more content rich. For instance, most of the websites are integrated with Google maps. Thus they show information about the location on a map making it easier to understand. Providing information about the location from a number of websites is advantageous as the results can be correlated for improved accuracy.

C. *Information from WHOIS Database*

WHOIS provides a very useful set of information such as the administrative contact, owner of a domain, the Internet Service Provider. Websites such as arin.net provide information from WHOIS. WHOIS query is also present in most of today's Operating System so one just needs to type 'whois' followed by the IP address or domain name to get information.

As WHOIS provides information about owner of a domain, it is especially useful in cases where scans are coming from private organizations as information provided by them during domain registration is most likely to be present in the WHOIS database. Another valuable piece of information provided by WHOIS is the Internet Service Provider for a particular IP address. If a large number of scans are observed to be coming from a particular IP address, further details about the owner can be obtained from the ISP indicated by WHOIS.

D. *Information from Traceroute*

Traceroute is the program that shows the route over the network which packets take between two systems. It lists all the intermediate routers a connection must pass through to get to its destination. It is useful to have information about the possible route which the incoming packets from the scanning machine might have taken. Traceroute gives information about the various routers through which the packets travel from source to destination. This also gives clues about the location of the routers. The country to which the router belongs can be understood through the IP address of that router. Thus, traceroute gives clues about the location of the scanning machine.

E. *Self DIagnosis*

A very basic precaution which should be taken to secure a system is to close all the unnecessary ports. Operating Systems by default keep some ports open. In order to ensure maximum security the open ports which are not being used for any services or by any applications should be closed. Performing port scanning over our own machines is a very effective security measure.

F. *Identify Attacks Based On Ports Scanned*

Many attacks and exploits are performed on open ports. The pattern usually followed is to find out if a particular port is open and then execute the attack or carry out an exploit on that port. If a number of scans are observed to be coming on a particular port, it indicates that an attack or exploit on that port may be performed. Thus, analysing the ports on which repeated scans are being performed gives clues about what kind of an attack may follow on that particular port. The system developed shows the number of times a scan has been performed on a particular port and based on this the corresponding attack or exploit which the attacker may try.

G. Trends and Patterns

It has been observed that in most of the crimes patterns are present, computer crimes are no different. Security systems should be capable of identifying trends or patterns followed by attackers in performing attacks. This helps in taking precautionary steps to avoid attacks or scans in future. Also, this gives a chance to nab the attacker by recording his activities and gathering sufficient information about him.

V. EXPERIMENTAL SETUP

The system has been developed in the Linux environment using Fedora Core 5. Code for the system has been written in Perl.

Data was collected in a LAN environment by performing port scans on a machine with the system installed. Well known port scanners such as Nmap (both windows and Linux version), Angry IP, Megaping were used.

VI. RESULTS

Results are shown for scans performed over a period of time. Both Windows and Linux port of Nmap were used along with other por scanners such as AngryIP and MegaPing.

Time Interval	Number of packets
14:40 - 14:41	15
14:41 - 14:42	13
14:42 - 14:43	3744
14:43 - 14:44	4379
14:44 - 14:45	3180
14:45 - 14:46	3515
14:46 - 14:47	19
14:47 - 14:48	12
14:48 - 14:49	3508
14:49 - 14:50	3722
14:50 - 14:51	3725
14:50 - 14:51	3420

Figure 3. Number of incoming packets at different time intervals.

The table above shows sudden increase in network activity when normal operations were performed. The sudden surge in incoming packets is a case when scans are being performed.

A. Scan Detection

The system has successfully identified scans coming from most of the port scanners available today. This includes scans from popular port scanners such as Angry IP, Nmap, MegaPing. Scans which establish full connection between the two hosts as well as stealth scans which open a half connection are detected. Most of the different scans supported by Nmap are identified based on the type of flag which is set in the incoming packets.

The scans from port scanners were performed when the machine running the port scan detector was online. Basic tasks such as checking the E-mail, Internet messaging using messengers were also performed; the system did not show these activities as port scans.

Scan Type	Count
Syn	129
Fin	97
Ack	63
Tcp-Connect	107
Xmas	49

Figure 4. Count of different scans detected over a month.

B. Information Gathering

Remote Operating System Detection

Operating System being used on the remote machine are being identified using the two methods mentioned in Section 3. Results given by the passive methods are more accurate compared to that given by active methods. The TTL is one of the parameters used in active method to identify the Operating System. Some of the current generation port scanners give the ability to manipulate the TTL which would be sent in the outgoing packets. The TTL can also be modified using other methods, for instance, the default value of TTL can be changed in Windows from the registry entry using Regedit. Thus, sometimes the results may not be conclusive.

Location from IP Address

Using the services of websites the probable location of a machine is being shown with the help of IP address. Websites such as www.ip2location, www.melissadata.com have been used to show this information. Clues about the information are also obtained from Traceroute by examining the IP address of the routers through which the packet travels in order to reach the destination. The information provided by WHOIS also gives clues about the location of the machine to which the IP address belongs.

Information from WHOIS

The default WHOIS application in Fedora 5 has been used. It gives valuable information such as the Domain owner, the contact number of a person from the organization. This information is especially useful when scans come from an organization. As organizations usually have Internet presence in the form of websites, they are most likely to have some sort of information in the WHOIS database. As a result it is easier to identify the source from where scans are coming. Another useful piece of information which is provided is the Internet Service Provider

Information from Traceroute

The default Traceroute application in Fedora 5 has been used. It shows the IP address of the routers through which the packets pass in order to reach the destination machine. Asteric in the hop count gives an indication that the ICMP packets might be blocked by a firewall or the host is unreachable. The hop count and the IP address of the routers give an indication about the location of the destination machine.

Self Diagnosis

The system is capable of showing the ports open on the host system. The Perl module IO::Socket has been used to find out whether a specified port or range of ports are open or not.

Trends and Patterns

The system developed stores a report of the results provided by scan detection. Analysis of data such as time and day a particular scan was performed, from which IP the scan was performed, different ports on which the scans were performed helps in understanding the behaviour of the attacker. The system presents this information in graphical format in the form of bar graphs. Combination of different parameters such as the number of times scans came from a particular IP address, the types of scans performed on different days of the week give indication of the patterns followed by attackers.

Figure 5 shows a trend that more scans come on Saturdays than other days of the week. This gives the administrators a hint of when scans might be expected.

Syn	Sunday	30
	Monday	19
	Tuesday	0
	Wednesday	18
	Thursday	2
	Friday	12
	Saturday	48
Fin	Sunday	25
	Monday	8
	Tuesday	0
	Wednesday	0
	Thursday	13
	Friday	14
	Saturday	37
Xmas	Sunday	14
	Monday	0
	Tuesday	5
	Wednesday	7
	Thursday	0
	Friday	0
	Saturday	23

Figure 5. Count of number of times scans were detected on corresponding days of the week over a month.

Date	Scan Type	Number of Packets
19.04.2008	Syn	35640
19.04.2008	Ack	17223
19.04.2008	Fin	43463
20.04.2008	Syn	5203
21.04.2008	Syn	6542

Figure 6. Scan types and number of packets from 19.04.2008 to 21.04.2008.

IP Address	Day	Count
192.168.23.3	Tuesday	3
192.168.23.7	Saturday	9
192.168.23.8	Monday	39
192.168.23.10	Thursday	4
202.159.228.80	Tuesday	12
202.194.16.5	Monday	23

Figure 7. shows a trend that a large number of scans came from IP address 192.168.23.8 which is an internal address. In an organizational environment this would indicate that an employee from within the organization is performing scans.

VII. CONCLUSION

The system is designed to detect a possible port scan, getting additional information about the scanner such as his probable location, Operating System being used by attacker would help in uncovering the identity of the scanner.

For administrator it would be very helpful to have knowledge that someone is performing a port scans over a system. This gives a hint that some sort of attack might follow. Having an understanding of which ports are being scanned, it is possible to predict what kind of attack may follow. This helps the Administrator in taking precautionary steps before an attack takes place.

REFERENCES

[1] Fyodor, "The Art of Port Scanning", *Phrack Magazine, Volume 7, Issue 51, September 01 1997, Article 11 of 17.*

[2] Shaun Jamieson, “The Ethics and Legality of Port Scanning”, October 8, 2001. http://www.sans.org/reading_room/whitepapers/legal/71.php

[3] Fyodor, "Remote OS Detection using TCP/IP Fingerprinting (2nd Generation)", Jan 2007. http://nmap.org/osdetect/index.html#id287961

[4] Roger Christopher, “Port Scanning Techniques and the Defense Against Them”, October 5, 2001. http://www.sans.org/reading_room/whitepapers/auditing/70.php

[5] Kimberly Graves, “Official Certified Ethical Hacker Review Guide”, Wiley Publishing, pp.15-65, February 2007.

[6] Kocher, J.E.; Gilliam, D.P., “Self port scanning tool: providing a more secure computing environment through the use of proactive port scanning”, 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise, 13-15 June 2005.

[7] Nmap Reference Guide (Man Pages) http://nmap.org/man/

[8] TCPDump Reference Guide (Man Pages) http://www.tcpdump.org/tcpdump_man.html

Low-Rate TCP-Targeted Denial of Service Attacks
(The Shrew vs. the Mice and Elephants)[*] [†]

Aleksandar Kuzmanovic and Edward W. Knightly
ECE/CS Departments
Rice University
Houston, TX 77005, USA
{akuzma,knightly}@rice.edu

ABSTRACT

Denial of Service attacks are presenting an increasing threat to the global inter-networking infrastructure. While TCP's congestion control algorithm is highly robust to diverse network conditions, its implicit assumption of end-system cooperation results in a well-known vulnerability to attack by high-rate non-responsive flows. In this paper, we investigate a class of *low-rate* denial of service attacks which, unlike high-rate attacks, are difficult for routers and counter-DoS mechanisms to detect. Using a combination of analytical modeling, simulations, and Internet experiments, we show that maliciously chosen low-rate DoS traffic patterns that exploit TCP's retransmission time-out mechanism can throttle TCP flows to a small fraction of their ideal rate while eluding detection. Moreover, as such attacks exploit protocol homogeneity, we study fundamental limits of the ability of a class of randomized time-out mechanisms to thwart such low-rate DoS attacks.

Categories and Subject Descriptors

C.2.0 [**Security and Protection**]: Denial of Service;
C.2.2 **[Computer-Communication Networks]:** Network Protocols

General Terms

Algorithms, Performance, Security

Keywords

Denial of Service, TCP, retransmission timeout

[*]This research is supported by NSF ITR Grant ANI-0085842, NSF Special Projects Grant ANI-0099148, a Sloan Fellowship, and by HP Laboratories.
[†]A shrew is a small but aggressive mammal that ferociously attacks and kills much larger animals with a venomous bite.

1. INTRODUCTION

Denial of Service (DoS) attacks consume resources in networks, server clusters, or end hosts, with the malicious objective of preventing or severely degrading service to legitimate users. Resources that are typically consumed in such attacks include network bandwidth, server or router CPU cycles, server interrupt processing capacity, and specific protocol data structures. Example DoS attacks include TCP SYN attacks that consume protocol data structures on the server operating system; ICMP directed broadcasts that direct a broadcast address to send a flood of ICMP replies to a target host thereby overwhelming it; and DNS flood attacks that use specific weaknesses in DNS protocols to generate high volumes of traffic directed at a targeted victim.

Common to the above attacks is a large number of compromised machines or agents involved in the attack and a "sledge-hammer" approach of high-rate transmission of packets towards the attacked node. While potentially quite harmful, the high-rate nature of such attacks presents a statistical anomaly to network monitors such that the attack can potentially be detected, the attacker identified, and the effects of the attack mitigated (see for example, [6, 22, 30]).

In this paper, we study low-rate DoS attacks, which we term "shrew attacks," that attempt to deny bandwidth to TCP flows while sending at sufficiently low average rate to elude detection by counter-DoS mechanisms.

TCP congestion control operates on two timescales. On smaller timescales of round trip times (RTT), typically 10's to 100's of msec, TCP performs additive-increase multiplicative-decrease (AIMD) control with the objective of having each flow transmit at the fair rate of its bottleneck link. At times of severe congestion in which multiple losses occur, TCP operates on longer timescales of Retransmission Time Out (RTO).[1] In an attempt to avoid congestion collapse, flows reduce their congestion window to one packet and wait for a period of RTO after which the packet is resent. Upon further loss, RTO doubles with each subsequent timeout. If a packet is successfully received, TCP re-enters AIMD via slow start.

To explore low-rate DoS, we take a frequency-domain perspective and consider periodic on-off "square-wave" shrew attacks that consist of short, maliciously-chosen-duration bursts that repeat with a fixed, maliciously chosen, slow-timescale frequency. Considering first a single TCP flow, if the total traffic (DoS and TCP traffic) during an RTT-timescale burst is sufficient to induce enough packet losses, the TCP flow will enter a timeout and attempt to send a new packet RTO seconds later. If the period of the DoS flow approximates the RTO of the TCP flow, the TCP flow will continually incur loss as it tries to exit the timeout state, fail to exit timeout, and obtain near zero throughput. Moreover, if the DoS period is near but

[1]recommended minimum value 1 sec [1]

outside the RTO range, significant, but not complete throughput degradation will occur. Hence the foundation of the shrew attack is a null frequency at the relatively slow timescale of approximately RTO enabling a low average rate attack that is difficult to detect.

In a simplified model with heterogeneous-RTT aggregated flows sharing a bottleneck link, we derive an expression for the throughput of the attacked flows as a function of the timescale of the DoS flow, and hence of the DoS flow's average rate. Furthermore, we derive the "optimal" DoS traffic pattern (a two-level periodic square wave) that minimizes its average rate for a given level of TCP throughput for the victim, including zero throughput.

Next, we use ns-2 simulations to explore the impact of aggregation and heterogeneity on the effectiveness of the shrew attack. We show that even under aggregate flows with heterogeneous RTT's, heterogeneous file sizes, different TCP variants (New Reno, SACK, etc.), and different buffer management schemes (drop tail, RED, etc.), similar behavior occurs albeit with different severity for different flows and scenarios. The reason for this is that once the first brief outage occurs, all flows will simultaneously timeout. If their RTOs are nearly identical, they synchronize to the attacker's period and will enter a cycle identical to the single-flow case, even with heterogeneous RTTs and aggregation. However, with highly variable RTTs, the success of the shrew DoS attack is weighted such that small RTT flows will degrade far worse than large RTT flows, so that the attack has the effect of a high-RTT-pass filter. We show that in all such cases, detection mechanisms for throttling non-responsive flows such as RED-PD are not able to throttle the DoS attacker.

We then perform a set of Internet experiments in both a local and wide area environment. While necessarily small scale experiments (given that the expected outcome is to reduce TCP throughput to near zero), the experiments validate the basic findings and show that even a remote attacker (across a WAN) can dramatically reduce TCP throughput. For example, in the WAN experiments, a remote 909 kb/sec average-rate attack consisting of 100 ms bursts at the victim's RTO timescale reduced the victim's throughput from 9.8 Mb/sec to 1.2 Mb/sec.

Finally, we explore potential solutions to low rate DoS attacks. While it may appear attractive to remove the RTO mechanism all together or choose very small RTO values, we do not pursue this avenue as timeout mechanisms are fundamentally required to achieve high performance during periods of heavy congestion [1]. Instead, we consider a class of randomization techniques in which flows randomly select a value of minRTO such that they have random null frequencies. We use a combination of analytical modeling and simulation to show that such strategies can only distort and slightly mitigate TCP's frequency response to the shrew attack. Moreover, we devise an optimal DoS attack given that flows are randomizing their RTOs and show that such an attack is still quite severe.

In summary, vulnerability to low-rate DoS attacks is not a consequence of poor or easily fixed TCP design, as TCP necessarily requires congestion control mechanisms at both fast (RTT) and slow (RTO) timescales to achieve high performance and robustness to diverse network conditions. Consequently, it appears that such attacks can only be mitigated and not prevented through randomization. Development of prevention mechanisms that detect malicious low-rate flows remains an important area for future research.

2. TCP'S TIMEOUT MECHANISM

Here, we present background on TCP's retransmission timeout (RTO) mechanism [28]. TCP Reno detects loss via either timeout from non-receipt of ACKs, or by receipt of a triple-duplicate ACK. If loss occurs and less than three duplicate ACKs are received, TCP waits for a period of retransmission timeout to expire, reduces its congestion window to one packet and resends the packet.[2]

Selection of the timeout value requires a balance among two extremes: if set too low, spurious retransmissions will occur when packets are incorrectly assumed lost when in fact the data or ACKs are merely delayed. Similarly, if set too high, flows will wait unnecessarily long to infer and recover from congestion.

To address the former factor, Allman and Paxson experimentally showed that TCP achieves near-maximal throughput if there exists a lower bound for RTO of one second [1]. While potentially conservative for small-RTT flows, the study found that *all flows* should have a timeout value of at least 1 second in order to ensure that congestion is cleared, thereby achieving the best performance.

To address the latter factor, a TCP sender maintains two state variables, SRTT (smoothed round-trip time) and RTTVAR (round-trip time variation). According to [28], the rules governing the computation of SRTT, RTTVAR, and RTO are as follows. Until a RTT measurement has been made for a packet sent between the sender and receiver, the sender sets RTO to three seconds. When the first RTT measurement R' is made, the host sets SRTT $= R'$, RTTVAR $= R'/2$ and RTO $=$ SRTT $+ \max(G, 4\text{RTTVAR})$, where G denotes the clock granularity (typically ≤ 100 ms). When a subsequent RTT measurement R' is made, a host sets

$$\text{RTTVAR} = (1-\beta)\,\text{RTTVAR} + \beta\,|\text{SRTT} - R'|$$

and

$$\text{SRTT} = (1-\alpha)\text{SRTT} + \alpha\,R'$$

where $\alpha = 1/8$ and $\beta = 1/4$, as recommended in [15].

Thus, combining the two parts, a TCP sender sets its value of RTO according to

$$\text{RTO} = \max(\text{minRTO}, \text{SRTT} + \max(G, 4\,\text{RTTVAR})). \qquad (1)$$

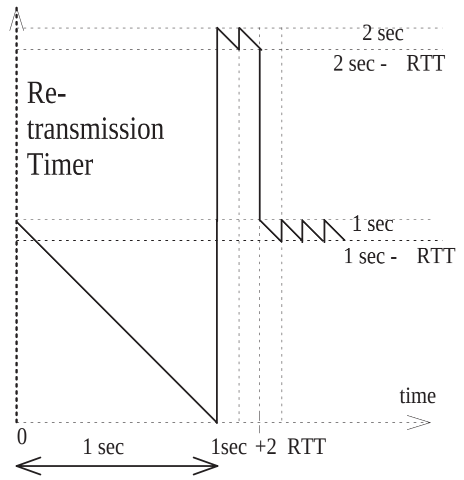


Figure 1: Behavior of the TCP retransmission timer

Finally, we illustrate RTO management via a *retransmission-timer* timeline in Figure 1. Assume that a packet with sequence number n is sent by a TCP sender at reference time $t = 0$, and that a retransmission timer of 1 second is initiated upon its transmission. If packet n is lost and fewer than three duplicate ACKs are

[2]Conditions under which TCP enters retransmission timeout vary slightly according to TCP version. We discuss this issue in Section 5.

received by the sender, the flow "times out" when the timer expires at $t = 1$ sec. At this moment, the sender enters the exponential backoff phase: it reduces the congestion window to one, doubles the RTO value to 2 seconds, retransmits the un-ACKed packet with sequence number n, and resets the retransmission timer with this new RTO value.

If the packet is lost again (*not* shown in Figure 1), exponential backoff continues as the sender waits for the 2 sec-long retransmission timer to expire. At $t = 3$ sec, the sender doubles the RTO value to 4 seconds and repeats the process.

Alternately, if packet n is successfully retransmitted at time $t = 1$ sec as illustrated in Figure 1, its ACK will arrive to the sender at time t=1+RTT. At this time, the TCP sender exits the exponential backoff phase and enters slow start, doubling the window size to two, transmitting two new packets $n+1$ and $n+2$, and reseting the retransmission timer with the current RTO value of 2 sec. If the two packets are not lost, they are acknowledged at time t= 1+2*RTT, and SRTT, RTTVAR and RTO are recomputed as described above. Provided that minRTO $>$ SRTT $+ \max(G, 4\,\text{RTTVAR})$, RTO is again set to 1 sec. Thus, in this scenario in which timeouts occur but exponential backoff does not, the value of RTO deviates by no more than RTT from minRTO for $t >$ minRTO $+$ 2 RTT.

3. DOS ORIGINS AND MODELING

In this section, we describe how an attacker can exploit TCP's timeout mechanism to perform a DoS attack. Next, we provide a scenario and a system model of such an attack. Finally, we develop a simple model for aggregate TCP throughput as a function of the DoS traffic parameters.

3.1 Origins

The above timeout mechanism, while essential for robust congestion control, provides an opportunity for low-rate DoS attacks that exploit the slow-timescale dynamics of retransmission timers. In particular, an attacker can provoke a TCP flow to repeatedly enter a retransmission timeout state by sending high-rate, but short-duration bursts having RTT-scale burst length, and repeating periodically at slower RTO timescales. The victim will be throttled to near-zero throughput while the attacker will have low average rate making it difficult for counter-DoS mechanisms to detect.

We refer to the short durations of the attacker's loss-inducing bursts as *outages*, and present a simple but illustrative model relating the outage timescale (and hence attacker's average rate) to the victim's throughput as follows.

First, consider a single TCP flow and a single DoS stream. Assume that an attacker creates an initial outage at time 0 via a short-duration high-rate burst. As shown in Figure 1, the TCP sender will wait for a retransmission timer of 1 sec to expire and will then double its RTO. If the attacker creates a second outage between time 1 and 1+2RTT, it will force TCP to wait another 2 sec. By creating similar outages at times 3, 7, 15, $\cdots$, an attacker could deny service to the TCP flow while transmitting at extremely low average rate.

While potentially effective for a single flow, a DoS attack on TCP aggregates in which flows continually arrive and depart requires periodic (vs. exponentially spaced) outages at the minRTO timescale. Moreover, if all flows have an identical minRTO parameter as recommended in RFC 2988 [28], the TCP flows can be forced into continual timeouts if an attacker creates periodic outages.

Thus, we consider "square wave" shrew DoS attacks as shown in Figure 3 in which the attacker transmits bursts of duration l and rate R in a deterministic on-off pattern that has period T. As explored below, a successful shrew attack will have rate R large enough to induce loss (i.e., R aggregated with existing traffic must exceed

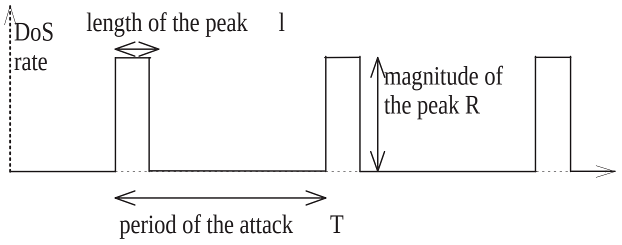


Figure 3: Square-wave DoS stream

the link capacity), duration l of scale RTT (long enough to induce timeout but short enough to avoid detection), and period T of scale RTO (chosen such that when flows attempt to exit timeout, they are faced with another loss).

3.2 Model

Consider a scenario of an attack shown in Figure 2(a). It consists of a single bottleneck queue driven by n long-lived TCP flows with heterogeneous RTTs and a single DoS flow. Denote RTT$_i$ as the roundtrip time of the i-th TCP flow, $i = 1, \cdots, n$. The DoS flow is a periodic square-wave DoS stream shown in Figure 3. The following result relates the throughput of the TCP flows to the period of the attack.

DoS TCP Throughput Result. *Consider a periodic DoS attack with period T. If the outage duration satisfies*
(C1) $l' \geq RTT_i$
and the minimum RTO satisfies
(C2) $minRTO > SRTT_i + 4 * RTTVAR_i$
for all $i = 1, \cdots, n$, then the normalized throughput of the aggregate TCP flows is approximately

$$\rho(T) = \frac{\lceil \frac{minRTO}{T} \rceil T - minRTO}{T}. \quad (2)$$

This result is obtained as follows. As shown in Figure 2(b), the periodic l-length bursts create short l'-length outages having high packet loss.[3] If l' reaches the TCP flows' RTT timescales, i.e., $l' \geq RTT_i$, for all $i = 1, \cdots, n$, then the congestion caused by the DoS burst lasts sufficiently long to force *all* TCP flows to simultaneously enter timeout. Moreover, if minRTO $>$ SRTT$_i + 4*$RTTVAR$_i$, for $i = 1, \cdots, n$, all TCP flows will have identical values of RTO and will thus timeout after minRTO seconds, which is the ideal moment for an attacker to create a new outage. Thus, in this case, despite their heterogeneous round-trip times, all TCP flows are forced to "synchronize" to the attacker and enter timeout at (nearly) the same time, and attempt to recover at (nearly) the same time. Thus, when exposed to outages with period T, Equation (2) follows. Note also that in Equation (2) we do not model throughput losses due to the slow-start phase, but simply assume that TCP flows utilize all available bandwidth after exiting the timeout phase.

Moreover, in the model, the aggregate TCP traffic is assumed to utilize the full link bandwidth after the end of each retransmission timeout and the beginning of the following outage. Observe that if the period T is chosen such that $T \geq 1 + 2\,RTT_i$, all TCP flows will continually enter a retransmission timeout of 1 sec duration. Thus, because Equation (2) assumes that RTO $=$ minRTO for $T >$ minRTO, while this is not the case in the period (minRTO, minRTO $+$ 2 RTT), Equation (2) behaves as an *upper bound* in practice. In other words, periodic DoS streams are not

[3]The relationship between l and l' is explored in Section 4.

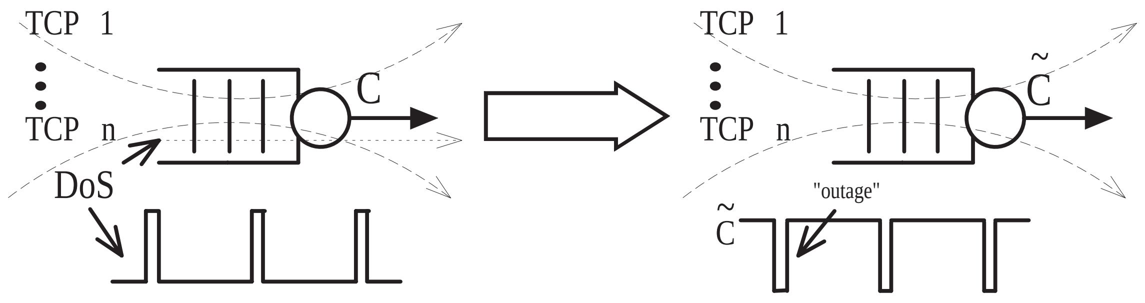


Figure 2: DoS scenario and system model

utilizing TCP's exponential backoff mechanism but rather exploit repeated timeouts.

Next, we consider flows that do not satisfy conditions (C1) or (C2).

DoS TCP Flow-Filtering Result. *Consider a periodic DoS attack with period T. If the outage duration $l' \geq RTT_i$ and minRTO $>$ SRTT$_i + 4 *$ RTTVAR$_i$ for $i = 1, \cdots, k$ whereas $l' < RTT_j$ or minRTO $\leq$ SRTT$_j + 4 *$ RTTVAR$_j$ for $j = k+1, \cdots, n$, then Equation (2) holds for flows $1, \cdots, k$.*

This result, shown similarly to that above, states that Equation (2) holds for *any* TCP sub-aggregate for which conditions (C1) and (C2) hold. In other words, if a shrew DoS attack is launched on a group of flows such that only a subset satisfies the two conditions, that subset will obtain degraded throughput according to Equation (2), whereas the remaining flows will not. We refer to this as "flow filtering" in that such an attack will deny service to a subset of flows while leaving the remainder unaffected, or even obtaining higher throughput. We explore this issue in detail in Section 5.

3.3 Example

Here, we present a baseline set of experiments to explore TCP's "frequency response" to shrew attacks. We first consider the analytical model and the scenario depicted in Figure 2 in which conditions (C1) and (C2) are satisfied and minRTO = 1 sec. The curve labeled "model" in Figure 4 depicts ρ vs. T as given by Equation (2). Throughput is normalized to the link capacity, which under high aggregation, is also the throughput that the TCP flows would obtain if no DoS attack were present.

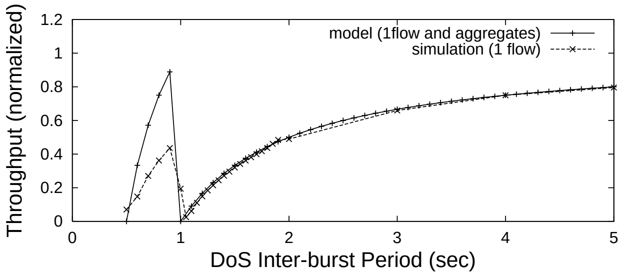


Figure 4: DoS TCP throughput: model and simulation

Note that the average rate of the DoS attacker is decreasing with increasing T as its average rate is given by Rl/T. However, as indicated by Equation (2) and Figure 4, the effectiveness of the attack is clearly *not* increasing with the attacker's average rate. Most critically, observe that there are two "nulls" in the frequency response in which TCP throughput becomes *zero*. In particular, $\rho(T) = 0$ when $T =$ minRTO and $T =$ minRTO$/2$. The physical interpretation is as follows: if the attacker creates the minRTO-periodic outages, it will completely deny service to the TCP traffic. Once the brief outage occurs, all flows will simultaneously timeout. When their timeout expires after minRTO seconds and they again transmit packets, the attacker creates another outage such that the flows backoff again. Clearly, the most attractive period for a DoS attacker is *minRTO* (vs. minRTO$/2$), since it is the null frequency that minimizes the DoS flow's average rate. When $T >$ *minRTO*, as the period of the attack increases, the TCP flows obtain increasingly higher throughput during durations between expiration of retransmission timers and the subsequent DoS outage.

Next, we perform a set of ns-2 simulations to compare against the model. In these experiments, we again consider the scenario of Figure 2 but with a single TCP flow.[4] The TCP Reno flow has minRTO = 1 second and satisfies conditions (C1) and (C2). More precisely, the propagation delay is 6 ms while the buffer size is set such that the round-trip time may vary from 12 ms to 132 ms. The link capacity is 1.5 Mb/s, while the DoS traffic is a square-wave stream with the peak rate 1.5 Mb/s and burst length 150 ms.

The curve labeled "simulation" in Figure 4 depicts the measured normalized throughput of the TCP flow. Figure 4 reveals that Equation (2) captures the basic frequency response of TCP to the shrew DoS attack, characterizing the general trends and approximating the location of the two null frequencies. Observe that the model overestimates measured TCP throughput between the two nulls because the model assumes that TCP can utilize the full link capacity between the end of an RTO and the occurrence of the new outage, which is not the case due to slow-start.

4. CREATING DOS OUTAGES

In this section, we explore the traffic patterns that attackers can use in order to create temporary outages that induce recurring TCP timeouts. First, we study the instantaneous bottleneck-queue behavior in periods when an attacker bursts packets into the network. Next, we develop the DoS stream which minimizes the attacker's average rate while ensuring outages of a particular length. Finally, we study square-wave DoS streams and identify the conditions in which they accurately approximate the optimal double-rate DoS streams.

4.1 Instantaneous Queue Behavior

Consider a bottleneck queue shared by a TCP flow and a DoS flow which every T seconds bursts at a constant rate R_{DoS} for duration l. Denote R_{TCP} as the instantaneous rate of the TCP flow, B as the queue size, and B_0 as the queue size at the onset of

[4]Recall that Equation (2) holds for any number of flows. We simulate TCP aggregates in Section 5.

an attack, assumed to occur at $t = 0$.

Denote l_1 as the time that the queue becomes full such that

$$l_1 = \frac{(B - B_0)}{R_{DoS} + R_{TCP} - C}. \qquad (3)$$

After l_1 seconds, the queue remains full for $l_2 = l - l_1$ seconds if $R_{DoS} + R_{TCP} \geq C$. Moreover, if $R_{DoS} \geq C$ during the same period, this will create an outage to the TCP flow whose loss probability will instantaneously increase significantly and force the TCP flow to enter a retransmission timeout with high probability (see also Figure 2).

4.2 Minimum Rate DoS Streams

Suppose the attacker is limited to a peak rate of $R_{\max}$ due to a secondary bottleneck or the attacker's access link rate. To avoid router-based mechanisms that detect high rate flows, e.g., [22], DoS attackers are interested in ways to minimally expose their streams to detection mechanisms. To minimize the number of bytes transmitted while ensuring outages of a particular length, an attacker should transmit a double-rate DoS stream as depicted in Figure 5. To fill the buffer without help from background traffic or the attacked flow requires $l_1 = B/(R_{\max} - C)$ seconds. Observe that sending at the maximum possible rate $R_{\max}$ minimizes l_1 and consequently the number of required bytes. Once the buffer fills, the attacker should reduce its rate to the bottleneck rate C to ensure continued loss using the lowest possible rate.

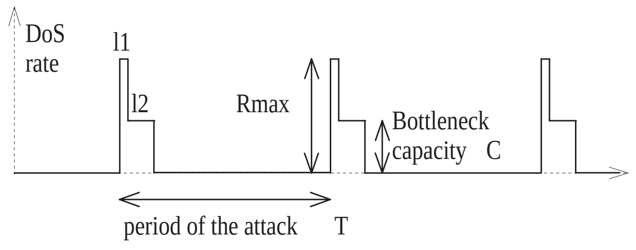


Figure 5: Double-rate DoS stream

Thus, double-rate streams *minimize* the number of packets that need to be transmitted (for a given bottleneck queue size B, bottleneck capacity C, and range of sending rates from 0 to $R_{\max}$) among all possible sending streams that are able to ensure periodic outages with period T and length l_2.

To generate double-rate DoS streams in real networks, an attacker can use a number of existing techniques to estimate the bottleneck link capacity [3, 4, 16, 19, 27], bottleneck-bandwidth queue size [21] and secondary bottleneck rate [26].

Regardless of the optimality of double-rate DoS streams, we consider the simpler square-wave DoS attack shown in Figure 3 as an approximation. First, these streams do not require prior knowledge about the network except the bottleneck rate. Second, they isolate the effect of a single timescale periodic attack.

To study the effectiveness of the square-wave, we perform simulation experiments to compare the two attacks' frequency responses. As an example, we consider a square-wave DoS stream with peak rate 3.75 Mb/s and burst length $l = 50$ ms and a double-rate stream with $R_{\max} = 10$ Mb/s. For the double-rate stream, l_1 is computed as $B/(R_{\max} - C)$, while l_2 is determined such that the number of packets sent into the network is the same for both streams. The simulation parameters are the same as previously.

The resulting frequency responses in this example and others (not shown) are nearly identical. Consequently, since square-wave DoS streams accurately approximate the double-rate DoS stream and do not require knowledge of network parameters, we use square-wave DoS streams henceforth in both simulations and Internet experiments.

5. AGGREGATION AND HETEROGENEITY

In this section, we explore the impact of TCP flow aggregation and heterogeneity on the effectiveness of the shrew DoS attack. First, we experiment with long-lived homogeneous-RTT TCP traffic and explore the DoS stream's ability to synchronize flows. Second, we perform experiments in a heterogeneous RTT environment and explore the effect of RTT-based filtering. Third, we study the impact of DoS streams on links dominated by web traffic. Finally, we evaluate several TCP variants' vulnerability to the shrew DoS attacks.

As a baseline topology (and unless otherwise indicated) we consider many flows sharing a single congested link with capacity 1.5 Mb/s as in Figure 2. The one-way *propagation* delay is 6 ms and the buffer size is set such that the *round-trip* time varies from 12 ms to 132 ms. The DoS traffic is a square-wave stream with peak rate 1.5 Mb/s, burst duration 100 ms, and packet size 50 bytes. In all experiments, we generate a FTP/TCP flow in the reverse direction, whose ACK packets multiplex with TCP and DoS packets in the forward direction. For each data point in the figures below, we perform five simulation runs and report averages. Each simulation run lasts 1000 sec.

5.1 Aggregation and Flow Synchronization

The experiments of Section 3 illustrate that a DoS square wave can severely degrade the throughput of a *single* TCP flow. Here, we investigate the effectiveness of low bit-rate DoS streams on TCP aggregates with homogeneous RTTs for five long-lived TCP flows sharing the bottleneck.

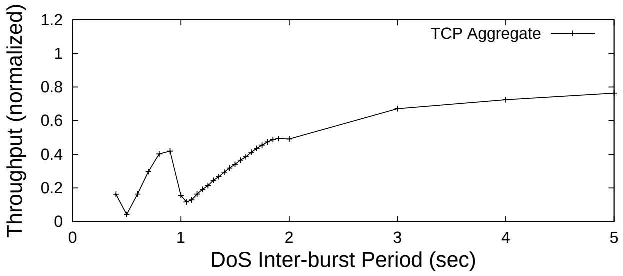


Figure 6: DoS and aggregated TCP flows

Figure 6 depicts the normalized *aggregate* TCP throughput under the shrew DoS attack for different values of the period T. Observe that similar to the one-flow case, the attack is highly successful so that Equation (2) can also model attacks on aggregates. However, we note that compared to the single-flow case, the throughput at the null 1/RTO frequency is slightly larger in this case because the maximum RTT of 132 ms is greater than the DoS burst length of 100 ms such that a micro-flow may survive an outage. Also observe that an attack at frequency 2/minRTO nearly completely eliminates the TCP traffic.

The key reasons for this behavior are twofold. First, *RTO homogeneity* (via minRTO) introduces a single vulnerable timescale, even if flows have different RTTs (as explored below). Second, *DoS-induced synchronization* occurs when the DoS outage event causes all flows to enter timeout nearly simultaneously. Together with RTO homogeneity, flows will also attempt to exit timeout nearly simultaneously when they are re-attacked.

Synchronization of TCP flows was extensively explored in [10, 31] and was one of the main motivations for RED [11], whose goal is the avoidance of synchronization of many TCP flows decreasing their window at the same time. In contrast, the approach and scenario here are quite different, as an external malicious source (and not TCP itself) is the source of synchronization. Consequently, mechanisms like RED are unable to prevent DoS-initiated synchronization (see also Section 7).

5.2 RTT Heterogeneity

5.2.1 *RTT-based Filtering*

The above experiment shows that a DoS stream can significantly degrade throughput of a TCP aggregate, provided that the outage length is long enough to force all TCP flows to enter a retransmission timeout simultaneously. Here, we explore a heterogeneous-RTT environment with the objective of showing that a flow's vulnerability to low-rate DoS attacks fundamentally depends on its RTT, with shorter-RTT flows having increased vulnerability.

We perform experiments with 20 long-lived TCP flows on a 10 Mb/s link. The range of round-trip times is 20 to 460 ms [12], obtained from representative Internet measurements [18]. We use these measurements to guide our setting of link propagation delays for different TCP flows.[5]

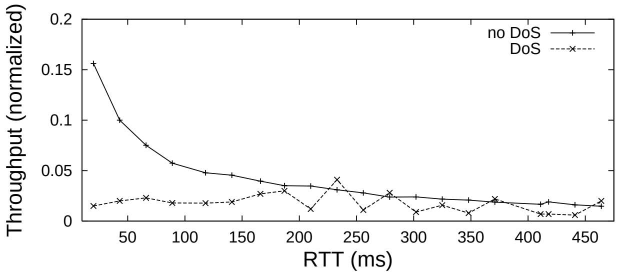


Figure 7: RTT-based filtering

Figure 7 depicts the normalized TCP throughput for each of the 20 TCP flows. The curve labeled "no DoS" shows each flow's throughput in the absence of an attack. Observe that the flows redistribute the bandwidth proportionally to 1/RTT such that shorter-RTT flows utilize more bandwidth than the longer ones. The curve labeled "DoS" shows each TCP flow's throughput when they are multiplexed with a DoS square-wave stream with peak rate 10 Mb/s, burst length 100 ms and period 1.1 sec. Observe that this DoS stream filters shorter-RTT flows up to a timescale of approximately 180 ms, beyond which higher RTT flows are less adversely affected. Also, observe that despite the excess capacity available due to the shrew DoS attack, longer-RTT flows do not manage to improve their throughput.

However, in a regime with many TCP flows with heterogeneous RTTs, the *number* of non-filtered flows with high RTT will increase, and they will eventually be of sufficient number to utilize all available bandwidth left unused by the filtered smaller-RTT flows. Thus, the total TCP throughput will increase with the aggregation level for highly heterogeneous-RTT flows as illustrated in Figure 8. Unfortunately, the high throughput and high link utilization with many flows (e.g., greater than 90% in the 80-flow scenario) is quite misleading, as the shorter-RTT flows have been dramatically rate-limited by the attack as in Figure 7. Hence, one can simultaneously have high utilization and an effective DoS attack against small- to moderate-RTT flows.

[5]We did not fit the actual CDF of this data, but have uniformly distributed round-trip times in the above range.

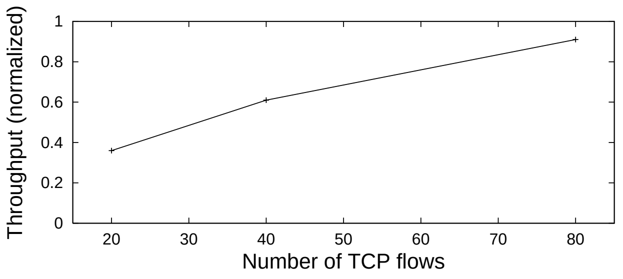


Figure 8: High aggregation with heterogeneous RTT

5.2.2 *DoS Burst Length*

The above experiments showed that DoS streams behave as a high-RTT-pass filter, in which the burst length is related to the filter cut-off timescale. Here, we directly investigate the impact of burst length.

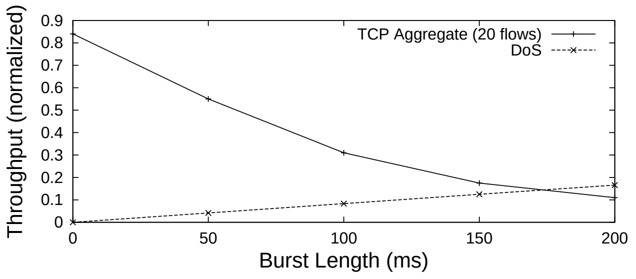


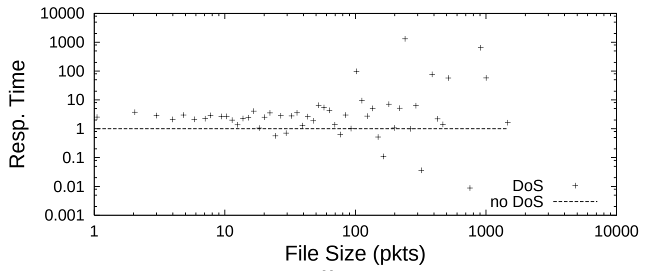
Figure 10: Impact of DoS burst length

For the same parameters as above, Figure 10 depicts aggregate TCP throughput as a function of the DoS burst length. The figure shows that as the burst length increases, the DoS mean rate increases, yet the aggregate TCP throughput decreases much more significantly. Indeed, as the burst length increases, the RTT-cut-off timescale increases. In this way, flows with longer and longer RTTs are filtered. Consequently, the number of non-filtered flows decreases such that aggregate TCP throughput decreases. In other words, as the burst length increases, the sub-aggregate for which condition (C1) holds enlarges. With a fixed number of flows, the longer-RTT flows are unable to utilize the available bandwidth, and the aggregate TCP throughput decreases.

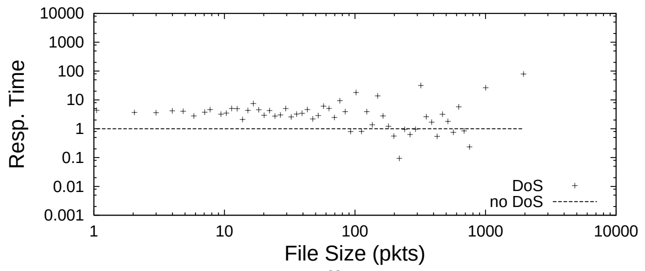
5.2.3 *Peak Rate*

Recall that the minimal-rate DoS streams studied in Section 4 induce outages without any help from background traffic and under the assumption that the initial buffer size B_0 is zero. However, in practice, the buffer will also be occupied by packets from reverse ACK traffic, UDP flows, etc. Consequently, in the presence of such background traffic, the DoS source can potentially lower its peak rate and yet maintain an effective attack.

Consider a scenario with five flows, a DoS flow and four long-lived TCP flows. We set the link propagation delays in the simulator such that one TCP flow experiences shorter RTT (fluctuates from 12 ms to 134 ms) while the other three have longer RTTs (from 108 ms to 230 ms). Figure 11 depicts the throughput of the



(a) 50% Offered Load



(b) 100% Offered Load

Figure 9: Impact on HTTP flows

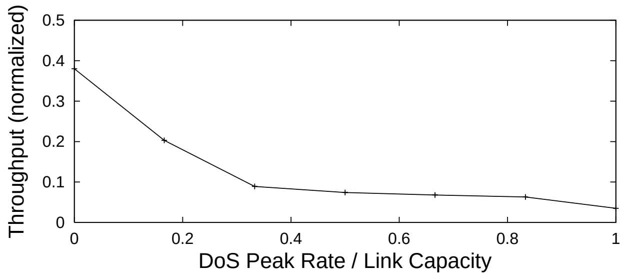


Figure 11: Impact of DoS peak rate

short-RTT flow as a function of the normalized DoS peak rate varied from 0 to 1. Observe that relatively low peak rates are sufficient to filter the short-RTT flow. For example, a peak rate of one third of the link capacity and hence an average rate of 3.3% of the link capacity significantly degrades the short-RTT flows' throughput at the null timescale. As hypothesized above, longer RTT flows here play the role of background traffic and increase both B_0 and the burst rate in periods of outages which enables lower-than-bottleneck peak DoS rates to cause outages. This further implies that very low rate periodic flows that operate at one of the null TCP timescales ($\frac{minRTO}{j}$, $j = 1, \cdots$) are highly problematic for TCP traffic. For example, some probing schemes periodically burst for short time intervals at high rates in an attempt to estimate the available bandwidth on an end-to-end path [17].

5.3 HTTP Traffic

Thus far, we have considered long-lived TCP flows. Here, we study a scenario with flow arrival and departure dynamics and highly variable file sizes as incurred with HTTP traffic.

We adopt the model of [8] in which clients initiate sessions from randomly chosen web sites with several web pages downloaded from each site. Each page contains several objects, each of which requires a TCP connection for delivery (i.e., HTTP 1.0). The inter-page and inter-object time distributions are exponential with respective means of 9 sec and 1 msec. Each page consists of ten objects and the object size is distributed according to a Pareto distribution with shape parameter 1.2. For the web transactions, we measure and average the response times for different sized objects.

Figure 9 depicts web-file response times normalized by the response times obtained when the DoS flow is not present in the system. Because of this normalization, the curve labeled "no DoS" in Figure 9 is a straight line with a value of one. The flows' mean HTTP request arrival rate is selected such that the offered HTTP load is 50% and near 100% for Figures 9(a) and 9(b) respectively.

On average, the file response times increased by a factor of 3.5 under 50% load and a factor of 5 under 100% load. Figures 9(a) and (b) both indicate that larger files (greater than 100 packets in this scenario) become increasingly and highly vulnerable to the shrew DoS attacks with the response times of files increasing by orders of magnitude. However, observe that some flows benefit from the shrew attack and significantly decrease their response times. This occurs when a flow arrives into the system between two outages and is able to transmit its entire file before the next outage occurs.

Next, observe that the deviation from the reference (no DoS) scenario is larger in Figure 9(a) than 9(b). This is because the response times are approximately 100 times lower for the no-DoS scenario when the offered load is 50% as compared to the no-DoS scenario when the system is fully utilized.

Finally, we performed experiments where DoS stream attack mixtures of long- (FTP) and short-lived (HTTP) TCP flows. The results (not shown) indicate that the conclusions obtained separately for FTP and HTTP traffic hold for FTP/HTTP aggregates.

5.4 TCP Variants

The effectiveness of low-rate DoS attacks depends critically on the attacker's ability to create correlated packet losses in the system and force TCP flows to enter retransmission timeout. While we have studied TCP Reno thus far, a large body of work has been done to help TCP flows survive multiple packet losses within a single round trip time without incurring a retransmission timeout. For example, New Reno [14] changes the sender's behavior during Fast Recovery upon receipt of a *partial ACK* that acknowledges some but not all packets that were outstanding at the start of the Fast Recovery period. Further improvements are obtained by TCP SACK [13] when a large number of packets are dropped from a window of data [7] because when a SACK receiver holds non-contiguous data, it sends duplicate ACKs bearing the SACK option to inform the sender of the segments that have been correctly received. A thorough analysis of the packet drops required to force flows of a particular TCP version to enter timeout is given in [7].

Here, we evaluate the performance of TCP Reno, New Reno, Tahoe and SACK under the shrew DoS attack. Figures 12 (a)-(d) show TCP throughput for burst lengths of 30, 50, 70 and 90 ms, respectively. Figure 12(a) confirms that TCP Reno is indeed the most fragile TCP variant, while the other three versions have better robustness to DoS. However, when the peak length increases to 50 ms, *all* TCP variants obtain near zero throughput at the null frequency as shown in Figure 12(b). The Figure also indicates that TCP is the most vulnerable to DoS in the 1 - 1.2 sec timescale region. During this period, TCP flows are in slow-start and have small window sizes such that a smaller number of packet losses are needed to force them to enter retransmission timeout. Finally, Figures (c)-(d) indicate that all TCP variations obtain a throughput profile similar to Equation (2) when the outage duration increases,

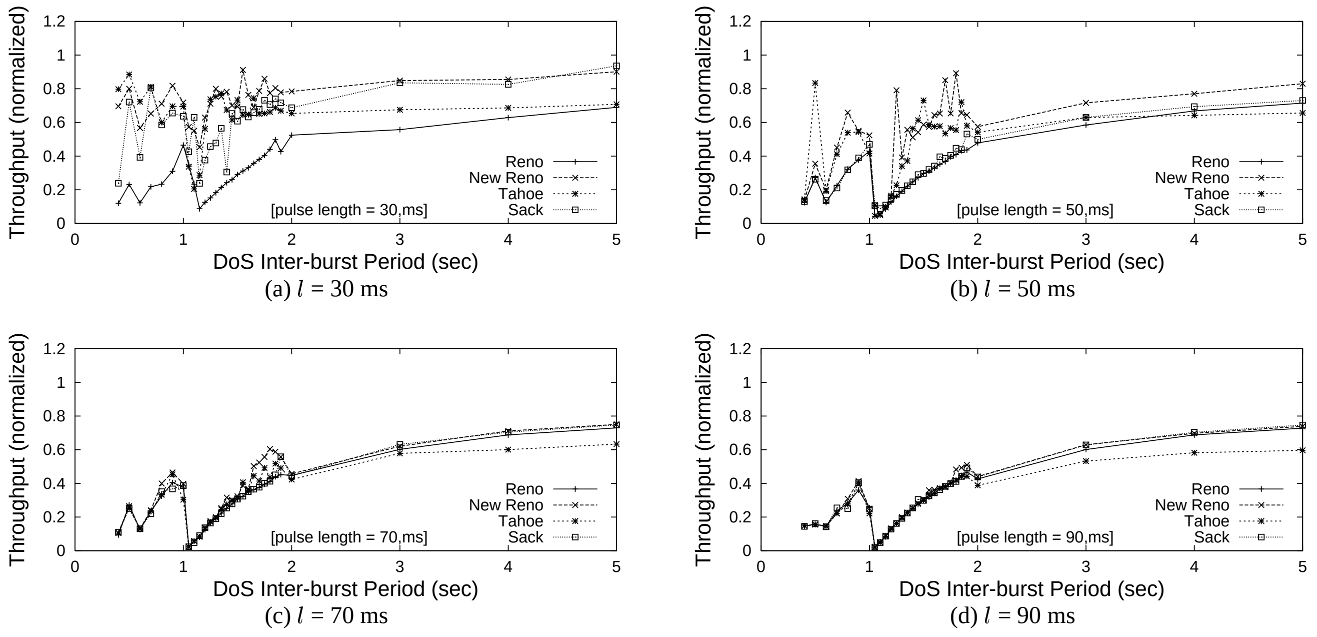


Figure 12: TCP Reno, New Reno, Tahoe and SACK under shrew DoS attacks

such that more packets are lost from the window of data. Indeed, if all packets from the window are lost, TCP has no alternative but to wait for a retransmission timer to expire.

6. INTERNET EXPERIMENTS

In this section, we describe several DoS experiments performed on the Internet. The scenario is depicted in Figure 13 and consists of a large file downloaded from a TCP SACK sender (TCP-S) to a TCP SACK receiver (TCP-R). We configured the TCP-S host to have $minRTO$=1 sec according to [28] and measured TCP throughput using *iperf*. The shrew DoS attack was launched from three different hosts using UDP-based active probing software from [25] in order to send high-precision DoS streams. All experiments are performed three times and averages are reported.

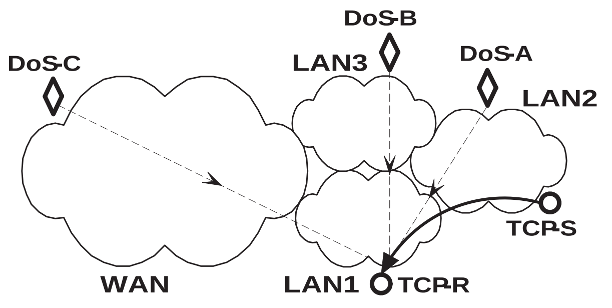


Figure 13: DoS attack scenario

Intra-LAN Scenario. In this scenario, both the TCP sender (TCP-S) and DoS (DoS-A) hosts are on the same 10 Mb/s Ethernet LAN on Rice University, while the attacked host (TCP-R) is on a different 10 Mb/s Ethernet LAN, two hops away from both TCP-S and DoS-A. The peak rate of the square-wave DoS stream is 10 Mb/s while the burst length is 200 ms. The curve labeled "DoS-A (Intra-LAN)" in Figure 14 depicts the results of these experiments. The figure indicates that a null frequency exists at a timescale of approximately 1.2 sec. When the attacker transmits at this period, it has an average rate of 1.67 Mb/s. Without the DoS stream, the TCP flow obtains 6.6 Mb/s throughput. With it, it obtains 780 kb/s throughput. Thus, the DoS attacker can severely throttle the victim's throughput by nearly an order of magnitude.

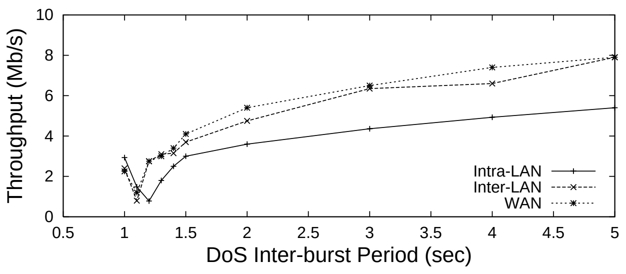
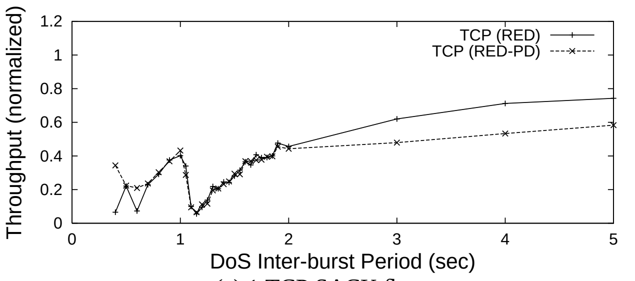


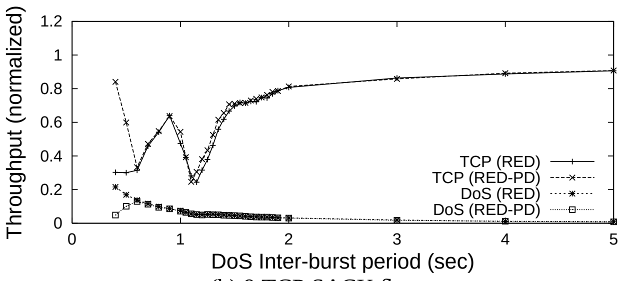
Figure 14: Internet experiments

Inter-LAN Scenario. In this, the TCP sender (TCP-S), DoS source (DoS-B) and attacked host (TCP-R) are on three different LANs of the ETH (Zurich, Switzerland) campus network. The route between the two traverses two routers and two Ethernet switches, with simple TCP measurements revealing that the TCP and DoS LANs are 100 Mb/s Ethernet LANs, while the attacked host is on a 10 Mb/s Ethernet LAN. The peak rate of the square-wave DoS stream is again 10 Mb/s while its duration is reduced as compared to the Intra-LAN Scenario to 100 ms. The curve labeled "DoS-B inter-LAN" in Figure 14 depicts the frequency response of this attack. In this case, a DoS timescale of $T = 1.1$ sec is the most damaging to TCP, since here the TCP flow achieves 800 kb/s throughput, only 8.1% of the throughput it achieves without DoS flow (9.8 Mb/s). At this timescale, the attacker has an average rate of 909 kb/s.

WAN Scenario. Finally, for the same TCP source/destination pair as in the Inter-LAN Scenario, source DoS-C initiates a shrew DoS attack from a LAN at EPFL (Lausanne, Switzerland), located



(a) 1 TCP SACK flow



(b) 9 TCP SACK flows

Figure 15: Impact of RED and RED-PD routers

eight hops away from the destination. The DoS stream has a peak rate of 10 Mb/s and a burst duration of 100 ms. The curve labeled "DoS-C (WAN)" shows the frequency response of these experiments and indicates a nearly identical null located at $T = 1.1$ sec. For this attack, the TCP flow's throughput is degraded to 1.2 Mb/s from 9.8 Mb/s whereas the attacker has average rate of 909 kb/s. This experiment illustrates the feasibility of *remote* attacks. Namely, in the WAN Scenario, the DoS attacker has traversed the local provider's network and multiple routers and Ethernet switches before reaching its victim's LAN. Thus, despite potential traffic distortion that deviates the attacker's traffic pattern from the square wave, the attack is highly effective.

Thus, while necessarily small scale due to their (intended) adverse effects, the experiments support the findings of the analytical model and simulation experiments. The results indicate that effective shrew attacks can come from remote sites as well as nearby LANs.

7. COUNTER-DOS TECHNIQUES

Here, we explore two classes of candidate counter-DoS mechanisms intended to mitigate the effects of shrew attacks: router-assisted detection and throttling, and endpoint-based randomization.

7.1 Router-Assisted Mechanisms

As described above, DoS flows have low average rate, yet do send relatively high-rate bursts for short time intervals. Here, we investigate if such traffic patterns can be identified as a DoS attack by router-based algorithms.

Mechanisms for per-flow treatment at the router can be classified as scheduling or preferential dropping. Due to implementation simplicity and other advantages of preferential dropping over scheduling (see reference [22]), we concentrate on dropping algorithms for detection of DoS flows and/or achieving fairness among adaptive and non-adaptive flows. Candidate algorithms include Flow Random Early Detection (FRED) [20], CHOKe [24], Stochastic Fair Blue (SFB) [9], the scheme of reference [2], ERUF [29], Stabilized RED (SRED) [23], dynamic buffer-limiting scheme from [5] and RED with Preferential Dropping (RED-PD) [22]. Of these, we study RED-PD as it uses the packet drop history at the router to detect high-bandwidth flows with high confidence. Flows above a configured target bandwidth are identified and monitored by RED-PD. Packets from the monitored flows are dropped with a probability dependent on the excess sending rate of the flow. RED-PD suspends preferential dropping when there is insufficient demand from other traffic in the output queue, for example, when RED's average queue size is less than the minimum threshold.

We perform simulation experiments with one and nine TCP SACK flows, RED-PD routers, and the topology of Figure 2. For one TCP flow, Figure 15(a) indicates that RED-PD is *not* able to detect nor throttle the DoS stream. For aggregated flows depicted in Figure 15(b), RED-PD only affects the system if the attack occurs at a timescale of less than 0.5 sec, i.e., only unnecessarily high-rate attacks can be addressed. Most critically, at the null timescale of 1.2 sec, RED-PD has no noticeable effect on throughput as compared to RED. Thus, while RED and RED-PD's randomization has lessened the severity of the null, the shrew attack remains effective overall.

Next, in the above scenario with nine TCP SACK flows, we vary the DoS peak rate and burst length to study the conditions under which the DoS flows will become detectable by RED-PD. We first set the burst duration to 200 ms and then change the peak rate from 0.5 Mb/s to 5 Mb/s. Figure 16(a) indicates that RED-PD starts detecting and throttling the square-wave stream at a peak rate of 4 Mb/s, which is more than twice than the bottleneck rate of 1.5 Mb/s. Recall that in Section 5.2.3 we showed that a peak rate of one third the bottleneck capacity and a burst length of 100 ms can be quite dangerous for short-RTT TCP flows.

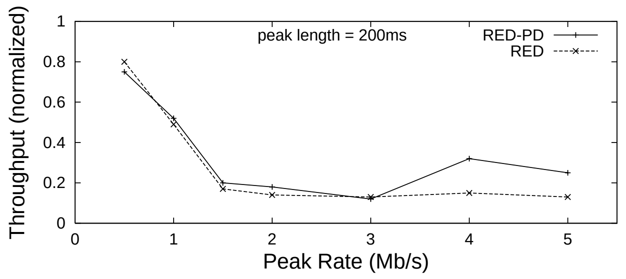
Further, we fix the DoS peak rate to 2 Mb/s and vary the burst length from 50 ms to 450 ms. Figure 16(b) shows that RED-PD begins detecting the DoS flow at 300 ms timescales in this scenario. Recall again that much shorter burst timescales are sufficient to throttle not only short-RTT flows, but the entire aggregates of heterogeneous-RTT TCP traffic.

Thus, Figure 16(b) captures the fundamental issue of timescales: RED-PD detects high rate flows on longer timescales, while DoS streams operate at very short timescales. If these shorter timescales are used to detect malicious flows in the Internet, many legitimate bursty TCP flows would be incorrectly detected as malicious. This issue is studied in depth in reference [22], which concludes that long timescale detection mechanisms are needed to avoid excessively high false positives. Therefore, while short timescale mechanisms such as [24, 20, 9, 5] may indeed be more effective at mitigating shrew attacks, [22] indicates that the penalty for their use may be quite high.

In summary, relatively long-timescale measurements are required to determine with confidence that a flow is transmitting at excessively high rate and should be dropped. Because DoS attacks can be of short RTT-scale duration, detection of low-rate DoS attacks is a fundamentally difficult task.

7.2 End-point minRTO Randomization

Since low-rate attacks exploit $minRTO$ homogeneity, we explore a counter-DoS mechanism in which endpoints randomize their *minRTO* parameter in order to randomize their null frequencies. Here, we develop a simple, yet illustrative model of TCP throughput under such a scenario. In particular, we consider a counter-DoS



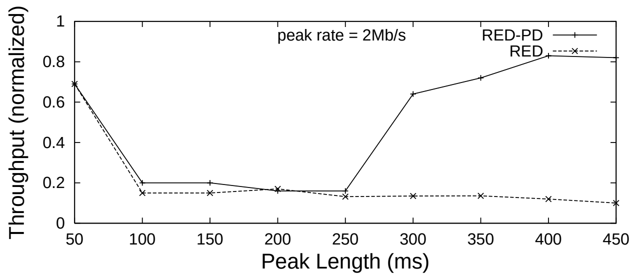


Figure 16: Detecting DoS streams

strategy in which TCP senders randomize their *minRTO* parameters according to a uniform distribution in the range $[a, b]$. Our objective is to compute the TCP frequency response for a single flow with a uniformly distributed *minRTO*. Moreover, some operating systems use a simple periodic timer interrupt of 500 ms to check for timed-out connections. This implies that while the TCP flows enter timeout at the same time, they recover uniformly over the $[1, 1.5]$ sec range. Thus, the following analysis applies equally to such scenarios.

We have three cases according to the value of T as compared to a and b. First, if $T \geq b$. Then $\rho(T) = \frac{T-E(RTO)}{T}$, where $E(RTO) = (a + b)/2$ so that

$$\rho(T) = \frac{T - \frac{a+b}{2}}{T}, \text{for } T \geq b. \tag{4}$$

Second, for $T \in [a, b)$, denote k as $\lfloor \frac{b}{T} \rfloor$. Then,

$$\rho(T) = \frac{T-a}{b-a}\frac{T - \frac{T+a}{2}}{T} + \sum_{i=1}^{k-1}\frac{T}{b-a}\frac{\frac{T}{2}}{(i+1)T} + \frac{b-kT}{b-a}\frac{(k+1)T - \frac{kT+b}{2}}{(k+1)T}. \tag{5}$$

Equation (5) is derived as follows. Since only one outage at a time can cause a TCP flow to enter retransmission timeout, we first determine the probability for each outage to cause a retransmission timeout and then multiply it by the corresponding conditional expectation for the TCP throughput. In Equation (5), the first term denotes TCP throughput in the scenario when the retransmission timeout is caused by the next outage after the initial one. The term $\frac{T-a}{b-a}$ denotes the probability that the initial RTO period has expired, which further means that the first outage after time a will cause another RTO. The conditional expectation for TCP throughput in this scenario is $\frac{T-\frac{T+a}{2}}{T}$, where $\frac{T+a}{2}$ denotes the expected value of the end of the initial RTO, given that it happened between a and T. The second term of Equation (5) denotes TCP throughput for outages $i = 2, \cdots, k-1$. The probability for them to occur is $\frac{T}{b-a}$, and the conditional expectation of TCP throughput is $\frac{T/2}{(i+1)T}$. Finally, the third term in Equation (5) denotes TCP throughput for the $(k+1)^{th}$ outage.

Finally, when $T < a$, it can be similarly shown that

$$\rho(T) = \frac{\lceil \frac{a}{T} \rceil T - \frac{a+b}{2}}{\lceil \frac{a}{T} \rceil T}, \text{for } k = 1, \tag{6}$$

and

$$\rho(T) = \frac{\lceil \frac{a}{T} \rceil T - a}{b-a}\frac{\lceil \frac{a}{T} \rceil T - \frac{a+\lceil \frac{a}{T} \rceil T}{2}}{\lceil \frac{a}{T} \rceil T} + \sum_{i=\lceil \frac{a}{T} \rceil}^{k-1}\frac{T}{b-a}\frac{\frac{T}{2}}{(i+1)T} + \frac{b-kT}{b-a}\frac{(k+1)T - \frac{kT+b}{2}}{(k+1)T}, \text{for } k \geq 2. \tag{7}$$

Figure 17 shows that the above model matches well with simulations for minRTO $=$ uniform$(1, 1.2)$. Observe that randomizing the *minRTO* parameter shifts both null time scales and amplitudes of TCP throughput on these timescales as a function of a and b. The longest most vulnerable timescale now becomes $T = b$. Thus, in order to minimize the TCP throughput, an attacker should wait for the retransmission timer to expire, and then create an outage. Otherwise, if the outage is performed prior to b, there is a probability that some flows' retransmission timers have not yet expired. In this scenario, those flows survive the outage and utilize the available bandwidth until they are throttled by the next outage.

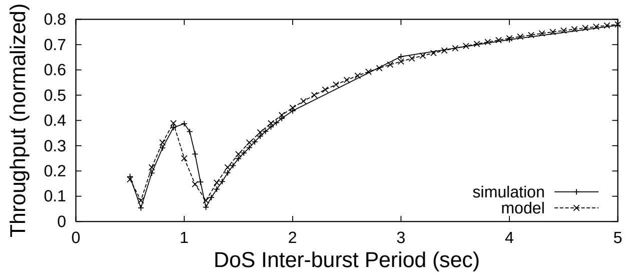


Figure 17: DoS under randomized RTO

Because an attacker's ideal period is $T = b$ under minRTO randomization, we present the following relationship between aggregate TCP throughput and the DoS timescale.

Counter-DoS Randomization Result. *Consider n long-lived TCP flows that experience b-periodic outages. The normalized* aggregate *throughput of the n flows is approximately*

$$\rho(T = b) = \frac{b - (a + \frac{b-a}{n+1})}{b} \tag{8}$$

The derivation is given in the appendix.

Equation (8) indicates that as the number of flows n increases, the normalized aggregate TCP throughput in the presence of $T = b$ timescale DoS attacks converges towards $\frac{b-a}{b}$. Indeed, consider the case that all flows experience an outage at the same reference time zero. When the number of flows in the system is high, a fraction of flows' retransmission timers will expire sufficiently near time a such that those flows can partially recover and utilize the available bandwidth in the period from time a to time b, when all flows will again experience an outage. For the scenario of operating systems that use a 500 ms periodic timeout interrupt, such that a flow "times out" uniformly in a [1,1.5] range, Equation (8) indicates that the TCP throughput degrades from 0.17 (single TCP flow) to 0.34 (TCP aggregate with many flows) under the 1.5 sec periodic attack.

There are two apparent strategies for increasing throughput on $T = b$ timescales. First, it appears attractive to decrease a which would significantly increase TCP throughput. However, recall that conservative timeout mechanisms are fundamentally required to achieve high performance during periods of heavy congestion [1]. Second, while increasing b also increases TCP throughput, it does so only in higher aggregation regimes (when n is sufficiently large) and in scenarios with long-lived TCP flows. On the other hand, increasing b is not a good option for low aggregation regimes (when n is small) since the TCP throughput can become too low since we have $\rho(T = b) = \frac{n}{n+1}\frac{b-a}{b}$. Moreover, excessively large b could significantly degrade the throughput of short-lived HTTP flows which form the majority traffic in today's Internet. In summary, minRTO randomization indeed shifts and smoothes TCP's null frequencies. However, as a consequence of RTT heterogeneity, the fundamental tradeoff between TCP performance and vulnerability to low-rate DoS attacks remains.

8. CONCLUSIONS

This paper presents denial of service attacks that are able to throttle TCP flows to a small fraction of their ideal rate while transmitting at sufficiently low average rate to elude detection. We showed that by exploiting TCP's retransmission timeout mechanism, TCP exhibits null frequencies when multiplexed with a maliciously chosen periodic DoS stream. We developed several DoS traffic patterns (including the minimum rate one) and through a combination of analytical modeling, an extensive set of simulations, and Internet experiments we showed that (1) low-rate DoS attacks are successful against both short- and long-lived TCP aggregates and thus represent a realistic threat to today's Internet; (2) in a heterogeneous-RTT environment, the success of the attack is weighted towards shorter-RTT flows; (3) low-rate periodic open-loop streams, even if not maliciously generated, can be very harmful to short-RTT TCP traffic if their period matches one of the null TCP frequencies; and (4) both network-router (RED-PD) and end-point-based mechanisms can only mitigate, but not eliminate the effectiveness of the attack.

The underlying vulnerability is not due to poor design of DoS detection or TCP timeout mechanisms, but rather to an inherent tradeoff induced by a mismatch of defense and attack timescales. Consequently, to completely defend the system in the presence of such attacks, one would necessarily have to significantly sacrifice system performance in their absence.

Acknowledgments

We thank Roger Karrer (Rice University) for help in obtaining accounts at ETH and EPFL, Luca Previtali and Matteo Corti (ETH's Lab of Software Technology) for providing the hardware and software for the Internet experiments, Peter Bircher and Armin Brunner (responsible for network security at ETH), for allowing the experiments, and Martin Vitterli (EPFL) for providing a computer account. Next, we thank Attila Pasztor and Darryl Veitch (University of Melbourne) for sharing their active probing software to perform the Internet experiments. Finally, we thank the anonymous SIGCOMM reviewers as well as our shepherd, Srinivisan Seshan (CMU), for providing helpful comments on this paper.

9. REFERENCES

[1] M. Allman and V. Paxson. On estimating end-to-end network path properties. In *Proceedings of ACM SIGCOMM '99*, Vancouver, British Columbia, September 1999.

[2] F. Anjum and L. Tassiulas. Fair bandwidth sharing among adaptive and non-adaptive flows in the Internet. In *Proceedings of IEEE INFOCOM '99*, New York, NY, March 1999.

[3] R. L. Carter and M. E. Crovella. Measuring bottleneck link speed in packet-switched networks. *Performence Evaluation*, 27(28):297–318, 1996.

[4] C. Dovrolis, P. Ramanathan, and D. Moore. What do packet dispersion techniques measure? In *Proceedings of IEEE INFOCOM '01*, Anchorage, Alaska, April 2001.

[5] F. Ertemalp, D. Chiriton, and A. Bechtolsheim. Using dynamic buffer limiting to protect against belligerent flows in high-speed networks. In *Proceedings of IEEE ICNP '01*, Riverside, CA, November 2001.

[6] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of ACM SIGCOMM '02*, Pittsburgh, PA, Aug. 2002.

[7] K. Fall and S. Floyd. Simulation-based comparison of Tahoe, Reno and SACK TCP. *ACM Computer Comm. Review*, 5(3):5–21, July 1996.

[8] A. Feldmann, A. C. Gilbert, P. Huang, and W. Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control. In *Proceedings of ACM SIGCOMM '99*, Vancouver, British Columbia, September 1999.

[9] W. Feng, D. Kandlur, D. Saha, and K. Shin. Stochastic fair BLUE: A queue management algorithm for enforcing fairness. In *Proceedings of IEEE INFOCOM '01*, Anchorage, Alaska, June 2001.

[10] S. Floyd and V. Jacobson. On traffic phase effects in packet-switched gateways. *Internetworking: Research and Experience*, 3(3):115–156, September 1992.

[11] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[12] S. Floyd and E. Kohler. Internet research needs better models. In *Proceedings of HOTNETS '02*, Princeton, New Jersey, October 2002.

[13] S. Floyd, J. Madhavi, M. Mathis, and M. Podolsky. An extension to the selective acknowledgement (SACK) option for TCP, July 2000. Internet RFC 2883.

[14] J. Hoe. Improving the start-up behavior of a congestion control scheme for TCP. In *Proceedings of ACM SIGCOMM '96*, Stanford University, CA, August 1996.

[15] V. Jacobson. Congestion avoidance and control. *ACM Computer Comm. Review*, 18(4):314–329, Aug. 1988.

[16] V. Jacobson. Pathchar: A tool to infer characteristics of Internet paths. *ftp://ftp.ee.lbl.gov/pathchar/*, Apr. 1997.

[17] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput. In *Proceedings of ACM SIGCOMM '02*, Pittsburgh, PA, Aug. 2002.

[18] H. Jiang and C. Dovrolis. Passive estimation of TCP round-trip times. *ACM Computer Comm. Review*, 32(3):5–21, July 2002.

[19] K. Lai and M. Baker. Measuring link bandwidths using a deterministic model of packet delay. In *Proceedings of ACM SIGCOMM '00*, Stockholm, Sweden, August 2000.

[20] D. Lin and R. Morris. Dynamics of Random Early Detection. In *Proceedings of ACM SIGCOMM '97*, Cannes, France, September 1997.

[21] J. Liu and M. Crovella. Using loss pairs to discover network properties. In *Proceedings of IEEE/ACM SIGCOMM Internet Measurement Workshop*, San Francisco, CA, Nov. 2001.

[22] R. Mahajan, S. Floyd, and D. Wetherall. Controlling high-bandwidth flows at the congested router. In *Proceedings of IEEE ICNP '01*, Riverside, CA, November 2001.

[23] T. J. Ott, T. V. Lakshman, and L. Wong. SRED: Stabilized RED. In *Proceedings of IEEE INFOCOM '99*, New York, NY, March 1999.

[24] R. Pain, B. Prabhakar, and K. Psounis. CHOKe, a stateless active queue management scheme for approximating fair bandwidth allocation. In *Proceedings of IEEE INFOCOM '00*, Tel Aviv, Israel, March 2000.

[25] A. Pasztor and D. Veitch. High precision active probing for Internet measurement. In *Proceedings of INET '01*, Stockholm, Sweden, 2001.

[26] A. Pasztor and D. Veitch. The packet size dependence of packet pair like methods. In *Proceedings of IWQoS '02*, Miami, FL, May 2002.

[27] V. Paxson. End-to-end Internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, June 1999.

[28] V. Paxson and M. Allman. Computing TCP's retransmission timer, November 2000. Internet RFC 2988.

[29] A. Rangarajan and A. Acharya. ERUF: Early regulation of unresponsive best-effort traffic. In *Proceedings of IEEE ICNP '99*, Toronto, CA, October 1999.

[30] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-based IP traceback. In *Proceedings of ACM SIGCOMM '01*, San Diego, CA, August 2001.

[31] L. Zhang, S. Shenker, and D. Clark. Observation on the dynamics of a congestion control algorithm: The effects of two-way traffic. In *Proceedings of ACM SIGCOMM'91*, Zurich, Switzerland, September 1991.

APPENDIX

Computing the throughput of a TCP aggregate on the $T = b$ time-scale.

Assume that an initial outage causes all TCP flows to enter the retransmission timeout and assume that $T = b$. Then, the throughput of the TCP aggregate can be computed as

$$\rho(T = b) = \frac{b - E(x)}{b}, \tag{9}$$

where $E(X)$ denotes expected value of a random variable X which corresponds to an event that at least one TCP flow's timeout expired at time x, $x \in [a, b]$. Assuming that each TCP flow's $minRTO$ is uniformly distributed between a and b, the CDF of X becomes

$$P(X \le x) = 1 - (\frac{b - x}{b - a})^n. \tag{10}$$

Denoting the corresponding pdf of random variable X as $p(x)$, we have

$$p(x) = \frac{\partial P(X \le x)}{\partial x} = n\frac{(b - x)^{n-1}}{(b - a)^n}. \tag{11}$$

The expected value of X, $E(X)$ can be computed as

$$E(X) = \int_a^b xn\frac{(b - x)^{n-1}}{(b - a)^n}\,dx. \tag{12}$$

The integral from Equation (12) can be solved by using integration by parts with the substitutes $n\frac{(b-x)^{n-1}}{(b-a)^n} = dv$ and $x = u$. The solution is $E(X) = a + \frac{b-a}{n+1}$. Thus, based on Equation (9), we have that Equation (8) holds.