Faculty of Electrical Engineering, Mathematics and Computer Science
Network Architectures and Services

A Study On TCP–SYN Attacks And Their Effects on A Network Infrastructure

Cliff Nyangasi Maregeli

June 23, 2010

A thesis submitted in partial fulfillment of
the requirements for the degree of
Master of Science

prof.dr.ir. P. Van Mieghem:	Chairman
dr.ir. F. Kuipers:	Member
dr.ir. E. Onur:	Member
dr.ir. T. Kleiberg:	Supervisor
ir. J. Omic:	Supervisor
Date of Defense:	July 8, 2010
Thesis no:	PVM-2010-063

A Study On TCP–SYN Attacks And Their Effects on A Network Infrastructure

Master's Thesis in Computer Engineering

Network Architectures and Services
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Cliff Nyangasi Maregeli

June 23, 2010

Dedication

In memory of my father, Frank G.S. Maregeli.

To my mother, Peris

To my wife, Flora and our daughter

Nora.

Abstract

Over the years, the Internet has evolved from a tool for the research community to an indispensable network connecting over a billion nodes world wide. There are many security threats existing on the Internet, one of them is the denial-of-service attack (DoS).

In this thesis, we study effect of denial-of-service attacks arising from TCP SYN flooding. SYN flooding attack has been widely observed world-wide, and occupies about 90% of the DoS attacks. We examine the effects of the attacks on individual host, and the underlying network infrastructure carrying the SYN flood packets. In laboratory, we deploy isolated network set-ups, to test the effects of the attacks on both the network and host. Finally, we design a queuing upper bound model to estimate the probability of connection loss on a host under a SYN flood attack. We compare the results from our upper bound model with results from selected models in the literature.

Acknowledgments

I am thankful to the Almighty God for His provision.

I am heartily thankful to my supervisors, Tom Kleiberg and Jasmina Omic, for their unlimited support and guidance from the start to the end of this project. Their insight and experience provided an invaluable contribution to the success of this project.

My sincere gratitude goes to Professor Piet Van Mieghem and the NAS group to which I had the opportunity to work on my graduation project.

Lastly, but not the least, I would like to extend my gratitude to the Computer Engineering (CE) class of 2008. It is a great pleasure and honor for me to share and cherish the memories of Delft with them, I wish them well, and hope to meet again in different walks of life.

Cliff N. Maregeli.

Delft, The Netherlands
June 23, 2010.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Internet has changed the way we work, live, and provides a big contribution to the growth of human socio-economic life. Undoubtedly the Internet is taunted as man's greatest invention of the 21^{st} Century along with television and other important inventions. Most of the Internet design was done more than thirty years [1] in the past. The Internet was then designed for research, envisioned by the Department of Defense (DoD) of the United States in the late sixties, for interconnecting mainframe computers; it was the ultimate tool that facilitated the exchange of data in the research community. One of the big weaknesses of the Internet design is the security. Internet was not designed with security awareness to meet the faced modern demands. Instead over the period of time security has been added in an ad-hoc fashion in form of patches to the Internet's protocols and applications.

Behind the scene, the Internet is running on top various protocols. TCP/IP is the most commonly used set of protocols, accounting for about 85% of the total Internet traffic volume [2][3]. The set provides transport to the majority of applications running on the Internet. TCP has a known design weakness that leaves it open to SYN-flood based attacks [13]. SYN-flood attack is one of several methods used by malicious Internet users to launch DoS and DDoS attacks. Other methods include SMURF-attack targeting ICMP service, and UDP-flood targeting UDP protocol [4][5].

In this work, we study the SYN flooding attacks. First, we study the effect of the flooding attack on an individual host, in an isolated network environment. We are interested in how factors, such as, the intensity of the attack, the design choices of the operating system, and the physical resources of the host, affect the host's ability to deliver TCP service during an attack. Thereafter, we look the effects of the flooding attack on the underlying network infrastructure delivering the flooding attack. In this latter study, we examine the behaviour of the deployed routing protocols and resource utilization of the network devices during the attack. Finally, we develop a mathematical model that estimates the blocking probability of

a TCP request in relation to the intensity of the flooding attack.
In the following sections we provide a brief description of the basics of the TCP/IP, routing, denial-of-service attacks (DoS and DDoS), which provide the literature review of the basic building blocks of our discussion.

1.1 Internet Protocol(IP)

The Internet protocol (IP) defines a packet delivery service that is

- delivery without guarantees nor acknowledgments,
- connection-less: each packet is treated independently from all others,
- best-effort: the Internet makes a fair attempt to deliver packet to the best of its capabilities [89].

The delivery between source and destination hosts in the Internet uses unique identification numbers, known as IP addresses. Every host connected in the Internet posses a unique IP address that is used to distinguish the host from all other hosts connected. The IP protocol specifies the basic unit of transfer, the IP datagram or IP packet.

In order to achieve its function, the IP layer uses the part of the datagram called header, to store control information. The datagram's header is divided into several fields as shown in Figure 1.1, of particular interest are the source and destination logical addresses. IP address is a global addressing scheme of the Internet. There are currently two versions of IP addressing schemes; IP version four (IPv4), the older version of 32-bits in length with a total number of 2^{32} possible address combinations, and the new generation, IP version 6 (IPv6) with 128 bits length, hence a larger address space (of about $3.4x10^{38}$), designed to supersede the rapid depleting IPv4. Every host on the Internet is identified by a unique IP address, this defines a unique connection, hence possible to route IP datagram from one host to another [1].
The datagram header of IPv4 also contains fields:

- Version: gives the version of IP used as mentioned on the last section.
- Header length: indicates the total length of the datagram.
- Service: differentiated services enhancements on IP as defined by IETF on RFC 2474 [46].
- Time To Live (TTL): limits the number of hops datagram can traverse across the Internet before they are discarded, this prevents undelivered datagram from bouncing around the network forever.
- Flags: this field is used in fragmentation across different heterogeneous networks, which often happens because of different frames used by different link layer protocols.

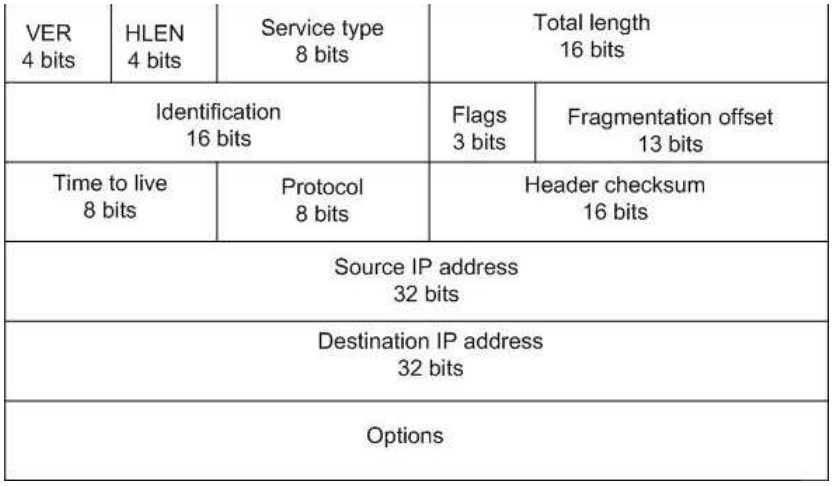

Figure 1.1: The format of an IP datagram's header.

- Total length: gives the total length of the IP datagram in bytes (data + header).
- Protocol: defines the higher level protocol that uses the services of IP layer.
- Checksum: a datum used for error calculation [1].

1.2 Routing

Routing is the process of selecting paths in a network along which to send network traffic. A router (intermediate-host) creates and maintains routing tables indexed by a network IP addresses to forward packets to the required destinations in the Internet. Routers are specialized nodes in the Internet, which are tasked with the routing function [1].

In the simplest form routing can be direct, meaning the source and sender are located on the same physical network. In this case a data-packet is forwarded to the destination network interface. Indirect routing is used when the source and destination host are not located on the same physical network. In this case the packet is forwarded to a router that knows how to reach the destination. The router determines the route using the destination address and index of routes maintained in the routing table [1].

In addition, depending on how a routing table is created and maintained within a router, routing is classified into dynamic and static routing. In static routing, the routing tables are created and maintained manually by the network administrator. This means that any changes that may occur in the network, for instance, a change of a default route, or an addition of a new network segment, must be manually reconfigured by the administrator.

This type of routing is sufficient for small networks that hardly go through frequent changes. The manual configuration is impossible for administration of a big network, like the Internet, which experience changes all the time [1][19].

On the other hand, dynamic routing deploys special routing protocols to create and maintain routing tables. The routing protocols facilitate the exchange of routing information between inter-connected routers. These protocols are grouped according to whether they are Interior Gateway Protocols (IGP) or Exterior Gateway protocols (EGP). IGPs exchange routing information within a single Autonomous System (AS). AS is a set of routers inside a single administrative domain. Open Shortest Path First (OSPF) and Routing Information Protocol(RIP) are examples of IGPs. EGP are used for inter-AS routing to make each AS aware of how to reach others throughout the Internet. BGP is the de facto EGP protocol [19][47].

1.2.1 Open Shortest Path First (OSPF)

OSPF is a link–state routing protocol designed to run within an autonomous system. Each OSPF router maintains an identical database describing the autonomous system's topology. A routing table is calculated using the database by constructing the shortest-path tree [9][19].

OSPF divides the routing domain into areas, starting with the area 0. Area 0 is the backbone, which is required on any OSPF configuration. This creates routing layers, and any inter-area routing must be routed via the backbone. Although, in some instances, this design causes inefficient routing, the layering helps to consolidate the addresses per area and reduce the size of the link-state database. In multi-access networks like the Ethernet, the OSPF uses a Designated-Router (DR), to manage all the link-state advertisements for the network. The router is selected through an election process during which a Backup Designated-Router (BDR) is also elected using the Hello protocol. The Designated-Router becomes adjacent to all other routers in the network [9].

The Hello protocol establishes and maintains a neighbor relationship in an OSPF network. Hello protocol packets are sent out periodically from all interfaces of an OSPF router. It is through these packets that OSPF routers in the neighborhood, learn each other, and their operational states. The Hello packets in broadcast networks contain the router's view of the Designated-Router's identity, and a list of routers to which Hello packets were received recently. If Hello packets are not received for a pre-configured period of time, the neighbor is marked as down. A Link State Advertisement (LSA) is generated to mark a link through a failed router as down. The aged out LSA's are eventually removed from OSPF routing tables after a pre-configured period of time. This ensures the routers forming the network are always update on the state of the network. Therefore, LSA's must be

frequently updated, before expiry time if the previous state still holds [9][19]. The Hello timers, managing the expiry, must be consistent across all routers in a network segment. OSPF is deployed in experiments in the sequel.

1.3 Intermediate System–Intermediate System (IS-IS)

The Intermediate System to Intermediate System (IS-IS) routing protocol is the de facto standard for large service providers' network backbones. The base specification of the protocol was published, for the first time, as ISO 10589 in 1987 and did not apply to IP packets at all. From then on, however, most of the work on the protocol has been done in the IS-IS working group of the Internet Engineering Task Force (IETF). The IETF group was responsible for the changes on the OSI[1] envisioned IS-IS. The group extended the protocol by defining additional Type-Length-Values (TLVs) carrying new functionality. Furthermore, the group clarified many operational aspects of IS-IS, for example, adjacency management that had not been exactly defined in the RFC 1195 [59], the first request for comment document to relate IS-IS to an IP environment [10].

IS-IS structures its network topology in a distinctive way. The idea is to structure a large network topology in smaller parts called areas, as we saw from OSPF. This keeps the topology horizon of the IS-IS routers small, therefore, keeping the CPU less busy during the route calculation process. The smaller networks resulting from the process are connected by special routers exchanging the traffic between the areas. Each router within an IS-IS network builds two level topologies: the Level-1 and Level-2 topology. A Level-1 router connects within the same area, while a Level-2 connects between areas of the network topology. IS-IS uses OSI addressing scheme.

As any other link state protocol, IS-IS includes a mechanism of self-discovery that determines any adjacent router running the same protocol. This process of creating adjacency involves: finding out if there are other routers speaking the same protocol, verification of a two-way capable communication and start of interaction with these remote devices to exchange network information. The exchange of Hello packets facilitates this process. The verification of link capabilities and bi-directional checks is done using a handshaking process. The reachability information of the newly installed router is then announced through flooding of Link State Protocol (LSP) packets. IS-IS also, maintains the state of the network by requiring each member to send a Hello at prescribed intervals. Network topology changes

[1] Open System Interconnection is a way of sub-dividing a System into smaller parts (called layers) from the point of view of communications. A layer is a collection of conceptually similar functions that provide services to the layer above it and receives services from the layer below it.

are exchanged through LSP's [10].

1.4 Transport Control Protocol (TCP)

TCP is a process to process (program to program) protocol. TCP provides the process to process communication using port numbers. TCP is a connection oriented protocol. This means for a process A to communicate with a process B, a connection has to be set up between A and B. This virtual connection set up across a network allows the sending process to deliver data as a stream of bytes, and the receiving process to receive data as a stream of bytes. TCP belongs to the transport-layer just above the IP protocol, in the network-layer, which provides a way for TCP to send and receive variable length data streams. TCP is a host-to-host protocol, intended to provide a reliable process to process communication within a multi network environment. A transport unit in TCP is called a segment.
TCP relies on the following features to achieve a reliable process to process communication:

- Numbering system: The segment header contains a sequence and acknowledgment number fields. The sequence number is assigned to every segment sent out by TCP, the number refers to the first byte of the segment in relation to a specific stream of bytes. The sequence number does not have to start from zero but is unique for a given connection direction. The acknowledgment field in a segment defines the number of the next byte the end connection expects to receive.

- Flow control: TCP provides flow control. The receiver controls the amount of data sent by the sender, to prevent the receiver from being flooded with data. Flow control is byte oriented. TCP uses sliding window mechanism to regulate the flow of data.

- Error control: TCP is able to detect corrupted, lost, out-of-order and duplicated segments. The mechanisms deployed for error control are acknowledgment, retransmission, and the checksums.

- Congestion control: The amount of data to be sent is not only determined by the receiver, but also on the level of congestion in the network. Congestion control is implemented by congestion window and congestion policy.

The segment consists of a 20 to 60 bytes header depending on the inclusion or omission of TCP options [84], followed by the application data. Figure 1.2 shows the format of a TCP header. In the sequel we provide a brief description of parts making the TCP header.

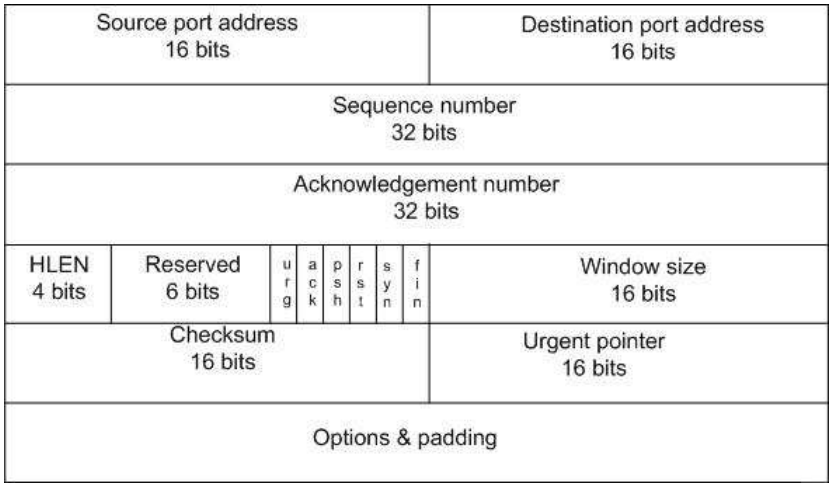

Figure 1.2: The format of a TCP segment's header.

- Source and destination port numbers: 16-bit field on the segment header that defines a particular source or destination application respectively on the interface between Application and Transport layer.

- Control: This field defines six control flags; URG– urgent pointer, ACK–acknowledgment field, PSH- push data, RST– reset connection, SYN–Synchronize sequence numbers during connection, FIN-terminate connection.

- Window size: defines the size of window in bytes that other party must maintain during transmission. The value as previously explained is determined by the receiver.

- Checksum: 16-bit field used for error checking calculations.

- Urgent Pointer: used to indicate urgent data

- Options: used to convey additional information to the destination or to align other options.

TCP protocol operation is divided into three parts. Connections are first established using a multi-step handshake (3 way-handshake). Successful connections proceed into the data transfer phase. After data transmission is completed, a connection termination phase starts that releases all the resources used by the connection [84].

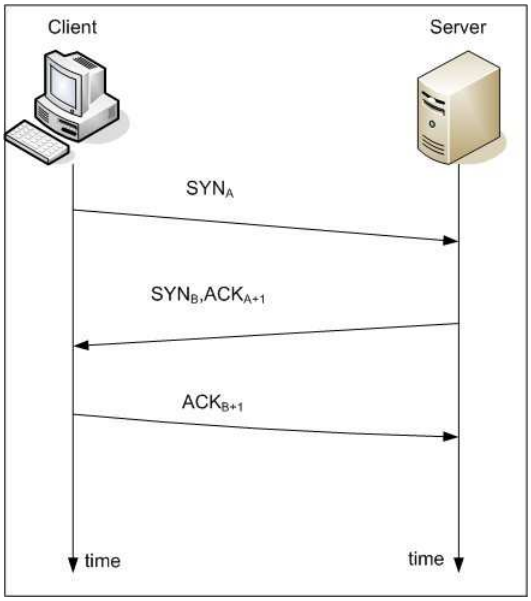

Figure 1.3: The 3-way-handshake protocol in TCP.

1.4.1 TCP Data Structures

Any TCP connection, that follows the Berkeley Software Distribution[2] (BSD) style code, requires remembering of several variables. The variables are conceived to be stored in a connection record called TCP Control Block (TCB) [84]. Among the necessary information stored regarding the TCP connection are: state information, protocols used, addressing information, connection queues, buffers and flags. The actual implementation of TCP records is done using the TCP data structures. Different BSD TCP implementations use different data structures. But at this juncture, it is sufficient to note that the whole process of establishing and maintaining a TCP connection, use a significant amount of memory resources from both the server and client[13].

1.4.2 A TCP Connection

TCP uses a 3-way handshake protocol to establish a logical connection. Figure 1.3, illustrates the TCP 3-way handshake protocol [1]. Before a client attempts to connect to a server, the server must first bind to a TCP port ready to accept connection. This is called a passive open state. The following section, explains the steps that a client goes through when requesting a connection to a server:

[2]Berkeley Software Distribution (BSD, sometimes called Berkeley Unix) is the UNIX operating system derivative developed and distributed by the Computer Systems Research Group (CSRG) of the University of California, Berkeley, from 1977 to 1995.

1. A client process issues a request for an active open by sending a TCP segment with a SYN flag set. It also sets the segment's sequence number to a random value A.

2. The server replies with a second segment, a SYN+ACK segment. The acknowledgment number is set to one more than the received sequence number (A+1), and for this segment the server selects a random number B, as its sequence number. This segment has two fold functions, first, it syncs communication from the server to the client, and secondly, informs the client that the previous SYN was successful received.

3. The client sends a third segment. This is a segment with ACK flag set directed to the server. The sequence number is set to the received acknowledgment value, and the acknowledgment number is set to one more than received sequence number (B+1). This segment informs the server that the previous SYN message was received successful.

At this point, both the client and sever have received an acknowledgment of the connection. The data transfer proceeds from this point.

1.4.3 TCP States

To keep track of all the different events from a connection establishment to termination, TCP implements a finite state machine. At any moment, the machine is in one of these states. The following is a brief description of all the states possible, Figure 1.4 shows the detailed state machine diagram.

CLOSED: There is no connection.

LISTEN: The server is waiting for calls from clients.

SYN-SENT: A connection request is sent; waiting for acknowledgment.

SYN-RCVD: A connection request is received.

ESTABLISHED: A connection is established.

FIN-WAIT1: The application has requested closing of a connection.

FIN-WAIT2: The other side has accepted the closing of a connection.

CLOSING: Waiting for the retransmitted segments to die.

CLOSE-WAIT: The server is waiting for the application to close.

LAST-ACK: The server is waiting for the last acknowledgment.

TIME-WAIT: Prevents delayed packets of data from being read by a later connection.

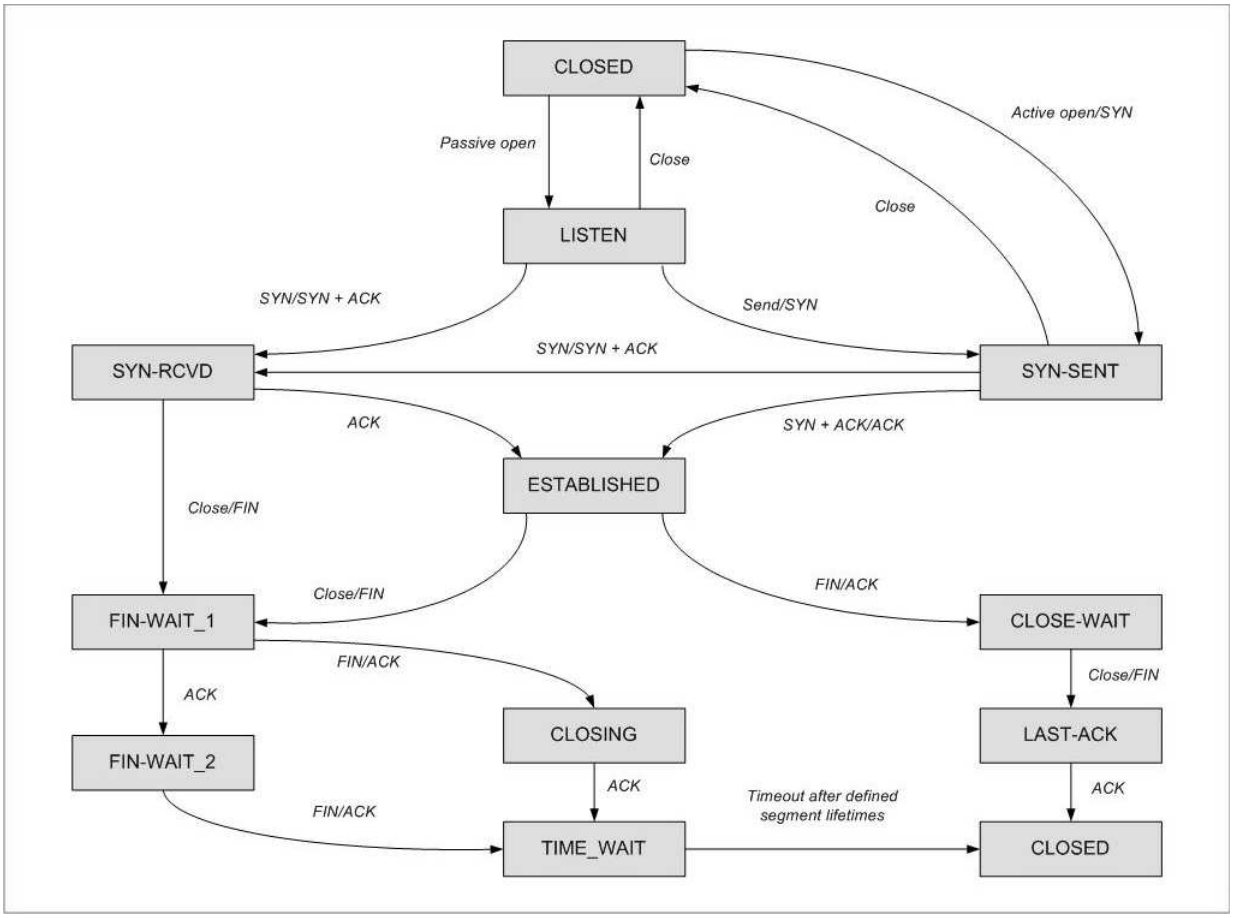

Figure 1.4: TCP State machine diagram.

1.4.4 SYN Flood Attack

The SYN flooding attack is a denial-of-service method on hosts that run TCP server processes. The attack leverages on TCP's state retention on establishing a new connection. As explained in the previous section, during connection establishment the client sends a SYN message to the server. After the server has received a SYN message to a port that is in the LISTEN state, the server creates necessary data structures to hold the information regarding the connection. The fundamental idea is to open many connections to the server until the server runs out of the memory resources for new connections.

TCP SYN flooding attack was first discovered early in 1994 by Ziegler et al [14]. However the first wave of attacks was first observed by September of 1996 [12]. The TCP implementation specification, may allow the LISTEN state to be entered with none, part or all of the IP address and port number being specified by the application. In fact, in most common applications, for example, in Web-servers the remote host information is unknown in advance. For it to be effective, the SYN flooding attack assumes the victim allocates a state for every SYN segment when it is received, and that there is a limit that exists on the number of states for different connections that can be kept at a time. If a client sends a SYN packet to request connection to the server with a spoofed (forged) [21][28] source address, a SYN+ACK reply from the server will be silently discarded by the network. It is important to emphasize that the spoofed IP address either belongs to a host that is unreachable or a host that is guaranteed to have an anomaly response to the SYN+ACK segment from the server. If a spoofed address belongs to a reachable and normal operating host in the network, the host will reply with RST segment to close the connection.

The server holds the half open connections, connections waiting for a ACK as response to SYN+ACK sent by the server, for a certain amount of time before giving up. If a victim has resources to admit N half-open connections, then a capacity to handle half-open connection requests can be modeled as a $(G/D/\infty/N)$ queue, where G is a general arrival process of SYN segments, D is deterministic life-time for each half-open connection that is not acknowledged by the respective client. This infinite-server queuing model with a finite capacity yields a minimal rate of SYN segments that will exhaust the server's allocated resources [45]. At this point, most operating-systems are designed to silently drop all new connections to the server, until resources can be reclaimed from timing out half-open connections or successful connections.

SYN flooding attack and its defenses have been widely researched, and continues to be an active research area [12][13][15][16][17][18]. A number of defense techniques are now available to the community. Below is a brief non-exhaustive list that highlights the defense methods.

Filtering: The ability of TCP SYN flooding being successful, depends on the ability to spoof source IP addresses. Removing attacker's ability to send spoofed IP addresses is an effective solution to the problem. Filtering described in detail [49], prevents the network from being a launch-pad for IP spoofed attacks. This measure requires ingress and egress filters [27] on all routers. For this solution to be effective all the networks will have to implement it. With a distributive nature of the Internet administration this is highly unlikely and infeasible to enforce. Moreover, the new concepts, for example, mobile IP [31] will make filtering deny access to legitimate network users.

Increasing Backlog: Increasing the backlog simply creates more resources for the server to accommodate more TCP requests in a half-open state. Depending on capacity of the server, the attacker could simply step up the packet rate that will be required for the server to run out of memory. In a case, that the attacker has a sufficient computing resources this could be quite useless as a constant packet rate increase will soon meet the server's ceiling capacity. Also it is proven that some implementations [12], like FreeBSD 4.4 do not scale well past back logs of few hundreds, without inefficiencies on data structures and search algorithms.

Reduce half-open connection timer: Another method to quickly mitigate the effects of SYN flooding attack is to reduce the half-open connection timer. This will make the operating system reclaim resources used by connections pending in half-open connections more rapidly. While reducing the timer will help the system reclaim resources used by bogus connections this could also discriminate the slow legitimate clients from establishing full connections. The tactic could also prove to be ineffective if the attacker can step up rate of TCP SYN flooding [12].

Recycling the half-open connections: Some TCP implementations enable new coming connections to overwrite older connections once the half-open queue is full. This scheme assumes that a legitimate connection will not spend a considerable time on the queue to be overwritten. This method will fail when the attack rate is high [12].

SYN Cache

The SYN cache method minimizes the amount of states allocates SYN segment, by not allocating the full TCB after receiving the SYN segment. The full state allocation is delayed until the full connection is successful established. The method uses a table indexed by the hash of the client's IP address and the requested TCP port in the SYN segment. Although SYN

cache allocates minimum state on the host, but even at this reduced rate it is possible to encounter resource exhaustion. The SYN cache implementation replaces the per-socket linear chain of incomplete queued connections with a global hash table, which provides two forms of protection against running out of resources. These are a limit on the total number of entries in the table, which provides an upper bound on the amount of memory that the SYN cache takes up, and a limit on the number of entries in a given hash bucket. The latter limit bounds the amount of time that the machine needs to spend searching for a matching, as well as limiting replacement of the cache entries to a subset of the entire cache. One of the major bottlenecks in earlier implementations that do not use SYN cache is the random drop implementation from the linear list, which did not scale. This is avoided in SYN cache, since the queue is split among hash buckets, which are then treated as FIFO queues instead of using random drop [6].

The hash value is computed on the incoming packet using source and destination addresses, the source and destination port, and a randomly selected secret hash. This value is then used as an index into the hash table, where SYN cache entries are kept on a linked list in each bucket. The hash secret prevents an attacker from targeting a given hash bucket by implementing denial-of-service attack, which targets on a specific bucket thus preventing a given set of clients from making a connection [6].

The drawback to this method is the inability to pass extra information regarding the TCP connection with the SYN segment. This additional information tunes the performance of a TCP connection. Although this does not have effect in TCP implementation, in some instance this may cause a glitch in performance [12].

SYN Cookies

SYN cookies unlike SYN cache do not store any state at all for the half-open connections. Instead, they encode most of the state needed into the sequence number to be sent back in a SYN+ACK segment. For legitimate SYN segments, the ACK reply is used to re-construct the state. This also means that the last ACK message from the client is the determinant factor of the connection set-up process.

The cookie is sent to the remote system as the system's Initial Sequence Number (ISN), and then returned in the final phase of the TCP's three way handshake. As connection establishment is performed by the returning ACK from the client, a secret is used to validate the connection, which is concealed from the remote system by using a non-invertible secret hash. Also, to prevent an intermediate system from collecting cookies and replaying them later, the hash also contains a time component [6].

The implementation of SYN cookie in Linux starts with a table of 32 bit value random numbers. The numbers are generated from a specially

designed random number generator. Each entry in the table is used for a duration of about 31.25 milliseconds, and has a total life time of 4 seconds, the latter is chosen as a reasonable upper bound for the Round Trip Time (RTT) to the client (remote) host, as the SYN+ACK segment containing the cookie must reach the host and be returned before the secret expires. To generate the cookie the system clock is scaled into units of 31.25 milliseconds by the use of shift operations, with the result used to choose the correct window index. If the secret of on the current window has expired, a new 32-bit secret is generated from the random number generator and timeout is reset [6].

The local address, foreign address, local port, foreign port and secret are passed through MD5 to create a cryptographic hash. The resulting hash is combined with the initial SYN from the remote host, and through a series of XOR operations a SYN cookie is finally generated, and is sent back as the Initial Sequence Number (ISN) from the server to the remote client. Since no state is kept on the server machine, any returning ACK which contains the correct TCP sequence number may serve to establish a connection. However the reverse of the process is done to validate any ACK segment received [6].

The problem with SYN cookies is the incompatibility with TCP options and also does not handle data piggybacked on the SYN segment as in SYN cache. In essence since the last ACK from the client establishes the connection, in case the segment is dropped there is no way to retrieve the connection as the neither the server nor the client will be aware. [12].

1.5 Denial of Service Attacks (DoS)

A Denial-of-Service (DoS) attack is an attack in which one or more hosts target a victim and attempt to prevent it from doing useful work. The victim in this case can be a network server, a router, a network link or an entire network, an individual network user or a company doing business using a network, an Internet Service Provider (ISP), country, or any combination of or variant of these [32][35].

There is a subtle difference between DoS and the so called flash crowd– a flash crowd is an incident when the victim experiences a heavy non-malicious traffic, which could potentially announce a similar effect to a denial-of-service attack [32]. The common DoS attacks, in general tend to exploits some aspects of the victim, the network or combination of them to achieve results. Below we discuss issues commonly exploited by the denial-of-service attacks.

Exploiting poor software quality– This exploits a bug in an application or operating system in the victim host that may cause the victim to misbehave or crush on attack. Examples of this form of attacks are buffer overflow and ping-of-death. Buffer over flow makes possible to

overwrite memory parts used by an application and cause the application to crush. Ping-of-death causes an erratic behavior to some of operating systems. When these systems receive a fragmented ICMP packet, with fragments totaled more than the 65535 bytes, which is allowed in an IPv4 packet, they behave unpredictably [32].

Application resource exhaustion– Applications running in network hosts require certain resources to operate properly. These resources like, memory, disk space, and CPU time are always finite. An attacker might prevent an application to work properly by exhausting the required resources by the application. Classic examples are the "zip bombs" that send series of emails with zip attached files that can fill up disk space of a mail server [60], and to impose computational intensive operations on the victim, for example, the Diffie-Hellman key exchange algorithm computation [32].

Operating system exhaustion– The idea here is quite similar to application resource exhaustion. A distinction is made because of the nature of the tasks carried by the operating system. The known attacks in this category are the TCP SYN flooding, already discussed in Section 1.4.4. Also SMURF attack that involves an attacker sending ICMP echo packets, with a source address spoofed to a victim's address, to a broadcast address. This makes all the hosts in the network segment including the victim to send echo reply messages to the victim. Also, UDP floods based on UDP echo and character generator services can produce a UDP flood when the two services are connected. This could potentially lead to a network overload [4].

DoS attacks on routers– IP routers are not immune to denial-of-service attacks, in fact, most of the attacks possible on end-host-devices, explained above can be launched against IP routers, for example, flooding the control ports, exhausting CPU and memory resources on the router. This may cause the router to cease routing packets or become excessively slow causing routing protocols to time out. In a different way from end-host-devices, attackers can launch attacks against the routing protocols running in IP routers. A common way accomplish this is to overload a router in a network by creating large routing table churns sufficient enough to make the router unable to process the changes. Moreover, attacker can propagate inconsistent routing information to the network that can cause traffic looping [32].

1.6 Distributed Denial of Service Attacks(DDoS)

A distributed-denial-of-service (DDoS) attack is a DoS attack with a multiple sources of the attack. The distributed format adds the "many to one"

dimension, by assigning one victim to multiple sources of attack [4]. DDoS attacks present a serious problem to a stability of the Internet [45]. An availability of many user friendly DDoS tools in the Internet does not provide any leverage on the quest to contain DDoS attacks.

In a DDoS attack, an attacker starts by gaining access to less secure (compromised) hosts over the Internet. Compromised host is any host on the Internet with insecure configurations, for example, trivial or no password, or known software bugs that can facilitate a break in. Once access is gained, the attacker proceeds to install "rootkits"[3]: main purpose of "rootkits" is to conceal any break in footprints and gain an administrative control over the compromised host [4][45].

Chang R in [45], classifies two types of DDoS attacks, direct attacks and reflector attacks. In a direct attack, the attacker commands to send a large number of attack packets directly to a victim. The packets can be of UDP, TCP, ICMP or a mixture of them. Reflector based DDoS attacks in [45] and [3], employs a different strategy in that the compromised machines send attack packets to some intermediary nodes, herein referred to as reflectors (reflector is an Internet host that replies to some request). The packets' source address is spoofed to the intended victim's address, and this causes the reflectors' replies to overwhelm the victim's resources. SMURF attack, as Section 1.5 explains, is a classic example a reflector attack. Otherwise, SYN requests from compromised hosts directed to several servers on the Internet will unavoidably lead to a flood of SYN-ACK replies to a victim. Although, a detection of authenticity of such packets is trivial by just observing the ACK field, depending on the size of the flood this can tie up a significant amount of resources to process the bogus replies. Contrary to Section 1.4.4 on SYN flood, this may be termed as SYN+ACK flood, because the victim is flooded with SYN+ACK segments.
A typical DDoS attack network consists of [52]:

1. A client computer that is operated by an attacker.

2. A number of handlers (also known as masters) which are controlled by the client.

3. A number of agents (also called zombies or daemons) controlled by the handlers and which perform the denial of service attack.

In general, DDoS utilizes tools to produce the denial-of-service attack from multiple sources. These tools are known in literature as DDoS tools or networks. Below is a non exhaustive survey of DDoS networks (tools) found in the literature.

[3]A rootkit is a software designed to gain administrator-level control over a computer system without being detected.

Trinoo (also Trin00) – Trinoo binaries were first found in Solaris 2.x systems. The Solaris systems were then known to have a Remote Procedure Call (RPC) bug [51]. The bug made the systems vulnerable to take over attacks. A zombie (taken over) host in a network is used to scan for more victims that can be broken into within the network block, and join the DDoS network. The owner of Trinoo uses TCP ports to communicate with the handler of the network, which in turn uses UDP port to communicate to the agents. The agents implement UDP flood attack against victims. The best defense against Trinoo is to prevent intrusion and root level compromise. Active monitoring can also be useful to isolate suspicious communications within the network [4][36].

Tribe Flood Network (TFN) – TFN binaries were also found in compromised Solaris hosts. TFN uses a command line execution of a client program for remote control. However, communication between handlers and agents is done via ICMP packets. This makes it difficult to be traced by some network monitoring tools. TFN can implement SYN flooding, UDP flooding and SMURF style attacks [37].

Stacheldraht – is an improved version of TFN. Stacheldraht, unlike the previous discussed tools, uses an encrypted communication between the handler and attacker. In addition, it deploys automated process to update agents. The Stacheldraht uses either TCP, UDP or ICMP for communication between handlers and agents. The tool can implement SMURF attack, TCP SYN, UDP flood and ICMP flood attacks [39].

Shaft – is a hybrid of Stacheldraht, TFN and Trinoo. Shaft network owner communicates to the handler via a TCP port, while all communications between handler and agents is through a randomly selected UDP port. A distinctive feature of Shaft is the ability to switch UDP ports in handler-agents communications to evade network scanners [40].

TFN2K – Binary files were found in Solaris and Windows NT hosts. TFN2K uses either TCP, UDP, or ICMP randomly for communication between handler and clients, also between handler and the network owner. The communication packets are interspersed with decoy packets to conceal their patterns. The network uses a key-based CAST-256 algorithm [53], in all communications. TFN2K implements TCP/SYN, UDP, ICMP, and SMURF packet flood [41].

1.7 Cascading Failures

The interconnection pattern of a network [54] is best modeled by a graph. Properties of a graph are often related to various characteristics that describe

the particular network that is modeled by the respective graph.

A graph, in this case representing a network, is made up of points [55], usually called nodes or vertices, and lines connecting the points, usually called edges. A graph is notated by [54] $G(N, L)$, where N - is the number of nodes and L - the number of edges. Also, a network interconnection pattern can be represented by a matrix, called adjacency matrix A, which in its simplest case is an nxn symmetric matrix, with n– denoting the number of nodes. The adjacency matrix has an element A_{ij} is set to one, if there exist a link between a node i and j, otherwise a zero is set, for all i not equal to j.

In real life, for example, common utilities and applications [42], like, the Internet, the WWW, road networks, and electrical power grids are complex networks in their own being, with a small distance between nodes and a highly organized distribution of link per node. In a complex network cascade [42][56] failures are initiated when a heavily loaded node-edge is lost causing the dynamics of the network to redistribute the load through the lost node-edge elsewhere in the network. This redistribution may cause other nodes (edges) to collapse as they struggle to keep the flow within their capacities. Hence, the number of failed nodes increases, propagating throughout the network. Cascading failures take place on the Internet, where traffic is rerouted to bypass malfunctioning routers that eventually leads to an avalanche of breakdowns to other routers due to heavy load. In man made network networks, the Internet included, the capacity of the network is largely limited by cost [43][54].

It is also found in literature [43] that for cascades triggered by the removal of a single node. If a node has a relatively small load, its removal will not cause major changes in the balance of loads, and subsequent overload failures are unlikely to occur. However, if a node has a relatively large load, its removal is likely to affect significantly loads at other nodes and possibly starts a sequence of cascade failures. It is then concluded that for a cascade failure to occur the following must hold; first, the network must exhibits a highly heterogeneous distribution of loads, and secondly, the removed node is among one with a high load. The distribution of loads is highly related to the distribution of links, in fact, a network with a heterogeneous distribution of links is expected to be heterogeneous with respect to load so that on average, nodes with larger number of links will have higher load [23].

1.8 Related Work

Denial-of-service is a problem in the Internet that has received a substantial research attention. In fact, a multitude and variety of both defense and attack approaches in the literature is [50] overwhelming. Most work in the literature has roughly covered the areas of taxonomy, simulations, pre-

valence, localization, tolerance and diagnosis. Localization, diagnosis and tolerance covers the general protection of a victim against denial-of-service attacks. On the other hand, taxonomy, simulations and prevalence entails on learning more about the patterns, tools and causes of denial-of-service attacks.

The design of TCP/IP protocol left it opens to a variety of security issues, in a classic paper [8], Bellovin et al, attempts to describe how these design issues can be used to launch denial-of-service attacks.

A number of solutions have been proposed [4][12][13][15][29][35] to provide tolerance to a system under a denial-of-service attack. In common, this class of solutions aims at controlling resource usage on victim side, ensuring that minimum work is spent on attacker's requests and the victim can survive for longer periods of time as possible after the onset of the attack. Also hardening the configuration of hosts and network elements, for example, applying required patches and disabling IP broadcast making sure hosts are not used as amplifiers. Solutions in this space are normally included within the operating systems, firewalls, switches and routers.

In localization, as opposed to tolerance discussed above, relates to the identification of attack patterns in the network. This approach is about the techniques and algorithms for automatically detecting the attacks as they [16] occur. The techniques in general, involve monitoring of the network links and analyzing the traffic patterns in the arriving and departing traffic looking for certain signatures of attacks [13][20][24][26][35][36]. This is usually a function of work Network Sniffer and Intrusion Detection Systems (IDS).

The diagnosis [12][18][4][25][27][35] focuses on identifying and isolating sources of an attack. The most widely deployed methods and techniques in this area are the ingress and egress filtering. These two techniques, which differ mainly in whether they are manually or automatically configured, cause routers to drop packets with source address that are not used in customer's network by the receiving router interface. Given the impracticality of ensuring all networks in the Internet deploy filters, some research has focused on development of tools and mechanisms for tracing flows of packets through the network independent of their ostensibly claimed source addresses.

Moor et al in [17], authors attempt to answer the question on how prevalent denial-of-service attacks exist in the Internet today. By using backscatter analysis, estimation was made on the world wide prevalence denial-of-service attacks. Mirkovic et al in [50], attempt to manage information generated form literature that discuss defenses and attacks in denial-of-service, and highlight a novel taxonomy for classifying attacks and defenses that provides to the research community a better understanding of the problem and the current solution space. Furthermore in Adan et al in [38], a simulation set up is proposed to study denial-of-service attack using a SYN flooding method in a wireless network. The set up elaborate the tools needed for the

experiment and how to deduce result of the attack. In addition, Lau et al [4] deploy a DDoS simulator to study performance of queuing algorithms in a router. Other denial-of-service attacks simulations are available in [24][34].

Cascading failure is a whole-some failure behavior in complex networks induced by overloads. Cascading failures are claimed to explain some failures in real world complex networks, for example, power grids, and the Internet. Crucitti et al and Motter et al in [42] and [43] respectively, examine how the redistribution of flows of quantities within the network, homogeneity and heterogeneity of the network and the degree of distribution of the nodes affects the likelihood of the cascade failures using analytical mathematical models.

1.9 Real Life Examples

There exist a number of reports on denial-of-service attacks and cascading failures observed in real life. In this section, we provide a non exhaustive survey from literature on denial-of-service attacks reported in the Internet, and cascade failures in both utility networks, for example, power grids, and the Internet.

In February 2000, a denial-of-service attack affected yahoo.com, cnn.com, ebay.com and other major commercial Web-sites. The attack brought Yahoo out of service for more than three hours [4]. It was a combination of a SMURF and UDP flood attack, and most likely used a TFN DDoS network. At its peak Yahoo was bombarded by traffic of more than a gigabytes per second of requests.

In April/May 2007, Estonian governmental and business servers were hit by a denial-of-service attack depriving service to hundred and thousands of legitimate users. The attacks were thought to emanate from Russia, opening yet another front on which two different countries can engage into an act of war. It is believed that a DDoS attack using a clandestine network was deployed [61]. In 2009 [44], the Australian government recruitment portal, SaaS, was also reported to be in a denial-of-service attack, the event prevented employers and the job seekers access to the portal. During the attack, the portal was slowed down to a point that it was unusable.

In another case, Comcast [22], a renowned Internet provider in the United States and Canada was reported to use a denial-of-service technique to regulate Peer-to-Peer(P2P) traffic. The traffic shaping techniques worked by sending a TCP packet with RST flag to all Peer-to-Peer seeds within the network after a certain threshold in bandwidth consumption was reached (mostly on peak hours), this closed the respective connections. The P2P service was denied to the customers without their knowledge to save bandwidth for the service provider. After it was found out, charges were investigated before the Federal Communication Commission (FCC).

The following events reports real life [43] examples of cascade failures, on August 10, 1996 in the western United States, a 1300MW electrical line in southern Oregon sagged in summer heat, initiating a chain reaction that cut power in eleven western states. Also it is suspected the same happened on 14 August, 2003 when an initial disturbance in Ohio triggered the largest blackout in the US's history that lasted for as long as fifteen hours. Cascading failures take place on the Internet [42]. In October 1986, during a first documented Internet congestion collapse, the speed of the connection between the Lawrence Berkeley Laboratory and the University of California at Berkeley, dropped by a factor of 100.

Chapter 2

SYN Flood On A Host

In this chapter, we describe a setup that creates a TCP SYN flooding attack in an isolated network environment. In the test network set up we create the SYN flooding attack, generated by one host in the network directed to another host connected in the same network segment. Our aim is to create an environment that provides us with an ability of controlling important factors (characteristics) of a SYN flooding attack relevant in our thesis work.

The test-bed is designed to study important features of a host in a SYN flooding attack, and denial-of-service induced by the SYN flooding attack. The services refer to the services offered by a host that is under TCP SYN attack that use the host's TCP stack. It is already known in literature, as Chapter 1 describes, that the SYN flood packet rate, state allocation strategy by the TCP process and some add-on features within the operating-system of the attacked host, dictate the impact of the denial-of-service resulting from a SYN flood.

2.1 Problem Statement

In the sequel we address important research questions that will ultimately provide the building blocks to more analysis required in the following sections. A brief of overview of the issues, which provide the impetus of the work in this chapter is briefly as follows.

- Investigate the design of a simple network but sufficient to perform a denial-of-service attack using TCP SYN flooding.
- Investigate software and hardware entities required in the test-bed to facilitate TCP SYN flooding.
- Investigate a convenient method to generate TCP SYN packets in a number that is sufficient to cause a denial-of-service attack on the designed test-bed.

- Investigate the rate of spoofed SYN packets required to bring a denial-of-service attack on a host with different configurations.
- Investigate how different settings affect (degrade or improve) the capability of an attacked host's operating-system to defend against TCP SYN flooding attack.
- Investigate how a host in a network can be said to experience a SYN flooding denial-of-service attack.
- Investigate the degradation of the services offered by the victim host during SYN flooding denial-of-service attack.
- Investigate the stress on network sub-system of the host i.e. the network interface cards and links, during the attack.

2.2 Experiment set-up

In this section, we present the detailed configuration of our experiment set-up. We first give the hardware configuration set-up, and thereafter we proceed into explaining the software configuration, which will discuss both, the operating-system and applications involved in our test-bed.

The applications form the necessary utility software, for example, packet generator and packet capture (sniffer), needed to generate and measure the TCP SYN attack. In operating-system section, we unveil our selected operating-system that will be used by the hosts in the test bed, and provide a specific implementation of the TCP stack in the selected operating-system. The TCP implementation is coupled with specific design choices that affect the SYN flooding attack.

2.2.1 Hardware configuration

In this part we give hardware details regarding the hosts and the network switch making up the hardware portion of the set-up. We deploy three hosts in our test-bed, the attacking host (host A): acts as a source of the TCP SYN flood packets, the victim host (host B): acts as a receiver of the TCP SYN packets, and the client host (host C): this host is connected on the same network segment as the host A and host B, it performs network sniffing, and access a selected TCP service from the victim host. The network switch provide the required network connectivity between all the hosts. The following summary covers the important aspects of the three hosts mentioned.
The attacking host

- Operating-System: Ubuntu Linux Desktop

- Processor: Intel Pentium 4
- Processor-Speed: 2600MHz
- System Type: X86 HP d530 Series Desktop
- Physical-Memory: 1024Mb DDR-SDRAM
- Network Card: Integrated Broadcom 10/100/1000

The victim host

- Operating-System: Ubuntu Linux Desktop
- Processor: Intel Pentium 4
- Processor-Speed: 2600MHz
- System Type: X86 HP d530 Series Desktop
- Physical-Memory: 1024Mb DDR-SDRAM
- Network Card: PCI Bus Connected 3COM 10/100

The client host

- Operating-system: Ubuntu Linux Desktop
- Processor: AMD Phenom X2 Dual Core
- Processor-Speed: 2800MHz
- System Type: X86 DELL DCSM
- Physical-Memory: 512MB DDR-SDRAM
- Network Card:Integrated Atmel 10/100/100

The network switch – A Dell Power Connect 3248 1U forty eight port switch.

- Switching Capacity: 13.6Gb/s
- 10/100BaseT Port: 48 Ports RJ 45
- Auto-Negotiation: Speed, Duplex, Flow Control
- VLAN - IEEE 802.1Q
- Availability - LACP, Spanning Tree, Port Mirroring and IEEE 802.3ad Link Aggregation

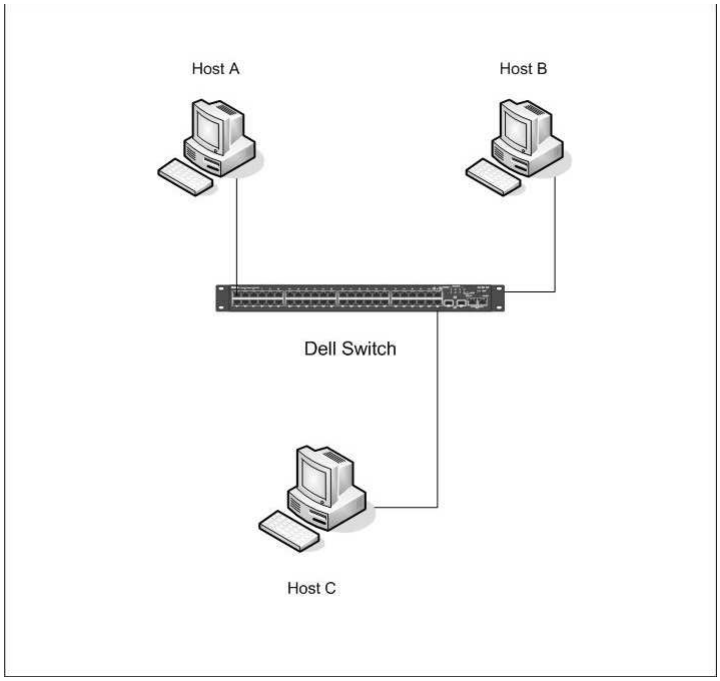

Figure 2.1: Experiment set-up: SYN attack on a host.

2.2.2 Network configuration

After hardware configuration of the three hosts, as stipulated on the last sub-section, we describe the network configuration of the three hosts. A standard switch is used to connect the three hosts, as the network diagram on Figure 2.1 shows.

Port Tap(Mirroring) is used in a switched network to send a copy of network packets seen on one switch port (or an entire VLAN) to a network monitoring connection on another switch port. This is commonly used for network appliances that carry out monitoring of network traffic, for instance, intrusion-detection system.

In this case we configure a Tap port on the Dell switch to duplicate all the traffic packets sent to the ports connecting host A and B. Host C is connected to the network segment through the Tap port. We then configure all the hosts with IP addresses from a selected network number, in order to obtain the IP connectivity in the network. A standard network connectivity test was performed on all hosts to ensure physical and logical network connectivity is present.

2.2.3 Operating–system configuration

We selected and install the Ubuntu Linux operating-system to all hosts in our test-bed. There are number of motivations for our operating-system selection. Firstly, Linux and its applications suite, are open source software, meaning they are governed by the General Public License (GNU GPL)[1]; Secondly, a large repository of additional software, which operates under Linux, and also under GNU GPL is available, which contains the necessary set of tools useful to our work; And thirdly, the source codes for Linux and its applications suite are also available, this is an important flexibility, it means we can make changes, on both applications and the operating-system, to meet our test-bed requirements.

The operating-system configuration refers to the settings, which are specific to the implementation of TCP stack in Ubuntu Linux. The TCP implementation of Linux is as defined on RFC documents [84][85] and [86]. For IP Version 4, used in our test-bed, specific TCP configuration parameters in the operating-system, can be viewed in specific files located in the directory `/proc/sys/net/ipv4` [83]. The TCP configuration parameters can be changed by adding appropriate lines in the config file `/etc/sysctl.conf`.

The TCP parameters `tcp_syncookies`, `tcp_max_syn_backlog`, `tcp_synack_retries`, and `tcp_abort_on_overflow` in Linux are directly involved in the mitigation of SYN flooding attacks [83].

Ubuntu is configured by default to send out syncookies when the SYN backlog queue of a TCP socket overflows. The `tcp_syncookies` feature attempts to protect a TCP socket from a SYN flooding attack. Section 1.4.4 explains in detail the operation of syncookies. However, syncookies are known not to have the ability to encode all the TCP options from initial SYN segment into the cookie. These options deprive the TCP stack the use of some TCP enhancements [6] and most importantly makes the TCP implementation not fully compliant with the TCP specification, i.e, retries on failures and the three-way handshake are violated. For this reason they are only activated once the SYN backlog queue is full. When the SYN backlog is full, the victim host may either be under attack or experiencing a heavy legitimate traffic load (a flash crowd).

The `tcp_max_syn_backlog`– is the maximum number of queued TCP connection requests, which have not received an acknowledgment from the connecting client. This default value is set to 1024 on the victim host in our test-bed. The number however varies from floor value of 128 to a ceiling value of 1024 depending on the physical memory available on a host. The backlog size provides an indication of how much resources are available to

[1]The GNU General Public License (GNU GPL or simply GPL) is the most widely used free software license, originally written by Richard Stallman for the GNU project. The GNU General Public License is a free, copy left license for software and other kinds of works.

handle half-open connections before they are moved into fully connected state. The bigger the backlog number the more difficult it is to overwhelm the system using the TCP SYN flooding.

The `tcp_synack_retries`– is the maximum number of times a SYN+ACK segment for a passive open TCP connection will be retransmitted by a server before giving up a particular connection. This is set to a default start value of 5 upon operating-system install in our hosts. The value means that TCP will spread out the retransmits of the segments in 180 seconds period before giving up a connection. A low value set on this parameter, the more it makes the system resilient to SYN flooding attacks, as half open connections will be quickly reclaimed, but will provide poor services to legitimate clients especially those from slow connections. The converse of this will make the system more vulnerable to the TCP SYN flooding in exchange for a good service. Ubuntu limits this value to a maximum of 255 [83]. This value should be chosen in such a way that it is able to accommodate a worse case RTT scenario for the clients serviced. In our test-bed scenario the client lies within the same network segment, hence the RTT for the worse case scenario is accommodated in all the tested value ranges.

The `tcp_abort_on_overflow`– enable resetting connections if the listening service is too slow and unable to keep up with the connection requests. This option is disabled by default of Linux. It means that if overflow occurred due to a burst, the connection will recover. This option must only be enabled when one is sure that the listening daemon can not be fine tuned to accept the connections quickly. Enabling this option has a warning of a possible harm [83] to the clients of a server. In our experiment we kept this option disabled as advised, thus our mentioning will be the only part of it in our work.

2.2.4 Software configuration

In this section we provide the description of software tools needed in our test-bed. Our most important tasks are creating the TCP SYN packets and investigating the conditions that will lead us to the answers of our main research questions, outlined in the problem statement section. The tasks mentioned provide us with the impetus for the selection of the tools as described below.

In line with our main tasks, generating SYN packets and performing measurements. We installed our packet generator (next section provides the details) on the attacking host and network sniffer for measurements on both the client host (host C) and the victim host (host B). Furthermore, a TCP server process (a Web-Server) was installed on the victim host. The server process will be flooded by the SYN packets during our experiment. The client host was installed with a TCP client, to provide access to the TCP server process.

Packet generator

For this task, we developed a C program script based on the BSD UNIX[2] socket application programming interface (API). The socket API is a library of rudimentary C functions that can be used to program communications, residing on different computer hosts, connected using a TCP/IP network.

A socket is one end of an inter-process communication channel. There are several types of sockets that can be implemented using the BSD API, for example, datagram, stream and raw sockets. Stream sockets offer reliable connection-based data transfer, the underlying protocol guarantees the data is read in the same order it is transmitted. TCP uses stream sockets. Datagram sockets deploy connection less transfer, with no guarantees on the delivery and the delivery order. UDP uses datagram sockets. Raw sockets allow access to raw network packets. Raw socket interface to IP or ICMP protocols [82]. The C program script we developed uses raw sockets.

The script crafts the TCP SYN packets and initiates a socket connection with a specified destination. Using the script, we can manually set a source and destination IP address of the packet, a destination TCP port, and a rate at which the SYN packets are generated and sent per unit of time. For the purpose of SYN flood attack the TCP source port, sequence number and IP identification are automatically set by a random generator. This ensures that a different TCP end point are generated with each SYN packet sent out, which provides for the requirement for generating a TCP SYN flooding attack.

Packet capturing

The packets are captured using tcpdump, which is a network packet capturer found in Linux and UNIX distributions. A packet capturer, like tcpdump, allows a user to capture and display network packets' details. Since a network connection may carry packets from more protocols than one's interests, network capturers provide a filtering mechanism using Boolean expressions. The captured packets can either be sent to the screen or stored in a file for more analysis later.

In our test-bed, tcpdump is used to ascertain that our packet generator (the C script) generates SYN packets, to collect statistics on the SYN packets processed by the victim and the attacking host, and to monitor TCP service requests sent by client host.

Server and client

To produce a TCP SYN flooding attack, we need an application that can make a TCP connection. File transfer, email, WWW, DNS and Telnet

[2]Berkeley Software Distribution (BSD UNIX) is a UNIX operating system derivative developed and distributed by the University of California, Berkeley.

are examples of applications that make use of the TCP stack for network connectivity. In our test-bed a TCP service is provided using a WWW application.

The Hyper Text Transfer Protocol (HTTP) is made up of a Web-server and Web-client. The Web-server is a program that delivers contents, such as Web-pages, using the HTTP, over the WWW. A web-client is a consumer of the content offered by a Web-server. In our test-bed we use Apache[3] as a Web-server and Mozilla as a Web client. Both Apache and Mozilla are open source software, available for download in the Ubuntu Software repository. The Web-server was installed in the victim host and the client on the client host in our test-bed.

In addition, we developed a C script, also using BSD socket which emulates Web-client's connection to the Web-server. The script simply perform a "connect" to the Web-server and disconnects when successful or after a timeout is reached, when not successful. For each successful/unsuccessful connection a count is kept. From the script we can set the number connection trials we need to make in a given experiment and the script will return both the count of successful and unsuccessful connections during that particular experiment. Moreover, the script takes the IP address and port number, which in our case is HTTP, of the host to connect to as its arguments.

The C script described above and the standard Mozilla Web-client are used as the client end of the HTTP service to be tested during the flooding experiment.

2.3 Detection of SYN flood on a Host

As far as hosts are concerned SYN flood can be detected by simply monitoring the TCP states, Section 1.4 describes TCP states in detail. Clearly this task is bound to be dependent on the operating-system deployed in the host in question. The `netstat` command in both Windows, Linux and Unix systems is used to display the status of network connections in the host.

In our test-bed environment we directly obtain the status of half open TCP connections in the host B by a combination of the netstat and some other utility commands in Linux. The half-open state of TCP in Linux is encoded as SYN_RECV state. We use `netstat` to obtain all network connections as described before, this output is fed to a `grep` command to filter all lines with SYN_RECV only, the half open connections needed, and lastly this output is piped to a word count program `wc` to print the number of all the half open connections in a system at that instant. Full command is as shown below.

```
#netstat -n -p -t | grep SYN_RECV | wc -l
```

[3]Apache is developed and maintained by an open community of developers under the umbrella of the Apache Software Foundation.

2.4 The Experiment

In this experiment we produce a TCP SYN flooding attack in our isolated test-bed network. We thereafter perform extended measurements to characterize in general a realistic number of packets required to produce such attack with different settings, that affect the SYN flood behaviour, on the operating system running on the host under attack. From the results we then deduce the maximum (ceiling) value of SYN packet rate needed to overwhelm a host with a TCP SYN attack.
The following details were supplied to our packet generator C script and tcpdump software

1. First we set the source and destination IP addresses; the source IP address was set to an arbitrary IP address number not reachable in our test-bed, and the destination IP address was set to the victim host's IP address.

2. In the TCP port section of the script, we set the destination port to 80, the HTTP server's listening port and the source port is automatically set by randomizer function in our script to an ephemeral port number ($portnumber > 1024$).

3. The script was then compiled using a C compiler, in our case the GCC compiler to produce the executable.

4. The Tap port was activated on the Dell switch to duplicate all the traffic from the ports connecting the host A and host B.

2.4.1 Generating SYN packets

After the explanation of the experiment prerequisites preparation, we proceed into the actual steps of generating, and making measurements of the TCP SYN flooding attack. The measurements are geared to capture the salient features of the attack, which enable us to characterize the TCP SYN flooding features in general. The following is an enumeration of the steps taken to conduct the attack during the experiment.

1. From a console terminal on the host C we run the tcpdump to capture all TCP packets in our network. Host C Ethernet port receives a copy of all the packets sent to Host A and Host B according to the Tap port configuration made earlier.

2. The C script was then provided with different rate for generating TCP SYN packets for each experiment trial, as shown on Tables 2.1–2.4. The packet rates were carefully selected on the basis of the resolution provided by our packet generator script, and a general figure of max

rate required to cripple a host found from literature of about 500 pps [87]. The script is launched from the attacking host.

3. Lastly, using the TCP client C script we perform trials of 20 consecutive connection-attempts for each packet rate generated by the script under specified operating-system settings. From the trials we measure the probability of making a successful connection and plot graphs of the Probability of Successful Connection vs SYN packet rate.

2.5 Results

In the set-up we vary three operating system configurations, as Section 2.2.3 describes; `tcp_syncookies`, `tcp_max_syn_backlog`, and `tcp_synack_retries`. The Tables 2.1, 2.2, 2.3, and 2.4 show the results of selected values of the experiments.

In the experiments we measure the probability of obtaining a connection to the HTTP port of a host under SYN flood attack. We set the SYN flood to a desired packet rate and after the attack stabilizes, stability is reached when the number of half-open connections in the victim host becomes 'fairly' constant, then we initialize 20 consecutive connections to the HTTP server. The probability of connection was obtained by the ratio of number of successful connections to the number of total connection attempts. The packet rates were selected by taking consideration of the resolution of our packet generator and a general ceiling of 500 pps[87], that is sufficient to cripple most practical systems with a TCP SYN flood attack.

The four tables of results represent four set measurements. In each table we keep the tcp_max_syn_backlog constant, this parameter is varied from 128, a minimum recommended value on current machines [6], to 1024 the standard value used by most modern Linux systems [6]. In addition for each table of results, the parameter tcp_synack_retries is varied with values 3, 5 and 7. The value of 5 is recommended, and it is by default, [83] set with a default Ubuntu Linux installation, the other values were selected to visibly demonstrate the effect of the parameter. In all cases the syncookies were completely turned off.

The graph on Figure 2.2, with a backlog of 128, the probability of making a successful connection drops with the number of retries set. This means that a graph with fewer number of connection retries, which is 3, is able to sustain an attack with more SYN packets rates as compared to the one with 5 and last with 7 retries. This is due to the fact that as more retries are sent out by the server the more time it takes for a connection to be removed from the half-open connections queue (recycled), hence the queues fills up quickly. In Linux implementation the SYN+ACK segment is transmitted twice the waiting time it took to transmit the previous SYN+ACK starting from 3 seconds after transmission of the original SYN+ACK. Of course, the

transmission stops after the number of retransmits reach the count determined the tcp_synack_retries setting. Graphs on Figures 2.3, 2.4, and 2.5 all show this behavior during the attack albeit with some different settings as pointed out earlier.

The last paragraph can make one think if it is possible to just make the retries as minimal as possible, in fact it is the solution offered by syncookies, no matter how appealing the solution seems, the catch is looking at how much is expected of a RTT time for a given client. RTT in this context is the time taken between the release of a SYN+ACK segment by the server and a receipt of an ACK segment from the client. This was not a problem in our test bed as both server and client reside on the same LAN segment and thus RTT was very low. In a real life scenario the RTT is bound to be big depending on both the distance and technology deployed between client and server and should determine the retries setting. It makes no sense for one to use a retry setting that will complete even before the RTT to a client as this will almost guarantee a poor service.

Looking at all the graphs we observe that the resiliency of the host against the SYN flooding attacks increase with an increased size of the backlog. The 1024 backlog show a very low probability of making a connection even when the attack is giving a packet rate of about more than 400pps, and the one with 128 the probability of connection is zero with a packet rate of just 50pps. This phenomenon supports the theory of the number of half-open connections that can be afforded by the host at different settings of the backlog. The more resources that are made available to support the half-open connections the bigger the SYN packet rate that will be required to overwhelm the host in a SYN flood attack.

In all the different set-ups described we also monitored the resource utilization on both the attacking host and the victim host (i.e. host A and B). The CPU utilization was not significantly high in both the victim host and attacking host. The attacking host had a maximum value of around 10 percent and a minimum of around 8 percent. The CPU utilization of the victim host was 10 percent on a high and 5 percent on a low. Memory wise the attacking host had a utilization of 30 percent on a high and a low of 20 percent, while the victim host had a high of 17 percent and a low of 15 percent. In all the experiments it was observed that running tcpdump in either host had brought a significant change to the the two resource utilization. From the bandwidth point of view as expected the bandwidth varied as we varied the packet rate of the attack. All the readings were taken off the resource monitor of the respective host during the experiments. This also shows that although the victim host was not able to service on the TCP port, it was able to perform some processing other than TCP. This is important as the literature also claims that the attacked host may crush as one of the effects of SYN flood attack. In many ways the latter behavior might be attributed to the TCP stack implementation within the operating system, which is not

the case with Ubuntu Linux used in our experiment.

Table 2.1: A table showing results for probability of connecting to a server with a backlog of 128

Packet rate(pps)	Probability of making a successful connection		
	retries = 3	retries = 5	retries = 7
0	1.000	1.000	1.000
5	1.000	0.400	0.100
10	0.300	0.200	0.050
20	0.200	0.100	0.000
50	0.000	0.000	0.000
100	0.000	0.000	0.000
200	0.000	0.000	0.000
400	0.000	0.000	0.000
500	0.000	0.000	0.000

Table 2.2: A table showing results for probability of connecting to a server with a backlog of 256

Packet rate(pps)	Probability of making a successful connection		
	retries = 3	retries = 5	retries = 7
0	1.000	1.000	1.000
5	1.000	1.000	1.000
10	1.000	0.950	0.950
20	1.000	0.900	0.500
50	0.900	0.650	0.400
100	0.500	0.250	0.050
200	0.050	0.000	0.000
400	0.000	0.000	0.000
500	0.000	0.000	0.000

Real life systems

In production Web-servers, for examples, online shops, banks and auctions. A clustering technology is mostly deployed [88], Figure 2.6 is a diagram of such a deployment, in the name of scalability, availability and capacity. In a network of cluster one entity sits on the network and acts as a proxy to the incoming connection requests. This entity is called a dispatcher. The back end of cluster is a series of servers connected by the dispatcher to the external networks. For reasons of redundancy two or more dispatchers are usually connected to the back end servers. The dispatcher handles requests from clients, from the Internet, and distributes those requests to the back

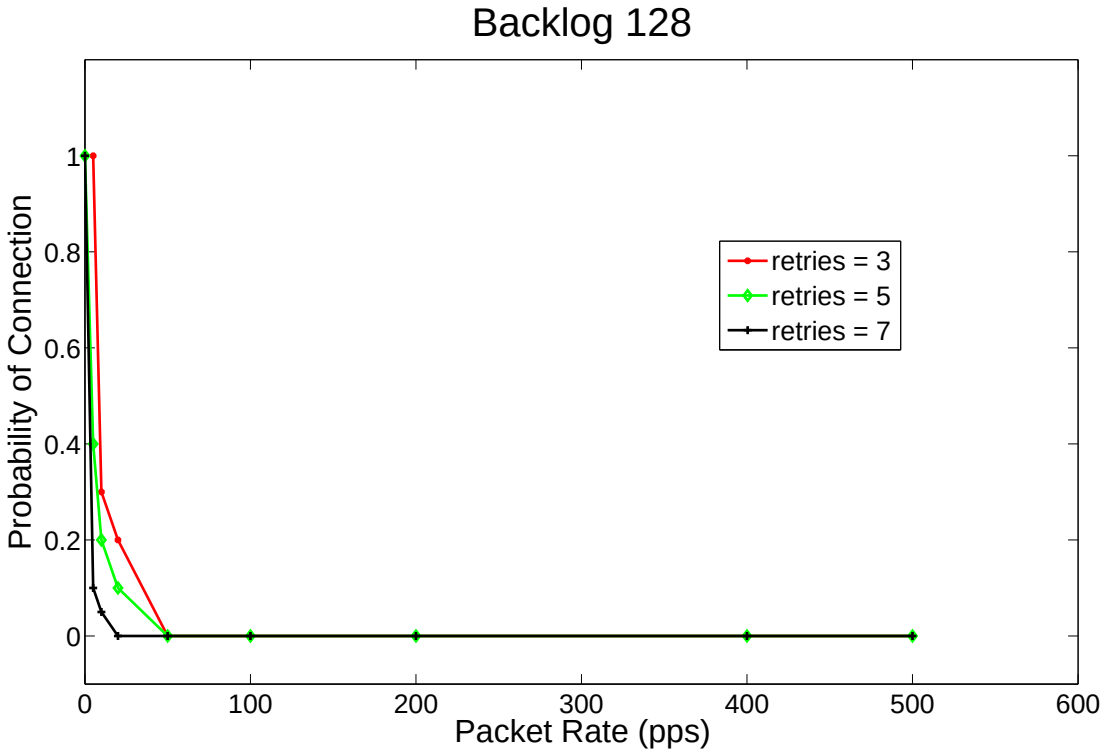

Figure 2.2: Probability that a host can connect to a server under attack with a backlog of 128.

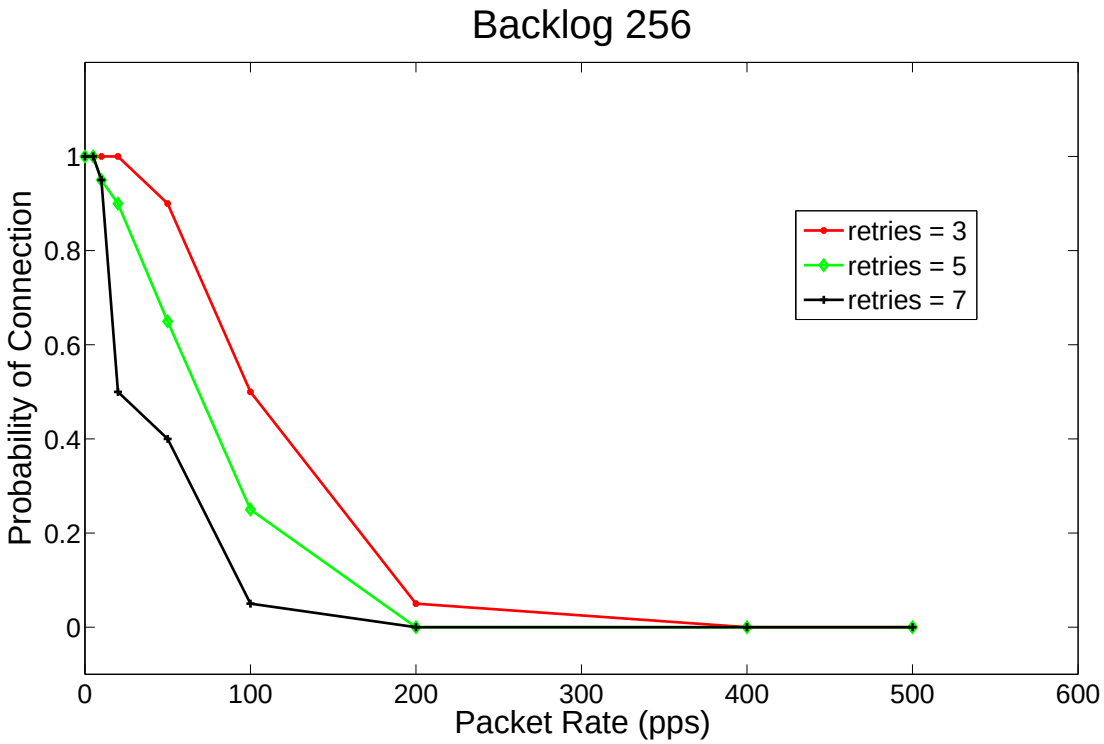

Figure 2.3: Probability that a host can connect to a server under attack with a backlog of 256.

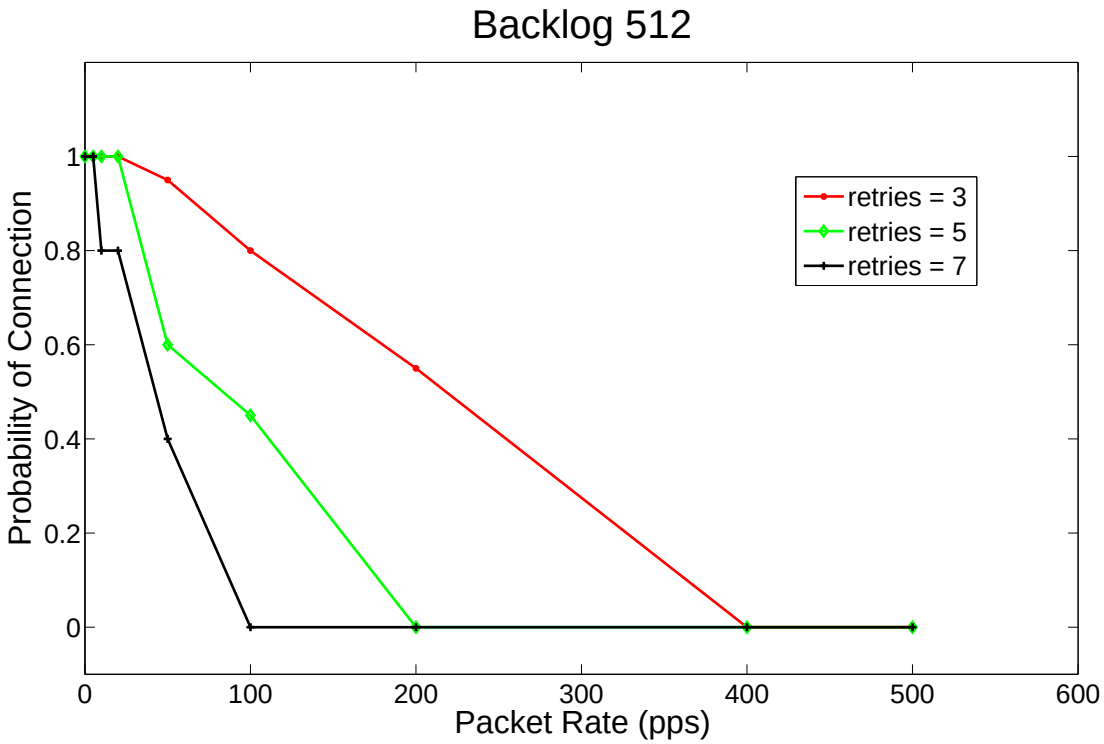

Figure 2.4: Probability that a host can connect to a server under attack with a backlog of 512.

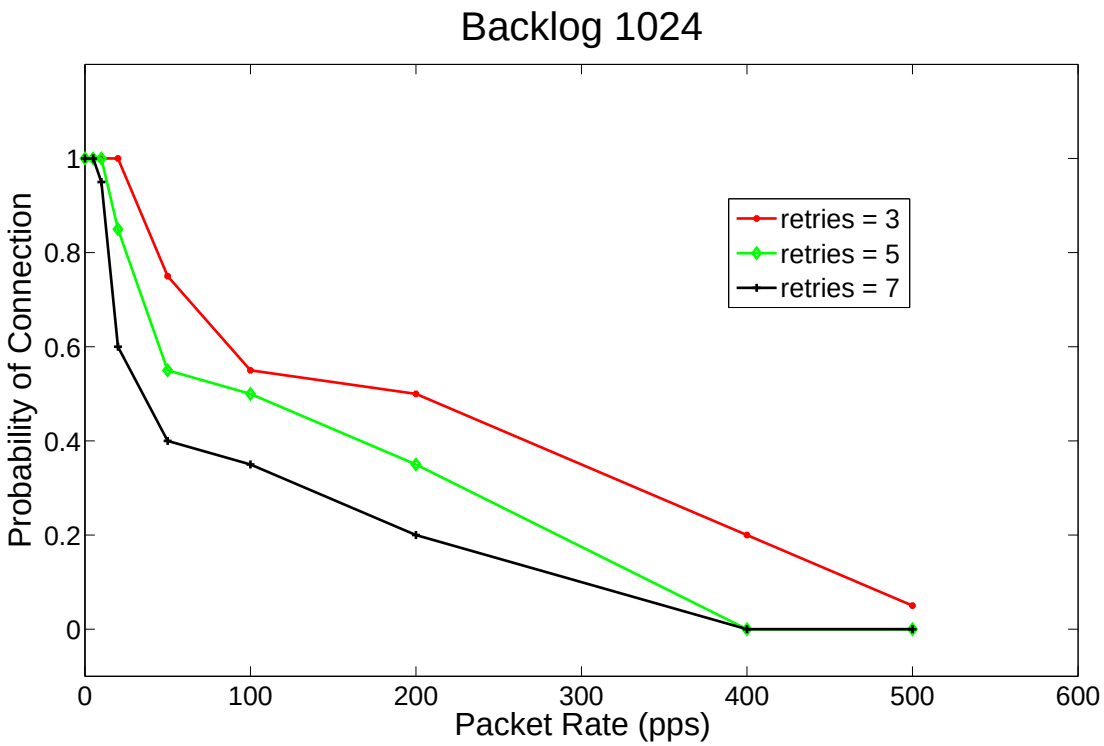

Figure 2.5: Probability that a host can connect to a server under attack with a backlog of 1024.

Table 2.3: A table showing results for probability of connecting to a server with a backlog of 512

Packet rate(pps)	Probability of making a successful connection		
	retries = 3	retries = 5	retries = 7
0	1.000	1.000	1.000
5	1.000	1.000	1.000
10	1.000	1.000	0.800
20	1.000	1.000	0.800
50	0.950	0.600	0.400
100	0.800	0.450	0.000
200	0.550	0.000	0.000
400	0.000	0.000	0.000
500	0.000	0.000	0.000

Table 2.4: A table showing results for probability of connecting to a server with a backlog of 1024

Packet rate(pps)	Probability of making a successful connection		
	retries = 3	retries = 5	retries = 7
0	1.000	1.000	1.000
5	1.000	1.000	1.000
10	1.000	1.000	0.950
20	1.000	0.850	0.600
50	0.700	0.550	0.400
100	0.550	0.500	0.350
200	0.500	0.350	0.200
400	0.200	0.000	0.000
500	0.050	0.000	0.000

end, according to some load balancing algorithm. The cluster is normally transparent to the clients and the Web-server software running on it.

In order to overcome a clustering system with a SYN flood attack one needs to take out the dispatcher (the front end), since they are the one responsible for receiving and distribution of client request. Furthermore if we take a scenario of the dispatchers running Linux as our test bed with retries set to 5, backlog of 1024, and without the syncookies. Then we need rate of about 1000 pps to overwhelm the front of the imaginary website. This amount of packet rate can be delivered from a single standard host available in the market. Worse still it only takes a bot network of 100 bots producing about 10pps each in a combined distributed denial-of-service attack. Although our example is based on HTTP but this can be extended to any application using TCP service, for example, FTP, Mail server or

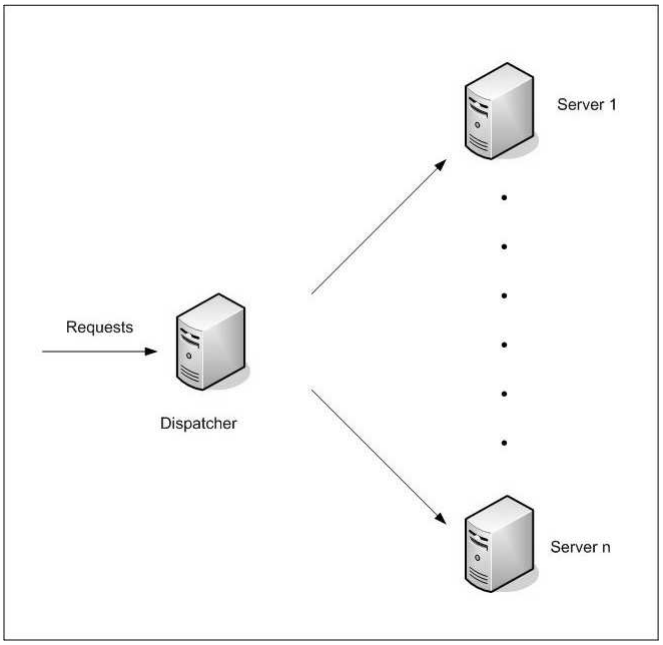

Figure 2.6: A high-level view of a basic server cluster showing the dispatcher and n servers in the pool.

Telnet.

2.6 Conclusion

In this chapter we successful provided a simple experiment to produce a TCP SYN flooding denial-of-service attack. Moreover, from the experiment we have established how different settings within the operating-system affect the practicality of the attack. In addition we also have a quantitative estimate on the packet rates required for a successful denial-of-service attack on different operating system configurations.

Another important research questions answered in this chapter are, the network architecture, the necessary tools and convenient method of generating enough SYN packets sufficient to produce a TCP SYN flooding denial-of-service attack. The results obtained in this chapter will prove useful from time to time in our work following in the next chapters.

Chapter 3

SYN Flood On A Network

Apart from the effects on end-hosts, discussed in Chapter 2, SYN flooding attack can have a detrimental effect on a network infrastructure carrying the flood packets. In this context we distinguish between end-hosts, which are the sources and sinks of the applications' data on the Internet, and intermediate-nodes (routers), which provide the control on how packets are routed to their desired destinations.

In this chapter, we investigate the ability of a SYN flooding attack to degrade or stop a computer network from dispensing an expected service. We design an isolated reference (test-bed) network for conducting a SYN flooding experiment. The flooding is done by one end-host, connected to one of the sub-networks of the reference network, as a source of the attack directed to another end-host located in another sub-network of the reference network. During the attack traffic traverses a connection of routers connected in our reference network.

We collect important details on the network performance during the attack. The network performance measurements are designed to provide us with an insight of the network degradation in relation to the strength of the SYN attack. In the sequel, we also investigate the response of common IGP protocols to SYN attacks, aiming to find out how the inherent designs of the protocols either amplify or limit the effect of the attack.

3.1 Problem Statement

The work in this chapter concentrates more on the effect of SYN flooding attack on the network infrastructure that carries the attack. The following research questions motivates the work presented.

- The design of an appropriate test-bed sufficient to conduct a SYN flooding experiment that can characterize the network performance.

- Identify the set of software tools required to extract and analyze the results.
- The relationship between the attack rate and network performance indicators.
- The resource utilization on intermediate hosts during the attack, for example CPU cycles.
- The effect of the attack on the dynamic routing protocol(s) deployed.
- A way to determine if a router in a network under attack is overloaded.
- The effect of the attack on the routing tables.

3.2 Measurements

When considering the performance characteristics of a computer network, several questions come into the picture: how do we know if the network is working/not working correctly? What are the main metrics that describe the network health in total? Is it possible to achieve network health measurement via a single metric, or this can only be done via an inference from different metrics? From the questions poised above the performance of a network is dependent to the end-user application; different applications in general do have a different network requirements. In our experiment we identified delay, connectivity, and packet loss as the network performance metrics, which guide us into experiment measurements. In addition, we identified CPU, memory utilization and status of the routing protocols on routers, also indirectly influencing the network performance.

In this chapter, we design and deploy three network scenarios. On each scenario we measure the metrics identified in the previous text. We refer the three scenarios as Scenario A, B and C. In Scenario A the reference network is made up of two Cisco routers. The routers run the OSPF dynamic routing protocol to exchange network routes between them. One of the routers is attached to the source of the SYN attack, and the other to the victim (sink) of the SYN attack.

In Scenario B, the reference network is made up of two Cisco routers. Scenario B deploys IS-IS, a dynamic routing protocol to route the networks connected by the two routers. In a similar fashion as in Scenario A, the source of SYN attack is connected on one of the routers, and the sink of the attack to the other router.

In Scenario C, the third scenario, we have a reference network with two routers, a Cisco router and a Quagga router. This time around the Quagga router is connecting the source of the SYN attack, and the Cisco router connects the victim (sink) of the SYN attack. This scenario deploys OSPF

routing protocol for exchange of network information between the respective routers. The Quagga router is preferred for connection to the attacking network to the Cisco router because of performance reasons that are going to become apparent in the following sections.

The network delay can be estimated using various tools, of varying degree of sophistication. One of a simple method of estimating the delay is through the round trip time (RTT) of the ICMP echo messages. The echo messages can be emanated from one host to another host traversing the network segment of interest. The average delay is then estimated by the average of round trip times for the given number of echo packets transmitted and received. We deployed this methodology in observing the delay in our scenarios. Network connectivity is simply the existence of a communication (physical and logical) path between two different hosts. This can be deduced from successful ICMP echo messages, and also in our case the SYN packets, received by the victim host.

Measurements of resources on routers can be done via collection of log data, as per vendor's specifications, or deployment of third party monitoring tools. In all cases an element within the router, to which the data is sought, works to provide the required data. This is a problem for a host that is under an attack at a considerable magnitude, as its resources ought to be tied up by the attack to hinder data collection along with some other critical functions. In our scenario we use Cacti a third party SNMP monitoring tool provide data on CPU, memory and bandwidth utilization as pointed out in relevant sections. Data estimation from the monitoring tools is sufficient for the purpose of our experiment.

One of impetus behind our experiment is to determine the behavior of deployed dynamic routing protocol during a SYN attack. This problem makes it paramount for us to be able to determine the status of running protocol in each scenario during our experiments. All scenarios deploy two different routing protocols, OSPF and IS-IS, the two protocols are the most deployed interior gateway protocol (IGP), hence the merit for our selection. At a reasonable high rate of the SYN attack we expect a high utilization of the router processors. This in turn may cause route flaps [68]; Route flaps refer to routing table changes in a router, usually in respect to network failure or recovery. We expect route flaps to occur in similar manner on both routing protocols but with specifics emanating from different state diagrams transited by them.

Figure 3.1 shows a state transition diagram of OSPF process, states Loopback and Point-to-point are omitted as they don't participate in our scenarios. Chapter 1 gives a detailed operation of the OSPF protocol. At an operation state, before an attack, the two routers in Scenario A and B exist as either designated router (DR) or backup designated router (BDR) from the election process during start-up, which establishes the adjacency among them. The routers use Hello protocol timers to make sure each side is up

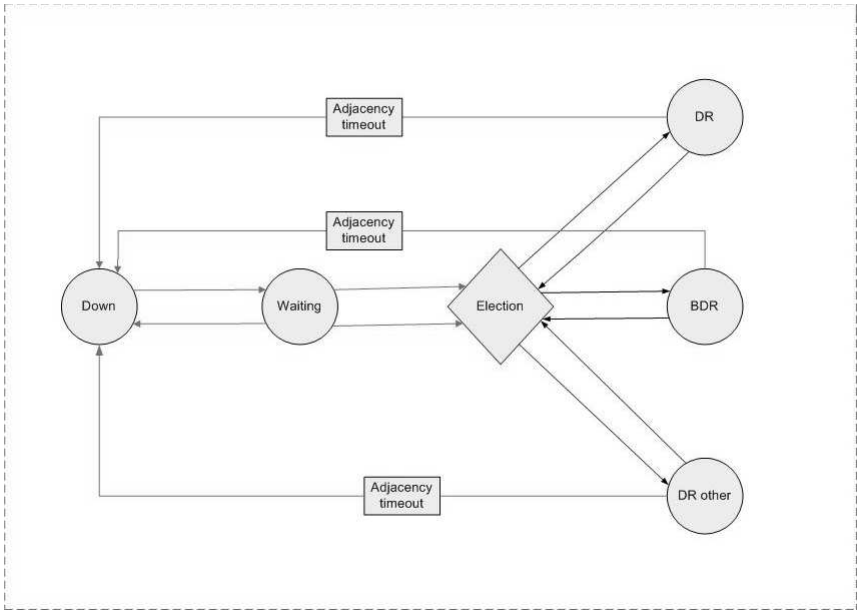

Figure 3.1: OSPF interface state machine

in addition to the exchange of network information. In an event one of the router fail to hear from the neighbor in a given time interval (Dead timer threshold), the relationship is broken down and any subsequent communication starts from the Down-state, re-establishing the OSPF. In this case all routes learned through OSPF are removed from the routing table.

With IS-IS, a router exists as DR - router or other states under normal operations. The DR router is either configured by the network administrator, or elected by an automated process during network start-up. Figure 3.2 show interface state machine diagram for IS-IS, some of the details (events) of the state machine not applying in scenario B are omitted for clarity. After adjacency is established in either DR or DR-other in UP state, IS-IS speaking routers maintain the health of each other through Hello packets, in addition to the routing information, after every predefined time intervals. Any violation of the timing by any of the router leads into break down of IS-IS adjacency and subsequent connection starts from Down-state to re-establish the adjacency. Furthermore, at this point all the routes on the tables are removed. Chapter 1 gives a detailed description of the IS-IS protocol. Thus, frequent changes of state of the routing protocols, caused by the violation of timers, in OSPF and IS-IS, are a source to the "route flap" phenomenon mentioned earlier.

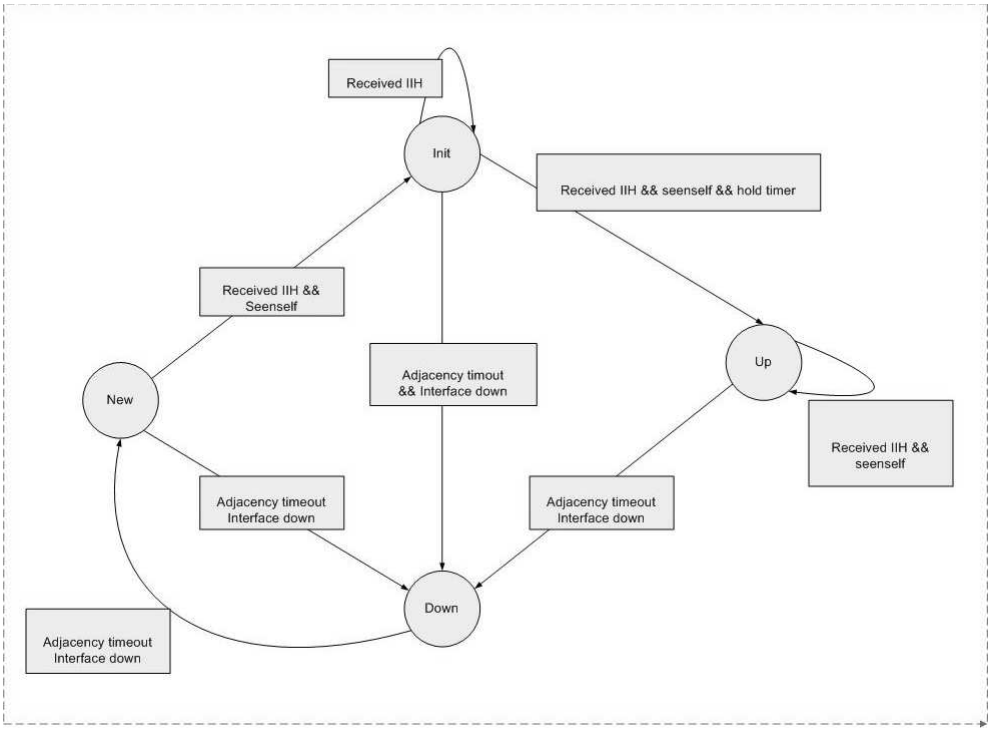

Figure 3.2: IS-IS interface state machine

3.3 Experiment set-up

In this section, we describe the detailed connection of our three identified scenarios, and the description behind hardware and software components used. The introductory notes highlight the technology and rationale behind the selection of the components in achieving the desired results. Finally, we provide details of how the experiments were conducted and the collection of data in all scenarios.

3.3.1 Hardware Configuration

In terms of hardware, the scenarios deploy host-machines, routers, and a network switch. There are five hosts connected during each scenario. All the hosts use Ubuntu Linux operating-system. The detailed configuration details are found in Chapter 2. The host-machines are marked alphabetically, from host -A to host- E. The different hosts play different roles, each explained in a specific scenario. The network switch is a Dell Power Connect also described in Chapter 2.

Cisco routers

In this chapter we add routing in the design of our reference networks. We have two identical Cisco 2621XM model routers. The two routers are provided with two Fast-Ethernet (100Mbps) and four Ethernet (10Mbps) interfaces each. They both run on IOS[1] version 12.3(26). Furthermore, each router is equipped with a Motorola MPC860P 50Mhz processor, a system flash memory of 32MB and 32K bytes of non-volatile configuration memory. The routers are identified by the host-names Rio and Miami.

Quagga host

Quagga software, described with a list of other software deployed, was installed on a multi-core HP desktop. The Desktop has a dual core with each core having a speed of 2600MHz and a RAM memory of 1024MB. Furthermore it is installed with three Ethernet cards, two of them with 100Mbps and one with 1000Mbps.

3.3.2 Software Configuration

In this section we provide a description to the software tools deployed in the experiment scenarios. The tools are selected to meet the observations and measurements needed to achieve our results. The main objectives of the current chapter are three fold; The first objective is to create SYN attack packets at different controlled rates, for this function we used the same C script developed in Chapter 2. The second objective is monitoring of some selected network metrics of interest as they will become apparent in the subsequent sections of this work. The last objective is the provision of routing functionality in our test-bed, because now more than one subnet is involved in our experiment.

Quagga

Quagga is a routing software suite, providing implementations of OSPFv2, OSPFv3, RIP v1 and v2, RIPng and BGP-4 for Unix platforms, particularly FreeBSD, Linux, Solaris and NetBSD [69]. Quagga makes use of the hardware provided on a general purpose computer (PC) to implement a shared bus and shared memory router. A PC-based router is implemented on a multi-homed computer.

A standard PC architecture consists of three major blocks, which are the central processing unit (CPU), the random access memory (RAM), and an array of peripherals all of them tied together by a chipset. The chipset provides complex control and interconnection functions. The peripheral

[1]Internetwork Operating System - is the software used on the vast majority of Cisco Systems routers and current Cisco network switches.

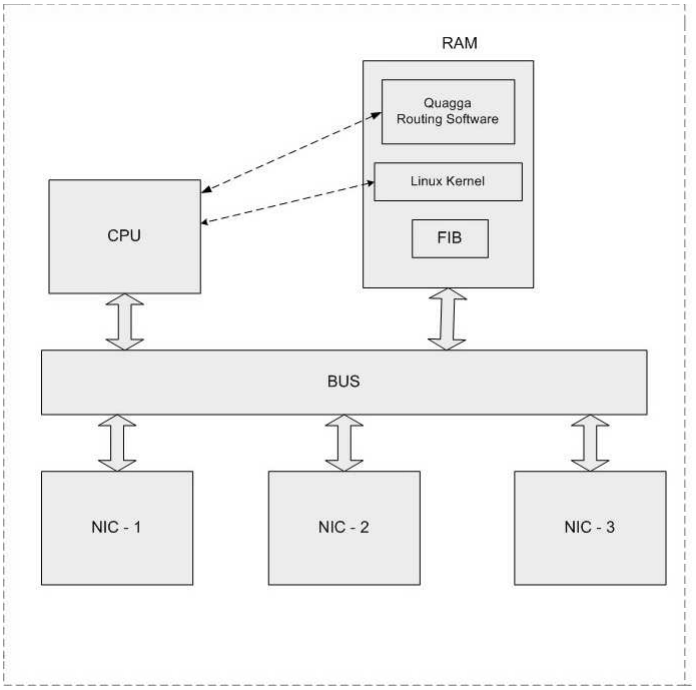

Figure 3.3: PC-based Router Architecture [70]

include the Network Interface Cards (NIC) and the storage. Figure 3.3 shows the architecture of a Quagga router.

The basic operation of a Quagga router is that NIC receive and store data packets in the main memory (RAM), the CPU executes routing instructions to route the packets to a correct output queue, and the NIC fetch packets from the RAM and transmit them on the wire. Therefore, each packet travels twice on the PCI bus, halving the effective bandwidth for the NIC to NIC packet flows [71].

The success of the Quagga software is attributed to the modular nature of the Linux networking code. The code has hardware independent IP stack with a well defined application programming interface toward the hardware-independent device driver, which is the glue in making the IP layer inter-operate with a diverse networking hardware. The Linux kernel network code implements a standard RFC1812 [72] IP router. The networking kernel code after completing the sanity checks, for example the checksum calculations, packets that are not addressed to the PC-router are processed by the routing function which determines the IP address of the next hop to which they will be forwarded, and the output interface on which they must be correctly queued for the transmission on the wire [70].

The kernel implements an efficient routing cache based on a hash table with collision lists; the number of hash is determined by the amount of

RAM memory in the host PC during initialization of the network code at boot time. A fast hash algorithm looks up the route for the outgoing packet in the routing cache, on hit the packet is forwarded to the proper queue, in case of a miss the forwarding in base (FIB) is then searched using a slow algorithm for longest prefix match entry. The dynamic FIB is built using routing protocols. These protocols provide means which routers communicate among themselves, giving each other information about the most efficient ways to route packets and the current state of the network. A dynamic routing table allows a router to switch data automatically to a back up route, if available, in the event of a failure on the main route [70].

Quagga is among a number of popular open-software projects available that implements the routing protocols for the TCP/IP network without the need of specific hardware (router). Similar projects to Quagga include, Click Modular router also based on Linux developed in MIT; Xorp currently under development in UC Berkeley, tipped support different hardware platforms from PC's to specialized network processor and also implements quality of service (QoS); LRP and Freesco are also free and open source distribution for Linux operating-system.

Quagga can be set-up on a multi-homed standard PC to run multiple routing protocols. Quagga deploys a collection of daemons, with roughly every daemon implementing a particular routing protocol. The daemons are; ospfd daemon that implements OSPF protocol version 2, the ripd implements the RIP protocol, bgpd daemon support the BGP version 4, and the zebra daemon that is responsible for Linux kernel routing table update changes and it redistributes the routes between different routing protocols [71].

Simple Network Monitoring Protocol (SNMP)

SNMP is management protocol for monitoring nodes in the Internet. SNMP is based on the manager/agent model consisting of an SNMP manager, an SNMP agent, a database of management information, managed SNMP devices and the network protocol. The SNMP agent manager provides the interface between the human network manager and the management system. The SNMP agent provides the interface between the manager and the physical devices being managed.

The SNMP manager and agent use an SNMP management Information Base (MIB) and a small set of commands to facilitate exchange of information. The MIB acts as data store and is organized in a tree structure with individual variables, such as point status or description, being represented as leaves on the branches. Variables in MIB are distinguished by a unique numeric tag called object identifier (OID). OID makes possible for SNMP queries to return unique messages regarding the managed device [73][74].

Cacti and RRDTool

RRDTool is a short form for round robin database tool. It is a logging and graphing tool for time dependant data, for example temperature, CPU load and network bandwidth. RRDTool is used with scripting languages like, Perl, PHP and Tcl to produce custom monitoring applications. RRDTool has a wide deployment in information technology resource monitoring applications, such as Cacti and MRTG.

Cacti is a network graphing solution designed to harness the power of RRDTool's data storage and graphing functionality. Cacti provides a fast poller, advanced graph templates, multiple data acquisition methods, and user management features out of the box. All of these wrapped in an intuitive, easy to use interface that is suitable for LAN-sized installations up to complex networks with hundreds of devices [76]. Cacti in addition to RRDtool also needs PHP scripting environment, MySQL database and a Web-server (in our case Apache as discussed in Chapter 2).

The components in Cacti work together to achieve three distinct functions, which results in the display of resource data of interest. The components are data retrieval, data storage and data display. Data retrieval is done via SNMP interface. A poller, which is a script written using PHP, is scheduled by the Linux job scheduler (crontab) to retrieve SNMP data on monitored device at regular time intervals. The RRDTool and MySQL database forms the storage portion of the data retrieved by the PHP poller. Lastly, using the RRDTool and the Web-server, data can be displayed on any machine with a network connection to the monitoring server by a Web-client.

The Cacti package includes graph templates and data queries that are sufficient for standard system monitoring purposes. However, more custom refined templates, for both data queries and graphs, are readily available on the Internet to supplement the standard package. All the configurations on Cacti, for example addition of new devices and creating graphs for the devices are done via the Web interface [76]. In our set-up Cacti is used for resource monitoring in our test-bed as described in the experiment scenarios.

3.3.3 Network Connections

In this chapter we deploy three distinct set-up scenarios. The three scenarios deploy a different combination of hardware and software tools, already described, to achieve specific results. However, the basic network performance metrics of interest do apply in all the three scenarios.

In Scenario A, the host A (attacker) is connected to the first Fast-Ethernet port of Miami, and is a source of SYN packets. Host B (victim) is connected to the first Fast-Ethernet port of Rio, and is the receiver (sink) of the SYN flood attack. Host C is connected to the second Fast-Ethernet port of Rio, installed with Cacti application for measurements, and Host D is connected

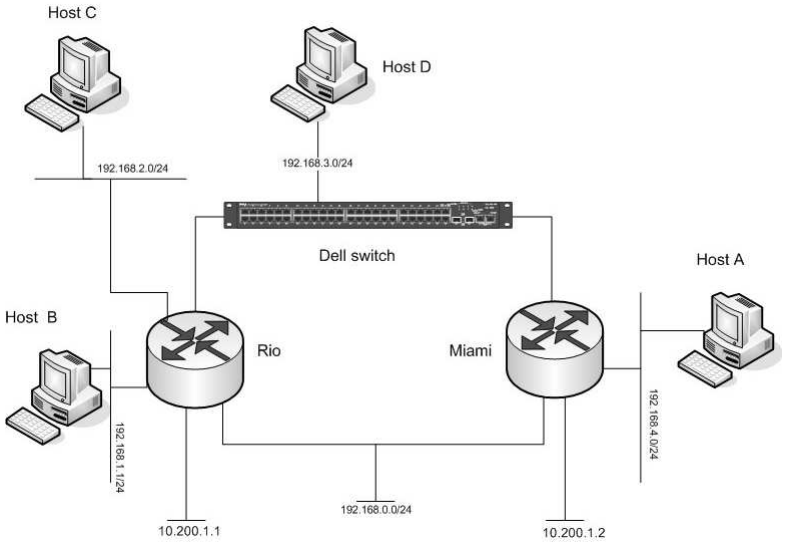

Figure 3.4: Network diagram with Cisco routers only

to the Dell switch, and runs a packet analyzer (tcpdump), the Dell switch is configured with a Tap port to facilitate the sniffing function.

The two routers exchange routing updates using OSPF routing protocol, running on the Ethernet interfaces connecting the subnet 192.168.0.0/24, shown on diagram. The network 192.168.3.0/24, joining the two routers and the host D via the Dell switch is configured as a static floating (back up) route for the connection of the two routers. To ensure that the 192.168.3.0/24 network is not installed in the routing tables under normal operating condition, a high metric as compared to the one of OSPF network was used. Figure gives 3.4 the diagrammatic representation of the connections.

In Scenario B, the physical connections and the components are exactly as Scenario A, only now the routing protocol is IS-IS (Intermediate System - Intermediate System) instead of OSPF as we saw in Scenario A. Again the back-up route on network 192.168.3.0/24 is installed with a higher metric than IS-IS for the similar reasoning of Scenario A. Figure 3.4 also describes the connection on Scenario B.

In Scenario C, the Miami router is replaced by the Quagga router. In this set-up the routing protocol used is again OSPF, with the rest of the components connected in the exact manner as in both Scenario A and Scenario B. Figure 3.5 gives a summary of the Scenario C set-up connection details

3.3.4 Experiment Procedure

In all the three scenarios, Scenario A to C, we go through similar steps to conduct the experiments. Therefore we describe the experiment procedure for all scenarios at once, to avoid repetition, and point out unique features

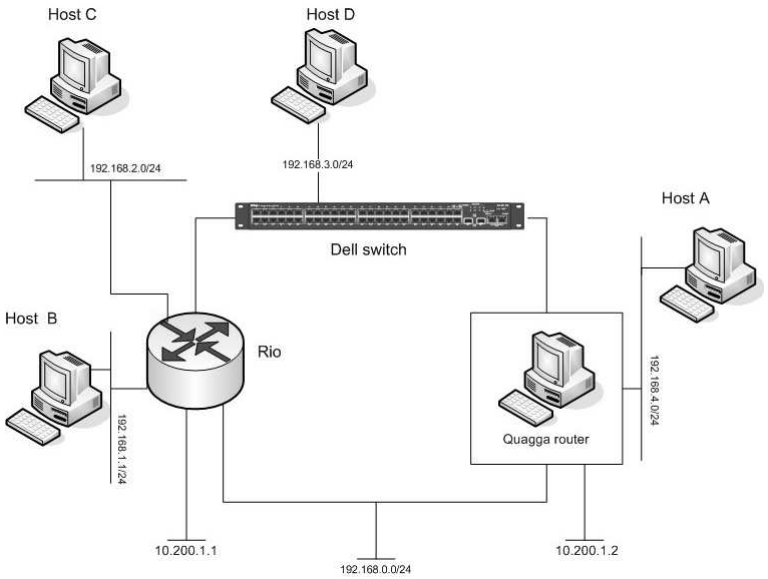

Figure 3.5: Network diagram with Cisco and Quagga routers

as they arise.

- Using the packet generator on host A (attacking host) , we set different packet rates of SYN packets, addressed to host B (victim host). Tables 3.1, 3.2 and 3.3 show the steps used for the adjustment of the packet rates.

- We monitored the SYN packets on the victim host during the experiments using the tcpdump, connectivity was inferred from this observation

- Host C was installed with Cacti, and was used to collect resource utilization, i.e, CPU, memory and bandwidth from the routers.

- Host D was used to monitor traffic on the back-up route using the tcpdump.

- During the experiments network delay was approximated by the average RTT of the ICMP echo packets from Host C to the Loopback interface of the next hop router (Miami or Quagga), depending on the scenario.

The above procedure constitutes a single iteration of the experiment, which gives a single row of results on the tables. Each iteration was run for at least ten minutes.

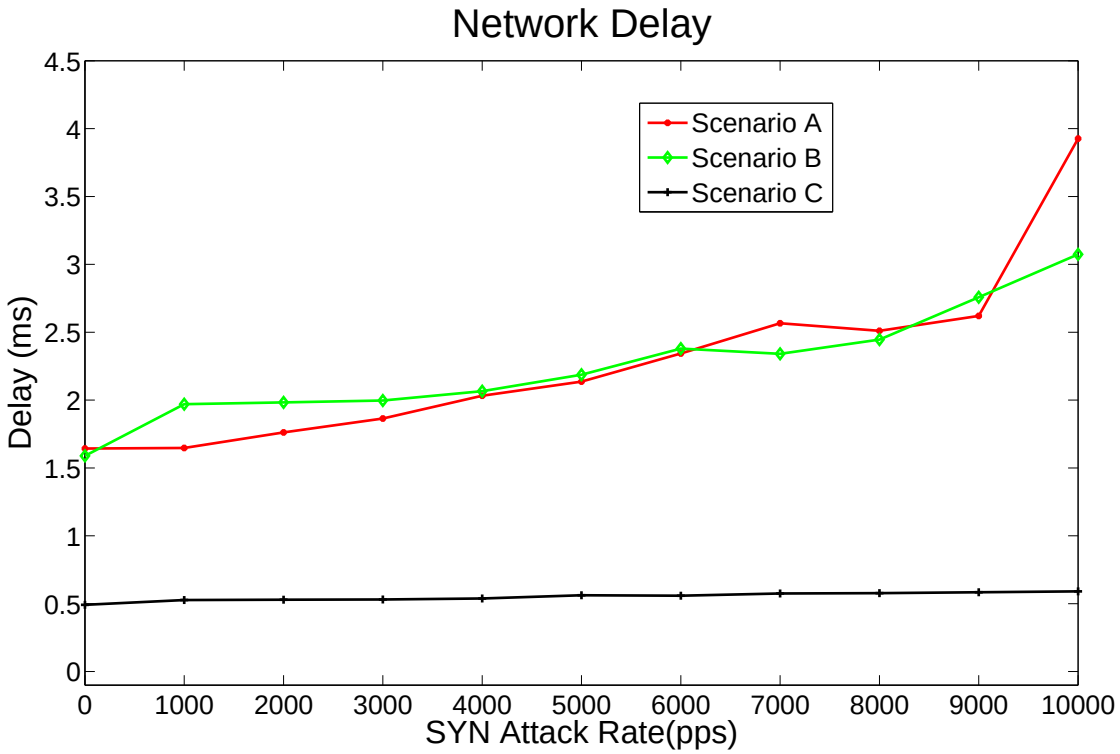

Figure 3.6: A network delay comparison between the three scenarios.

Table 3.1: Network with two Cisco routers running OSPF

Packet rate (pps)	OSPF status	CPU_Rio (%)	CPU_Miami (%)	Delay (ms)
0	Ok	1	2	1.643
1000	Ok	9	11	1.647
2000	Ok	12	13	1.762
3000	Ok	16	18	1.864
4000	Ok	21	22	2.032
5000	Ok	17	21	2.136
6000	Flaps	26	26	2.343
7000	Flaps	30	34	2.566
8000	Down	28	38	2.510
9000	Down	35	42	2.620
10000	Down	58	67	3.926

3.4 Results and Discussion

In this section, we present our results from the experiments in our scenarios. The results are aligned to answer the research questions put before on the introductory part of this chapter. In all the analysis results for every scenario is found on the Tables 3.1, 3.2 and 3.3 respectively.

We measured the network delay during the experiments; this was estimated by the ICMP packets as explained on the experiment procedure and results shown on the tables. In general, the network delay increased with the attack packet rate. The network delay for Scenario A and Scenario B show a comparable comparison between the two set-ups. In both Scenario A and B, the building components are similar, with a difference in OSPF and IS-IS protocols deployed respectively. The results tell us that there was no significant difference gained by the different choices of dynamic routing protocols.

However, the Scenario C shows some unique features from the last two despite the rising trend of delay with the attack packet rate. The network delay in this scenario is comparable low as to Scenario A and B. There is a decrease of about 84% between the maximum delay in Scenario A and B and maximum delay on Scenario C. The only difference on the latter scenario is the Quagga router in place of a Cisco router. This can be explained in terms of the computing performance capability of the Quagga host and Cisco routers. Whilst the Cisco router clocks at 50 MHz, the Quagga router clocks on dual processor at 2600 MHz. This gap of performance explains a quicker response on the echo packets. Thus in the end the Quagga-host is much quicker in response as to the replaced Miami Cisco router. The

Table 3.2: Network with two Cisco routers running IS-IS

Packet rate (pps)	IS-IS status	CPU_Rio (%)	CPU_Miami (%)	Delay (ms)
0	Ok	1	1	1.589
1000	Ok	7	3	1.970
2000	Ok	11	14	1.983
3000	Ok	14	15	1.997
4000	Flap	17	21	2.066
5000	Flap	23	19	2.187
6000	Down	28	22	2.379
7000	Down	31	29	2.341
8000	Down	39	38	2.446
9000	Down	40	51	2.758
10000	Down	43	66	3.074

network delay in the three scenarios is presented by the graph on Figure 3.6.

There was however a small deviation from the general trend of on increase of network delay with packet rate. The Table 3.1, for example, on packet rate of 7000 and 8000, the delay went down instead of going up. We dismiss this minor observation as an error of the measurement process. The same applies to the other tables.

The CPU and memory utilization of both routers was measured using Cacti, SNMP monitoring tool described in Software Configuration section. These resources are important to the network characteristics because a high utilization is an indicator of instability of the routers hence the network overall. In general all three scenarios show an increase in CPU utilization as the attack packet rate. In Scenario A and B, there is a pronounced rise of the CPU utilization of the Miami as compared to the Rio. Miami is the Cisco router connected to the attacking host (source of the SYN packets). This observation led us to believe there was more work done on Miami as compared to Rio, thus suggesting not all attack packets made their way to the victim host. However, our monitoring of all the router interfaces through Cacti and the local Cisco logs suggested nothing to prove this theory because no packet loss was explicitly recorded. The delay measurement, however, did report some loss of echo packets, with the loss increased with packet rate. Assuming a random loss the attack packets must have suffered the same fate.

The Scenario C shows, again as was in case of network delay, a deviation in CPU utilization although the general trend of rise with packet rate is obeyed. There is a less pronounced rise of CPU in this scenario as in comparison with

Table 3.3: Network with Cisco and Quagga router running OSPF

Packet rate (pps)	OSPF status	CPU_Rio (%)	CPU_Quagga (%)	Delay (ms)
0	Ok	2	1	0.492
1000	Ok	4	1	0.527
2000	Ok	4	1	0.529
3000	Ok	12	2	0.531
4000	Ok	15	2	0.538
5000	Ok	19	5	0.562
6000	Ok	21	6	0.559
7000	Flaps	27	8.5	0.575
8000	Flaps	30	9.5	0.578
9000	Flaps	32	10	0.584
10000	Flaps	42	10.5	0.591

Scenario A and B. Again this explained by the difference in CPU resources in Quagga-host and Cisco router.

There was no significant rise in memory utilization recorded in all three scenarios. The memory utilization at the peak of the attack was much less than the overall memory capacities of all the three routers. This is explained by the large amount of memory installed on the Quagga host (1000 MB), which was maybe more than enough for the packet range simulation in our experiments. For the Cisco routers, they are specialized hardware built for the function of routing packets, thus most of the functions are performed within the hardware (ASIC implementation) level.

The status of the routing protocols, OSPF and IS-IS, was observed in two ways: 1. by monitoring traffic on the back-up link (192.168.3.0/24), this link carries traffic only in the event of a failure on dynamic routing protocol, 2. Through status commands on the routers that display the instantaneous status of the routing protocol in them.

The tables of results indicate three distinct states for the respective routing protocols, "Ok","Flaps" and "Down". In OSPF interface status state machine, Figure 3.1 shows the state machine, the "Ok" status represents a DR or DR-Other status, in these two states the OSPF is functioning correctly. The "Down" status represents a similar state on the state machine, while the "Flaps" represent network under transients, in this status the network is unstable and keeps changing the status between up and down states. The same explanation applies for the IS-IS except there is a minor difference of the equivalent names in the IS-IS interface finite machine state shown in Figure 3.2. In IS-IS interface state machine there is no DR-Other state, hence on "Ok" the interface is on UP state, Down remains the same,

and “Flaps” is the transient state. Thus “Flaps” represent a network in transients.

In Scenario A, the status of OSPF changes from “Ok” to “Flaps” at an attack rate of 6000, and from “Flaps” to “Down” at a rate of 8000. The OSPF is configured to send “Hello” packets on intervals of 40sec, with adjacency removed after an interval of four Hello’s lapse without a reply from the neighbor. At a high SYN attack rate between 6000 and 8000 the adjacency between the Cisco routers goes up and down owing to a loss of Hello packets required to maintain the adjacency. The process of re-establishing the adjacency is not instantaneous, as by design OSPF spends time, this time is know as hold down, on Waiting state, before establishing full adjacency and move to Up state. The transition to Down-state at a packet rate of 8000 is because of a higher loss of Hello packets, such that it is not possible to establish the adjacency any longer.

In Scenario C, we replace the Miami router of Scenario A with a Quagga router, and the rest of the details remaining the same. The OSPF status changes to from “Ok” to “Flaps” at a packet rate of 7000, the status remained the same in the remaining range of our experiment, i.e., up to a packet rate of 10,000. The reason of the flaps is similar to Scenario A, loss of adjacency between the two routers attributed to loss of the Hello packets. However, in this scenario the OSPF process does not go completely down but rather remains in a transient. Since the Cisco routers are of exactly the same configuration, and the Quagga host is much powerful than the two of them, we expected a trend similar to Scenario A, if all the attack packets were forwarded to Rio. Thus, the OSPF process was never completely down because the Quagga host acted as a buffer of SYN attack by dropping the attack packets. The OSPF process was down at a much higher packet rate, of about 100 000.

In Scenario B, with IS-IS as the routing protocol. We notice the same trend as from Scenario A; however, refer to Table 3.2, the IS-IS starts to flap at a much lower rate of 4000 as compared to Scenario with OSPF. Also note, at a much lower rate of 6000, the IS-IS status moves to Down-state permanently. The observation is marginal and at best can be explained by the implementation specifics of both IS-IS and OSPF on Cisco hardware. Cisco hardware are proprietary, hence nothing is known about their implementations.

In all experiment scenarios we observed instability of the dynamic routing protocols, at a packet rate of 3000 and above, depending on a scenario. The instability observed is caused by the “monolithic” [77][78] architecture of the Cisco’s operating system (IOS): this means the operating system image runs a single image and all processes share the same address space. The IOS deploys a run-to-completion scheduling algorithm, allowing a process to consume as much CPU as it requires with no pre-emption mechanism. Also, there is no memory protection between the individual processes, which gives

a potential of one process corrupting data belonging to a different process, luckily, the IOS does not deploy third party software [79]. At a high packet rate of SYN packets, routers CPU-resource is tied to the packet forwarding engine, this leaves none or insufficient resources to run the dynamic routing protocol process within the router. This is a reason behind the instability of the routing protocols in our experiments.

3.5 Conclusion

The results presented in this chapter, show that a network that carries a SYN flood attack is also affected by the attack, where the effect is proportional to the attack. Contrary to our observation in Chapter 2, where a design flow on TCP three way handshake was exploited, in the current context the SYN flood exhausted the resources (CPU) of the network routers. In essence any type of packets, for example, ICMP or UDP, can be used with the same intensity to produce similar results.

In these experiments, the routers overload was determined by monitoring the traffic on the back-up static route. Since we know that at high CPU utilization the resources ought to be tied by the forwarding engine, this scenario leads to a dead or transient dynamic routing protocol, thus activating the back up route. The dynamic routing protocol deployed had a minimal influence on this observation, since the bottom line is any routing process requires CPU for maintenance and other network updates according to it's design, any luck of CPU thereof is bound to cause instability of the respective routing protocol.

The routers deployed in our experiments are quite old, in fact Cisco has put the model into the end of life category. Hence, there is a need of a re-run of the experiment with much newer models available to study the changes, if there any, obtained from the observed results. During our experiment we had only attack traffic, our request packets for connection to the victim host and some control traffic for monitoring, etc. In an actual scenario a network is bound to contain more cross traffic not involving the attack and victim host. In this case, as well, a different set of results might be deduced depicting even closer to a real life scenario in the Internet.

Chapter 4

Modeling

In this chapter, we seek a simplified model, which approximates the TCP SYN flooding attacks. We examine the characteristics of the SYN flood DoS attack, and attempt to capture the salient features of its effects on a victim host using a mathematical relation. The model is derived from the queuing theory. In this model we treat the finite buffers, storing the state of a TCP connection, as limited resources of a queuing system that dictate the probability of achieving a successful TCP connection. These buffers are referred to as the backlog in Chapters 1 and 2.

A number of publications discuss the subject of modeling of TCP SYN attacks; Wang et al [87], present a novel method using embedded Markov chains to characterize the loss probability and buffer occupancy of TCP SYN attacks, Aissani in [62] takes a different approach by modeling the residual service times of both attack and regular request packets, and solve the resulting partial differential equations to obtain the loss probability by a method similar to Sebastianov [67]. In all cases modeling of SYN attacks is complex and more often arrive at solutions that involve complex computation of results. We intend to make a number of mathematical simplifications to derive a simple model that estimates the probability of connecting to a TCP server under SYN attack, and compare the results with an experimental set-up and the complex models available.

In the sequel, we implement an experimental set-up to validate our model. The set-up closely follows the structure of the set-up in Chapter 2, with a major difference on the TCP client and the packet generator. In this chapter, we design a new multithreaded TCP client. With this client we can spawn (start) independent, separate and parallel TCP connections, as if from independent users, unlike the client in Chapter 2 offering sequential TCP client connection requests. The packet generator is also modified to read Poisson inter-departure times from an external file.

4.1 Problem Statement

In this chapter, we deploy the queuing theory to derive a simplified mathematical model for SYN attacks, and design an experiment to validate the derived model. In this chapter, we will answer the following questions:

- Derive a simplified mathematical model that estimates the salient features of TCP SYN flood attacks.
- Design a TCP client capable of initiating independent TCP connections at a controlled rate, thus, emulate a real life arrival rate of connection requests.
- Evaluate and contrast the results from the model and the experimental set-up.
- Compare our simple model with other models found in the literature.

4.2 Queuing Modeling

In this section, we provide the in-depth detail of our simplified model for the TCP SYN attacks. The model focuses on the consumption of the state storage that is a limited resource on a victim host, and a crucial characteristic of denial-of-service SYN attacks. In this case, the SYN flooding attack exploits the well known vulnerability on establishing a TCP connection, where the three-way handshake algorithm is used.

Queuing theory is a branch of applied probability theory. Its applications are in different fields, e.g communication networks, computers systems, machine parts and other applications in which a limited service is prescribed to a customer population. Examples of queuing system in everyday life: an admission queue to a football stadium, the waiting room in a city hall, hospital or bank hall, etc. In the Internet, the packets arriving at the input port of a router or switch are buffered in the output queue before transmission to the next hops towards their final destinations. In general, the subject of queuing can be described as follows: (i) arriving items (packets or customers), (ii) a buffer (waiting room), (iii) a service center, and (iv) departures from the system.

The queuing process, illustrated by the Figure 4.1, is a stochastic process[1] in nature. The queuing theory tries to answer questions like e.g. the mean waiting time in the queue, the mean response time (waiting time in the queue plus service times), mean utilization of the service facility, distribution of the number of customers in the system, etc. Again, the questions are mainly

[1] A stochastic process is a family of random variables $\{X(t) \mid t \in T\}$ defined on a given probability space, indexed by the time variable t, where t varies over an index set T.

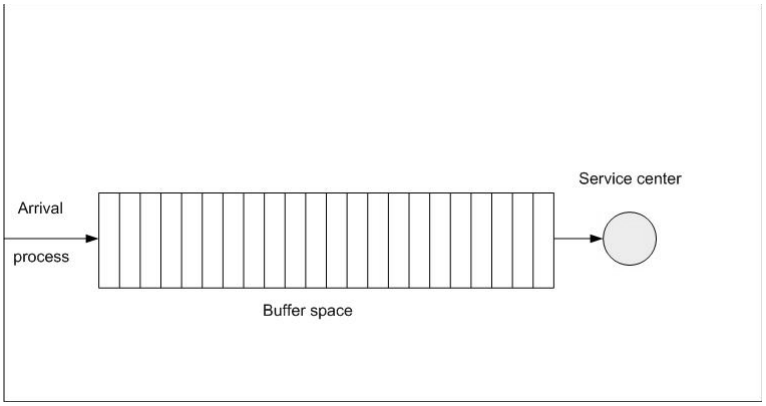

Figure 4.1: A general diagram of a queuing process.

investigated in a stochastic scenario, or mostly in approximate stochastic scenario, where the inter-arrival times of the customers and/or the services times are assumed to be random. Queuing systems may not only differ in their distributions of the inter-arrival and service times, but also in the number of servers, the size of the waiting line (infinite or finite), and the service discipline. Some common service disciplines are [54][64]:

- First in, first out (FIFO): a customer that finds the service center busy goes to the end of the queue.
- Last in, first out (LIFO): a customer that finds the service center busy proceeds immediately to the head of the queue. The customer will be serviced next, provided that no further customers arrive.
- Random Service: the customers in the queue are served in a random order.
- Round Robin: every customer gets a time slice. If service is not completed for a given customer, the customer re-enter the queue and wait for the next time slot.
- Priority disciplines: every customer has a priority assigned, the server selects always the customers with the highest priority. This scheme may, or may not use preemption.

The Kendall's Notation is used for a short characterization of queuing systems. A queuing system description using the notation looks as follows:

$$A/B/m/N/S$$

Where A denotes the distribution of the inter-arrival times, B denotes the distribution of the service times, m denotes the number of servers, N denotes

the maximum size of the waiting line in the finite case (if $N = \infty$ then the latter is omitted), and the optional S denotes the service discipline used, for instance, FIFO, LIFO and so forth, described earlier. In all cases, when S is omitted, the service discipline is assumed to be FIFO. For A and B the distributions can be Markov (M), Deterministic (D), Erlang-k (E_k), and General (G) [64].

4.2.1 Model Description

In the sequel, we use queuing theory to create a model to estimate connection loss probability on a SYN attack. We model the 3-way handshake module of TCP service in a host as queuing system. Therefore, the model will evaluate the use of half-open backlog seats on a victim host under defined conditions of SYN attack and legitimate (regular) traffic. Connection loss probability is an important measure in connection depletion DoS attacks, and one of the key performance metrics of the network. In general, SYN flood queueing process consists of two distinct arrival processes, for attack packets and regular connection requests, two distinct service process for attack packets and regular connection requests and, service points equaling the size of the backlog on a host. In our simplified model we attempt to merge the arrival process as a single queue, and we do the same for the service process, in order to reduce mathematical complexity.
We assume the following in order to simplify the analysis of the work.

1. The arrival process of SYN flood packets is Poisson distributed with a known rate.

2. The arrival process of legitimate TCP connection requests is Poisson distributed with a known rate.

3. The service time of an attack packet in the queue is the maximum holding time (timeout) defined in the queue.

4. The service time of a regular packet is Poisson distributed, counted from the epoch an SYN+ACK packet is sent to the epoch an ACK packet is received to complete the connection.

For the model we define the following variables.

- m, represents the number of servers in the system.

- λ_1, the rate of regular connection requests.

- λ_2, the rate of the attack (SYN flood) packets.

- λ, the total arrival rate into the queuing system ($\lambda = \lambda_1 + \lambda_2$), as a consequence of assumptions 1 and 2.

- n, represents the size of the backlog in the queuing system.
- t_0, time spent by an attack packet in the queuing system, i.e, the timeout.
- t_r, the mean time spent by a regular connection packet in the queue.
- t, the mean service time,

$$t = \left(\frac{\lambda_1}{\lambda_1 + \lambda_2}\right) t_r + \left(\frac{\lambda_2}{\lambda_1 + \lambda_2}\right) t_0 \tag{4.1}$$

  .
- μ, the average service rate of the queuing system, $\mu = 1/t$.

In our case, the number of servers equals to the number of buffer spaces ($n = m$). Thus the resulting queue is denoted as an $M/G/m/m$ queue using the Kendall's notation. The queue has a Markovian arrival process from our earlier assumption of Poisson arrival process with mean rate of, λ [54]. Furthermore, the queue is composed of two different service times: first, service time to regular (legitimate) TCP connection request packets, which is the epoch between the sending of a SYN+ACK packet by the server and the receipt of an ACK segment that completes the handshake from the client. Second, the service time of a SYN flood packet, the holding time (timeout), which is the maximum time a packet can spend in the system before being dropped. Moreover, Equation (4.1) provides a rough estimation of the service time by taking the average of both arrival process. However in the range of our measurements Equation (4.1) is similar to a case, which assumes maximum service time (timeout) for all packets. This is because of a small value of t_r compared to t_0. Thus, our model provides an approximate upper bound prediction on the probability of connection loss to a host under DoS attack.

In Linux systems, used in our model evaluation set-up, the timeout is dictated by the number retries (tcp_synack_retries), discussed in detail in Chapter 2. The backlog size is modeled by the number of servers in the queuing system, this parameter is also discussed in Chapter 2. The size of the buffers is the same as the number of servers, this is analogous to the classical telephony network, which the trunks [64] are modeled as servers and the queue size is the same as the number of servers available. Therefore, a customer on reaching a system when it is not full proceeds directly to receive service, otherwise she drops.

The state transition diagram of a queuing system is shown in Figure 4.2. The general system shown, contains m states, each state represents an occupied seat, by either regular or an attack packet, on the victim host's backlog. Therefore the system is in state 0, when empty at the start, and state m when full. We are interested in evaluating the probability of finding the

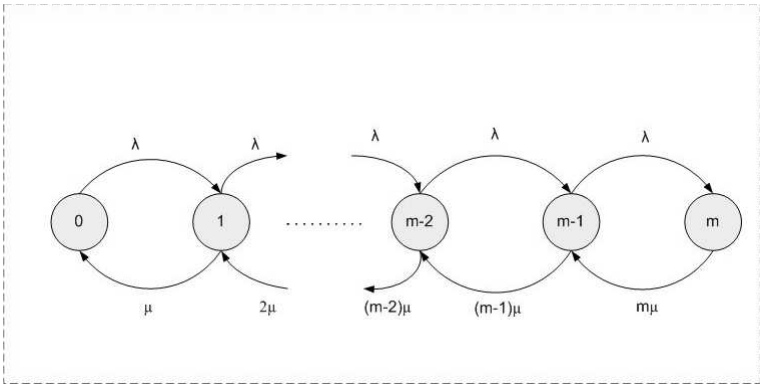

Figure 4.2: State transitions in an M/G/m/m queue.

system on state m, given a certain number of retries and arrival processes of SYN attack packets and regular connection requests to TCP service on the host.
We use the theory behind the general birth and death process and the law of conservation of probability to deduce the blocking probability. From the law of conservation of probability, the rate at which probability "accumulates" in any state n is the difference between the rates at which the system enters and leaves that state [65]. Assume the system contains arbitrary n states. Each state in our case, represents an occupied seat on the backlog. This produce the equations (4.2) and (4.3).

$$\frac{dp_n(t)}{dt} = \lambda p_{n-1}(t) + \mu_{n+1}p_{n+1}(t) - (\lambda + \mu_n)p_n(t),\ for\ , n \geq 1 \tag{4.2}$$

$$\frac{dp_0(t)}{dt} = \mu_1 p_1(t) - \lambda p_0(t),\ for\ , n = 0 \tag{4.3}$$

In the steady state, which we assume exists,

$$\frac{dp_n(t)}{dt} = 0,$$

and equations (4.2) and (4.3) become,

$$0 = \lambda p_{n-1} + \mu_{n+1}p_{n+1} - (\lambda + \mu_n)p_n,\ for\ n \geq 1 \tag{4.4}$$

$$0 = -\lambda_0 + \mu_1 p_1 \Rightarrow p_1 = \frac{\lambda}{\mu_1}p_0,\ for\ n = 0 \tag{4.5}$$

Using (4.5), equation (4.4) can be re-written to,

$$p_n = \frac{\lambda_n}{\mu_n\mu_{n-1}\ldots\mu_2\mu_1}p_0 = p_0\prod_{i=1}^{n}\frac{\lambda}{\mu_i},\ for\ n \geq 1 \tag{4.6}$$

and making appropriate substitutions of the values of μ as from the state transition diagram on Figure 4.2,

$$p_n = \frac{\lambda_n}{n\mu(n-1)\mu \ldots 2\mu 1\mu} p_0 = \frac{1}{n!} \left(\frac{\lambda}{\mu}\right)^n p_0, \ 0 \leq n \leq m \tag{4.7}$$

We also know that,

$$\sum_{i=0}^{K} p_i = 1$$

therefore,

$$p_0 = \left[\sum_{n=0}^{m} \frac{1}{n!} \left(\frac{\lambda}{\mu}\right)^n\right]^{-1} \tag{4.8}$$

Finally, eliminating p_0 in (4.7) using (4.8) we get,

$$p_n = \frac{(\lambda/\mu)^n/n!}{\sum_{i=0}^{m}(\lambda/\mu)^i/i!} \tag{4.9}$$

The equation (4.9) is famously known as Erlang-B formula, named after the work of a Danish engineer, A. K. Erlang, in 1918 [54]. We use this formula to calculate the probability of connection loss during a TCP SYN attack. The quantity λ/μ, is defined as the utilization, ρ, of the queuing system. Thus equation (4.9) simplifies to,

$$p_n = \frac{(\rho^n)/n!}{\sum_{i=0}^{m} \rho^i/i!} \tag{4.10}$$

4.3 Related Models

Modeling of DoS attacks has attracted a fare share of interests in the research community, partly fueled by the network security requirements. In this section we provide a brief description of selected models in literature, which we use as a baseline comparison with our model. The models try to capture the mathematical relationship between exhaustion of host resources' in relation to the probability of connection loss with a consideration of the services and arrival process of both attack and regular request packets. In the sequel, we use the models developed in [87] and [62] for comparison with our work.

Aisani et al, in [62] models the DoS attack as a stochastic process with the input queue consisting of a composition of regular and attack packets. The victim has a connection buffer (the backlog) queue, in which at most N half-open connections are allowed simultaneously. Furthermore, they also assume that connection requests arrive according to a Poisson process with a rate λ (both legitimate and attack traffic). Let the $v(t)$ and $u(t)$ represent

the number regular and attack packets at time t. Then through a rigorous mathematical evaluation using Sebastianov [67] method on the residual time of the attack and regular packets, they arrived on equation (4.11) that measures the probability of loss connection.

$$p_{i,j} = \frac{(\rho_1)^i(\rho_2)^j}{i!j!} \left[\sum_{i,j=0,0\leq i+j\leq N} \frac{(\rho_1)^i(\rho_2)^j}{i!j!} \right]^{-1} \tag{4.11}$$

where i and j are number of regular and attack packets respectively in a given system state, and $\rho_i = \lambda_i/\mu_i$

Wang et al in [87], build a queuing model for SYN flood attack and characterize it using two-dimensional embedded Markov chain. In their case, as previously in [62], they also concentrate on the consumption of the limited resources, which is a common feature of DoS attacks.

Wang et al assume that the victim has a buffer of size N, each half-open connection is held in the queue for a deterministic period b, the half-open connection of regular packet is held for random time which is exponentially distributed with parameter μ, the arrivals of request and attack packets is a Poisson process with rates λ_1 and λ_2 respectively. The two arrivals processes are independent of each and the holding times of the half-open connections. In addition, let n_1 and n_2 represent the number of regular and attack packets at time t. The two dimensional Markov chain with the given states space is defined as,

$$\Omega = \{(n_1, n_2) : n_1 \geq 0, n_2 \geq 0, n_1 + n_2 \leq N\} \tag{4.12}$$

Wang et al then propose an A level-eliminating algorithm to compute the stationery distribution of the two-dimensional Markov chain. The resulting stationery probability distribution gives the connection loss probability as the equation (4.13),

$$P_{loss} = \sum_{i=0}^{N} \Pi_{i,N-i}. \tag{4.13}$$

The two models are similar in both solving the probability of finding the system at a saturated state, with different attack service and arrival processes. The mathematical analysis used in both cases to arrive at their respective, as briefly described above, and the practical results obtained, as Section 4.7 describes, are different.

4.4 Measurements

In this section, we describe the measurements needed for comparison between our model and the experiment, also the comparison between our model and

models found in [87] and [62]. There are several aspects that we consider in order to perform experimental measurements that are closely related to our model: 1. Create a Poisson process of arrival for the SYN attack packets, 2. Create a Poisson arrival process for TCP client requests, and 3. Measure the probability of connection loss.

Firstly, we need to generate SYN packets at a rate required by the experiment. The model also assumes a Poisson distribution of the attack packets. In the previous chapter we already described a C - script, which was used to generate SYN packets with different selected rates. We modify the script to read the inter-departure times of sending packets from a file. This file contains random generated numbers with Poisson distribution from MATLAB.

Secondly, we need a process to test TCP connections on a host that is on the receiving end of a TCP attack. However, this process should have the ability of opening client connections at specified rate with each connection opened independently of all other previous connections. That is to say as long as the average inter-arrival time of the connection is adhered, the next connection proceeds regardless of the status of all previous connections. The inter-departure time for the requests is also read from a Poisson generated random number file.

Thirdly, once we have controlled rate of SYN attack packets, and the TCP request packets, the probability of failing to make a connection is estimated by a ratio of successful TCP request to the total number of requests.

We consider how our model compare against other models found in literature. In this task, we define parameter k, which is the ratio of arrival rates between the attack and the regular request packets: $\lambda_2 = k\lambda_1$.

However, we do not compare the results from other models with our experiment. The reasons are two fold, firstly, the parameters used by the authors cannot be simulated in our test-bed without going through a rigorous process of changing some parameters in the Linux kernel and re-compile the kernel. Secondly, both methods used are complex enough to warrant special methods to evaluate results, that were covered by the authors, thus in the interest of time, which is limited we only cover the experimental comparison of our model only.

4.5 Experiment set-up

In this section, we provide the details of our experiment carried out to measure the probability of connection loss. We deployed a network consisting of three computer hosts, host A, B, C, and a network switch. The diagram on Figure 4.3, shows the set-up connectivity. We installed the SYN packet generator on host A, host B is the victim of the SYN attack and host C runs the multi-threaded C script that opens parallel TCP client connections.

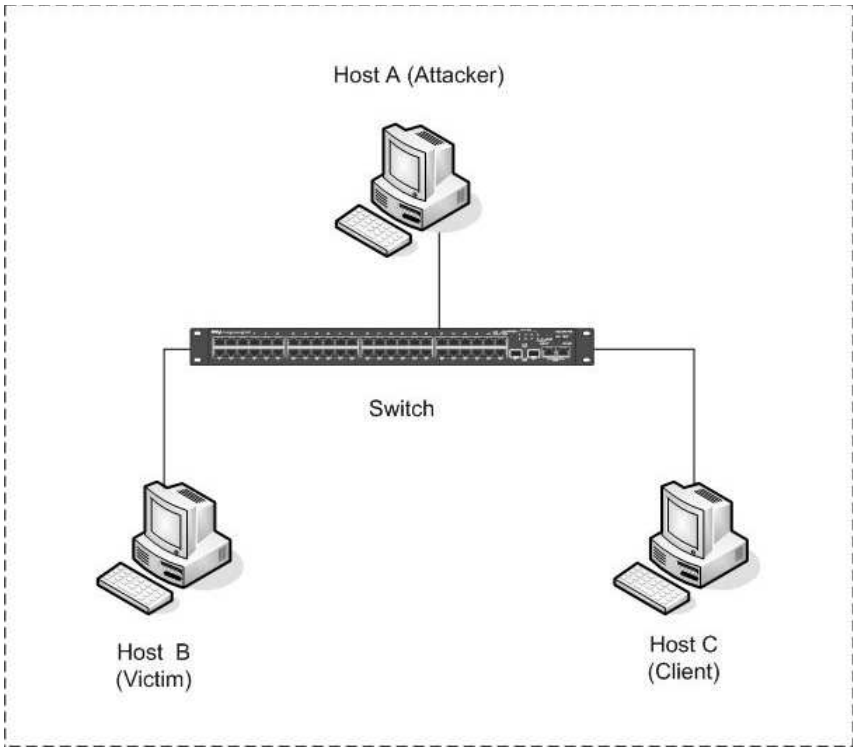

Figure 4.3: The experiment set-up: Model validation.

4.5.1 Multithreading

Multithreading (MT) is a technique that allows one program to do multiple tasks concurrently. A thread is sometimes referred to as a lightweight process. A thread will share all global variables and file descriptors of the parent process which allows the programmer to separate multiple tasks easily within a process. Linux operating systems contain Posix-Thread (PThread) programming environment that can be used to implement multithreaded applications. We used PThread to develop the multithreaded TCP client [66].

The multithreaded script developed is the answer to generate independent TCP requests according to the specification of our model. Using the script one can set the mean inter-departure times between any two contiguous TCP requests to the host victim. The script uses Linux signal library to start (spawn) an independent thread of TCP connection, from a value read from a list of Poisson generated random numbers, after the expiration of the previous inter-departure time set. The script joins (merge) all the threads before exiting and provides statistics on the total number of requests that were successful. The data provided is used to estimate the connection loss probability.

4.5.2 Procedure

In the set-up, we have a TCP server, in Apache Web-server described in Chapter 2, running on host B, the Poisson enabled SYN packet generator on host A and the multithreaded TCP client on host C. A packet analyzer (tcpdump) was installed on host-C for verification whenever needed. The TCP flood is directed to the port 80, standard HTTP port, at all times during the experiments. The following procedure is used to collect data from the experiment.

1. A selected rate of SYN packets is set and the packet generator is initiated from host A. The selected rates are shown in Table 4.1.

2. A desired rate of regular (HTTP requests) packets is selected and set on the multithreaded client. The client requests are initiated simultaneously with the attack packets in Step 1.

3. We record the statistics of successful connections from each trial and, a probability of connection loss is calculated and inserted in Table 4.1, for different backlog sizes.

The above steps are repeated on every trial. Table 4.1 show results with a TCP backlog of 20, 32, and 40 respectively. In the experiment we operate in the range of which the ratio of the average rate of attack packets and regular packets is between 0.1 and 2. The average rate of regular packets is kept

constant at 10pps. During the experiments we disabled the TCP retries in Linux. This was done because the theory behind model assumes no retries on requests once blocked [54].
Although in real world, the backlog sizes are much higher than those deployed in our experiments, in fact, about 1024 in a modern Linux machine at the time of writing this document [6], the smaller backlog sizes prevents us from requiring approximation methods on evaluating our results (MATLAB does so with factorial from higher numbers), and makes possible for us to compare our results with other models, which were evaluated in the same range.
The probability of loss is calculated from the equation,

$$p_L + p_S = 1 \tag{4.14}$$

where p_L and p_S are the probabilities of loss and success on a connection respectively on a given experiment trial.

Table 4.1: Experimental results of values of connection loss probability, with different backlog sizes.

SYN rate(pps)	Connection Loss Probability		
	backlog = 20	backlog = 32	backlog = 40
1	0.380	0.250	0.180
2	0.640	0.473	0.433
4	0.713	0.683	0.663
5	0.776	0.691	0.681
8	0.840	0.783	0.716
10	0.821	0.813	0.721
16	0.941	0.913	0.840
20	0.957	0.926	0.901

The measurements of k and probability of loss on Model 1 and Model 2 where extracted from Table 1 in [62], all the measurements were done at a constant regular packet requests of rate 10pps. We used the same values to generate the arrival rate of attack packets for given k's and use our model to evaluate the probability of connection loss that is inserted on Tables 4.2–4.4. Thereafter, we plot graphs for probability of connection against k in Figures 4.4–4.6, to demonstrate the comparison between the three models.

4.6 Results

The graphs on Figures 4.4, 4.5 and 4.6 show the comparison of the results from our experiment and the model developed in Section 4.2.1. The graphs

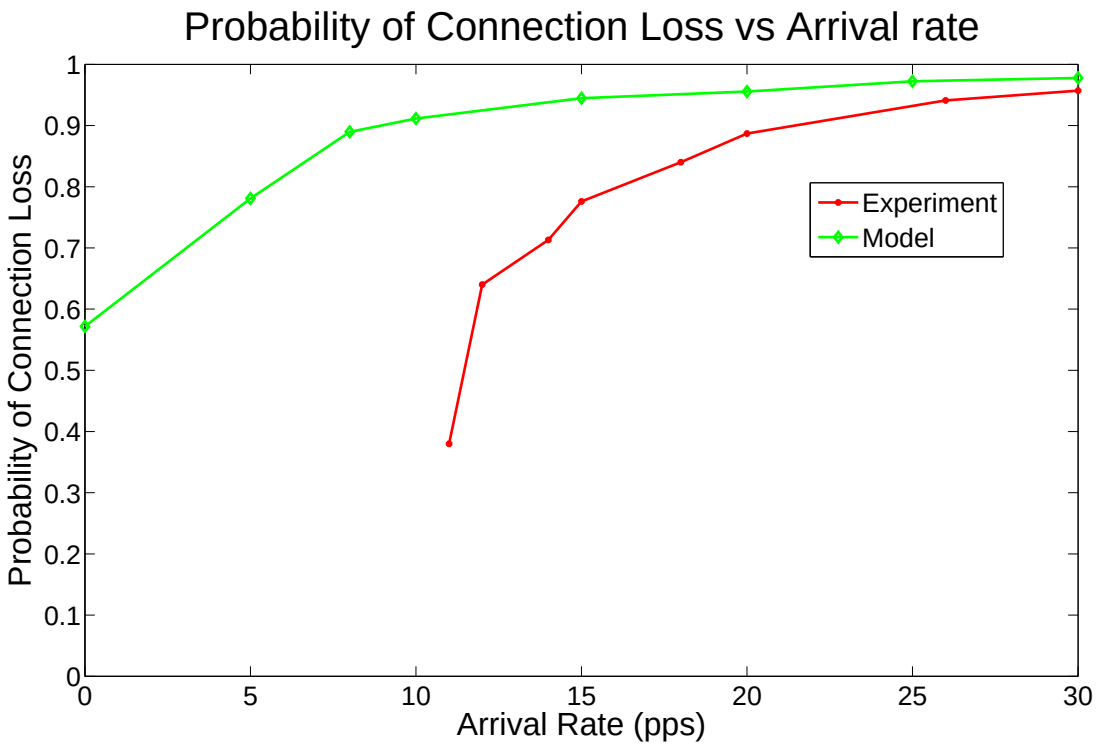

Figure 4.4: Comparison between our and experimental results on a host with a backlog of 20.

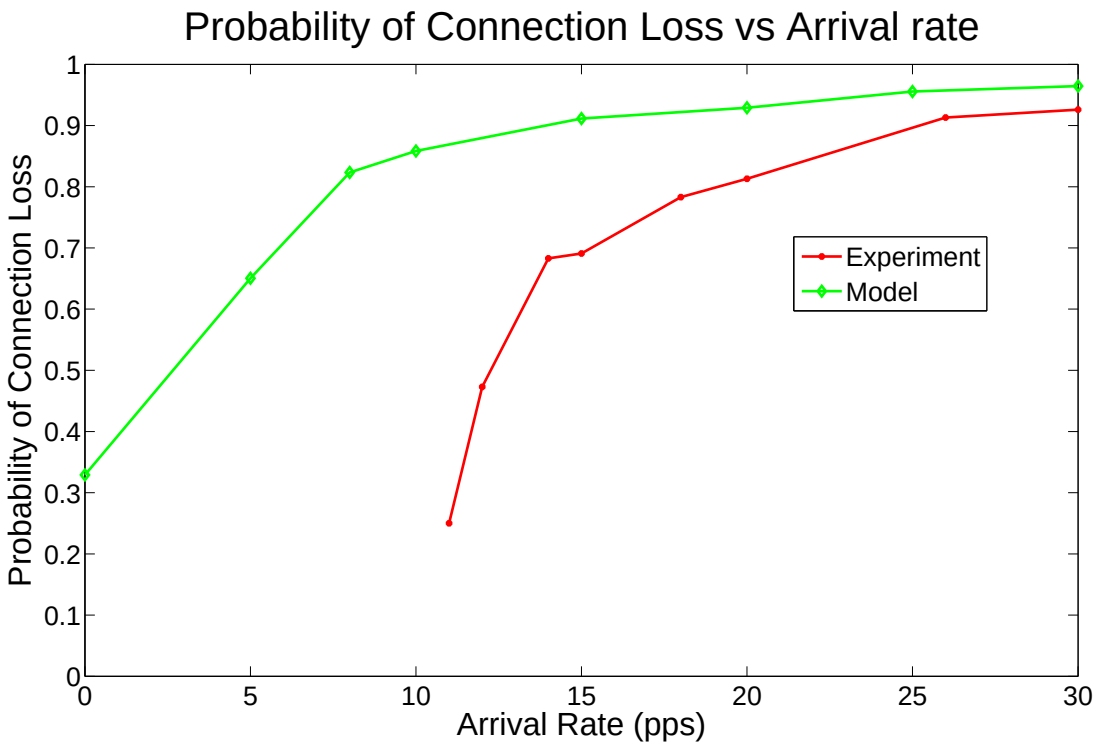

Figure 4.5: Comparison between our model and experimental results on a host with a backlog of 32.

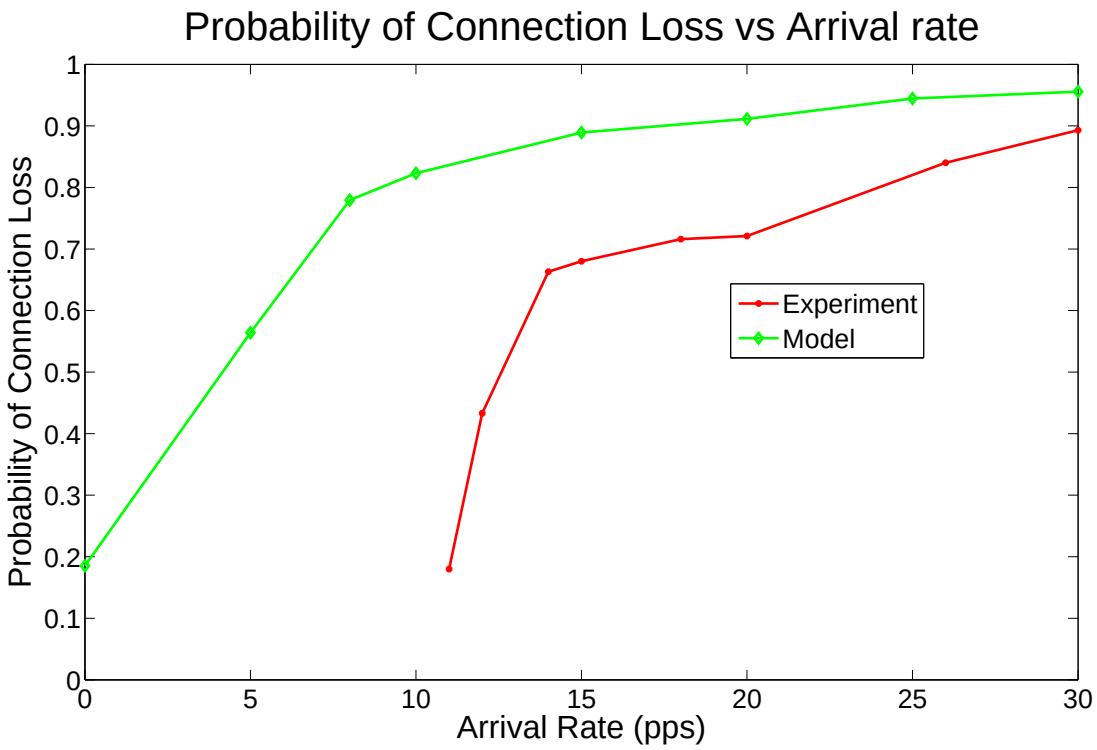

Figure 4.6: Comparison between our model and experimental results on a host with a backlog of 40.

Table 4.2: Comparison of loss probabilities for model 1, 2 and 3 with backlog size of 10.

k	backlog=10		
	Model 1	Model 2	Model 3
0.00	0.01	—	—
0.20	0.5	0.8324	0.2146
0.40	0.9	0.8559	0.5380
0.70	0.9	0.8809	0.7246
1.00	0.9	0.8986	0.8047
1.25	0.9	0.9097	0.8435
1.60	0.9	0.9218	0.8767
1.90	0.9	—	0.8959
2.30	0.9	—	0.9131

summarize the results obtained from a victim host with backlog size of 20, 32, and 40 respectively.

In general, the results from our model are much higher than those observed in the experiments, which accounts for the upper bound. The strength of the attack is more pronounced as the backlog size decreases, i.e., going from the backlog of 40 to 20. The lower the backlog the earlier users are bound to experience service degradation from a SYN attack. The probability of connection loss rises sharply with an increase of attack packet during experiments.

We compare our model with models found in [87] and [62]. Tables 4.2–4.4 contain values extracted from Table 1 in [62] coupled with results from our model in the same set of values of k, backlog and holding time. Model 1 and Model 2 belong [87] and [62] respectively. The comparison results show the loss probability increase with the increase of the attack traffic load. Both graphs show an increase on the ability to sustain the attack with increased backlog. All models tend to converge starting from $k = 1.5$ with an increasing value of k. Our model (Model 3) show a low probabilities as compared to the other models. Model 2 and Model 3 show a comparable results and a similar graph nature, while Model 1 posses a unique deviation from the rest.

4.7 Discussion

In this section, we explain our results from the comparison of both experimental data with our model, and the comparison of our model with the selected models from the literature.

From the experiment, the results from the model are close to one for

Table 4.3: Comparison of loss probabilities for model 1, 2 and 3 with backlog size of 20.

k	backlog=20		
	Model 1	Model 2	Model 3
0.00	—	—	—
0.20	—	0.6665	0.0019
0.40	0.45	0.7128	0.1589
0.70	0.9	0.7624	0.4590
1.00	0.9	0.7975	0.6121
1.25	0.9	0.8197	0.6882
1.60	0.9	0.8437	0.7540
1.90	0.9	0.8597	0.7922
2.30	0.9	0.8766	0.8264

increasing arrival rates compared to the experiment, in all the graphs (Figures 4.4–4.6). There is a wide gap between the experimental results and the model, which narrows as the arrival rate increase. The reason behind this observation is the simplified service time linear approach of our model, on the average service time estimation. Therefore in the experiment, with higher values of arrival rates, the model closely approximates the connection loss probability from the experimental results. Contrary in lower arrival rate the simplified model delivers a poor approximation of the probability.

The graphs on Figures from the experiment 4.4– 4.6 show as we expected that the more backlog resources deployed on a host the more resilient it is against a SYN attack. There is a sharp rise in the loss probability in all three cases, from the experiment results, as the attack rate is increased, which is followed up by plateau approaching the y-ordinate one, the maximum probability. The graphs from our model show a similar curve nature.

Our model does not compare well with the results found from other models in literature. Model 1 and Model 2 are complex (contrary to our simplified model), and were derived from complex mathematical distribution and require special algorithms [87] for evaluation. The results however are not very far off in justification of the complexity on both Model 1 and Model 2. In essence Model 2 seems to be upper bound itself, because the results from the model are always close to one in the range of measurements. Model 1, is better from the rest, but with also a large complexity on evaluation. In addition, our model did not compare well with rest on the literature, but we hope it serves as a starting point for designing less complex models for TCP SYN attacks.

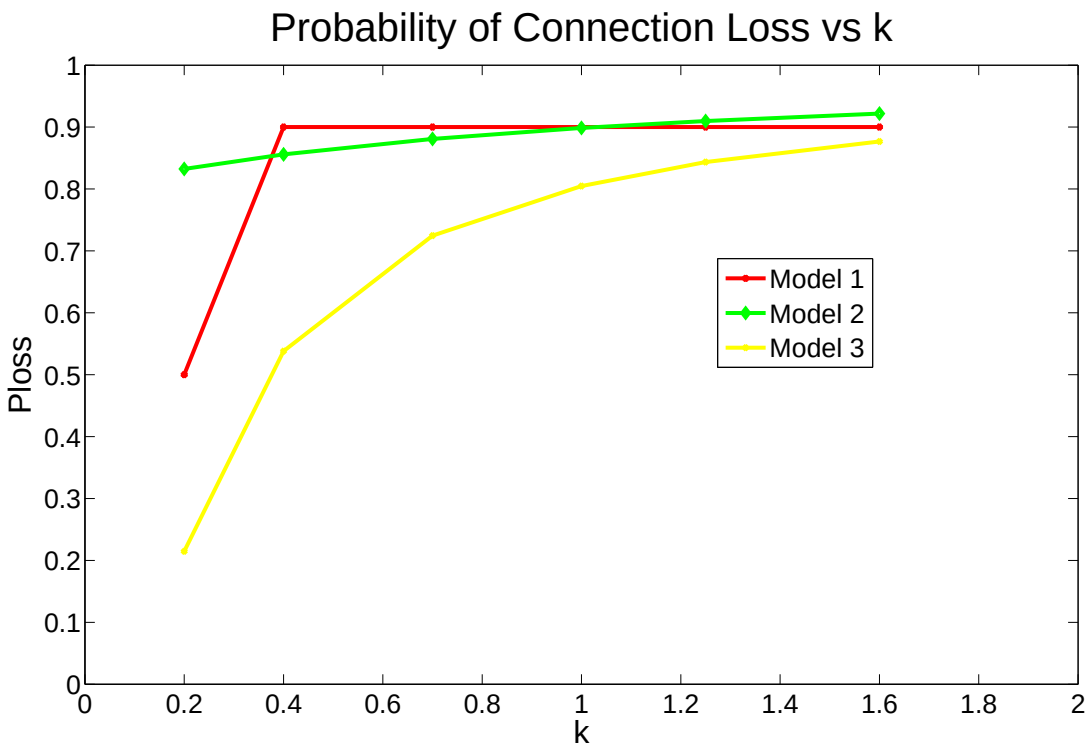

Figure 4.7: Comparison between Model1, Model2 and Model3 with backlog 10.

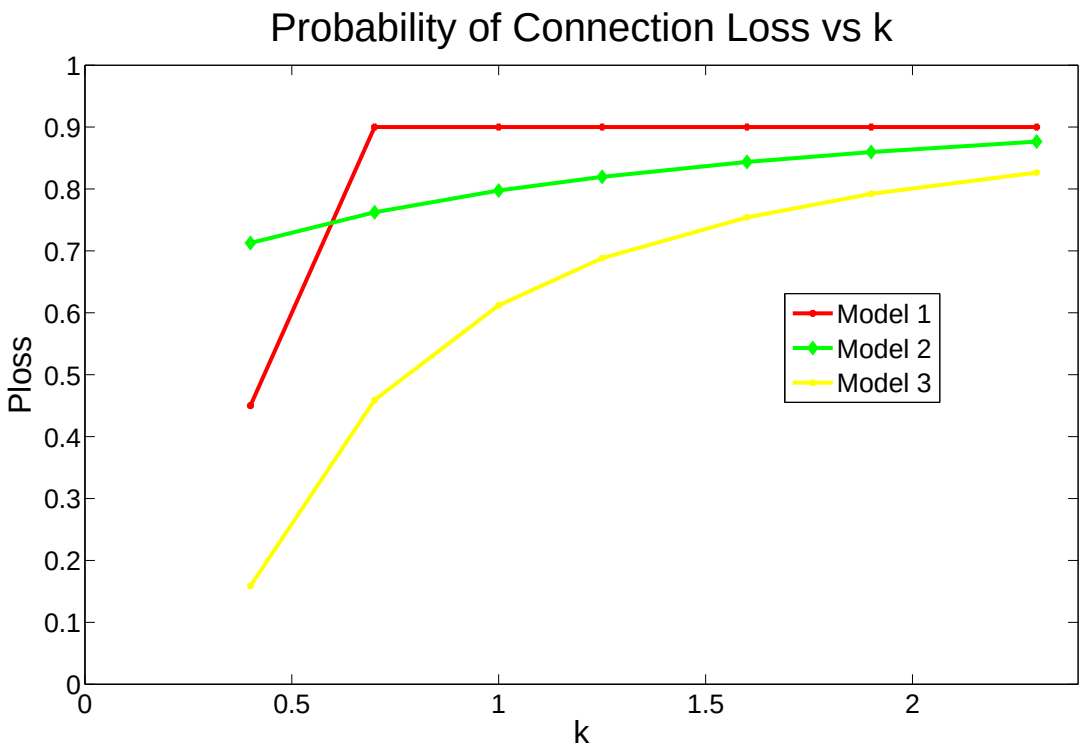

Figure 4.8: Comparison between Model1, Model2 and Model3 with backlog 20.

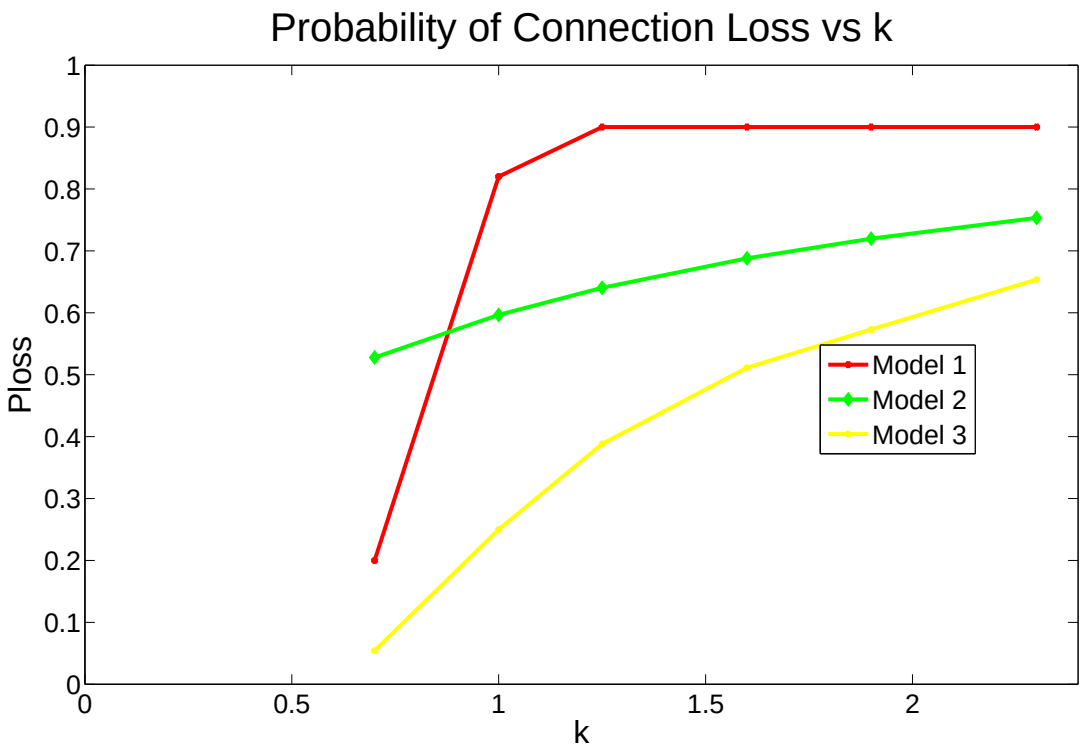

Figure 4.9: Comparison between Model1, Model2 and Model3 with backlog 40.

Table 4.4: Comparison of loss probabilities for model 1, 2 and 3 with backlog size of 40.

k	backlog=40		
	Model 1	Model 2	Model 3
0.00	—	—	—
0.20	—	0.3456	—
0.40	0.001	0.4321	—
0.70	0.2	0.5279	0.0542
1.00	0.82	0.5966	0.2498
1.25	0.9	0.6404	0.3881
1.60	0.9	0.6880	0.5114
1.90	0.9	0.7198	0.5731
2.30	0.9	0.7534	0.6537

4.8 Conclusion

In this chapter, we designed a simplified model to estimate the loss probability on connecting to a server under TCP SYN DoS attack. Along with the model we designed an experiment to validate the results, and we compared the results from the model to existing models of the same kind found in literature. The results from other models in the literature were not compared with experiments because of complexity of evaluation and time. The model developed is an upper bound inferred from our experimental results, which were always below the one's approximated by the model.

The comparison of the model we developed and models found in literature did not yield good results, but the results obtained are not far off to current results in literature. This calls for more work to be done in order to develop simple model that can actually estimate the TCP SYN flood DoS with reasonable accuracy. The current models although better, they involve a high level of complexity that can inhibit cost-effective deployment.

Chapter 5

Conclusions And Future Work

The Internet has evolved over the years from a network connecting researchers around the globe, to an indispensable communication network, connecting about every economic sector with billions of nodes world wide. The early vision of the Internet dictated the early aspects, notably security, reliability and availability, which still apply today although with a complete different service requirement on the network. The Internet at present provides a range of services from online gaming to complex banking transaction. Thus, a stringent set of security requirements are now in place to safe guard user data. Various forms of attacks have graced the Internet over the course of its evolution. A common form of attack is denial of service (DoS). DoS attacks consume a remote host or network's resources, thereby denying or degrading service to legitimate users. Distributed denial of service attacks are DoS attacks in which one victim is attacked from multiple sources. DoS attacks can be launched in several ways. One approach is to exploit victim's design weakness, such as in ping of death. Another approach is to impose computationally intensive task on victims, such as encryption and decryption computation. Another approach is to flood the network or host with a barrage of packets, for example in UDP and TCP floods.

In this thesis we have focused on the DoS attack emanating from TCP floods. These attacks are executed by exploiting the three-way handshake protocol on initiating a TCP connection. The exploitation is one of the inherent flaws in the design features of the Internet. The connection algorithm blindly assumes that the other party is a trusted node and will respond in to a connection request in the defined manner. The TCP flooding DoS attacks deplete the connection data structures of the victim's TCP backlog and prevent it from serving legitimate users. Literature rate TCP flood based attacks to constitute more than 90% [16] of all DoS attacks observed in the Internet. The famous reported and documented TCP flood

attacks are those of February 2000, which affected major commercial 1 such as yahoo.com, cnn.com and ebay.com.

5.1 SYN flood on a host

In Chapter 2 we have studied the performance of a host under SYN flood attack. The study distinguishes the fact that the effect of SYN flood on a host is affected by the operating-system installed. The operating-system implements the specification of the TCP as defined by standard documents, and also controls the resources of the host. In the study we selected Linux operating-system. Linux defines three configuration parameters for amortizing the effect of SYN flood attack on the host. The parameters are, the backlog - is the capacity of TCP half-open connection seats, retries - is the holding time of TCP-half open connection, and syncookies - this activates the SYN cookies on a host. SYN cookies provide a protection on the host by stop storing the states of half-open connections.

In the chapter, we provided an experiment to study the behavior of SYN attack on the host. The experiment aimed at studying the behavior of the Linux host under different rates of attack, and SYN flood protection parameters mentioned above. The findings show that with SYN cookies activated the tolerance of a Linux box is dictated by network bandwidth capacity and the Network Interface Card (NIC). This means that since the connection set-up does not store any state, the flood can cause a denial of service by either flooding the network or the NIC of the host, whichever comes first. Furthermore, a big backlog size provides more protection to attack as more resources become available to store half-open connections. Finally, the holding time, this is the amount of time a packet stays in the half-open state before is dropped. The shorter the holding time the quicker is the process of reclaiming the half-open connections. But, this time has to be carefully calibrated so as not to exceed an average RTT of a client request. In the end we study a commonly deployed architecture for commercial Web-severs and evaluate its vulnerability to SYN flood attacks. For this we showed that it is still possible to take out a standard Website using a reasonable amount of SYN flood packets.

5.2 SYN flood on network

In Chapter 3 we studied the effect of SYN flood attacks on the underlying network carrying the attack. The impetus to the work found in this chapter is to determine the behavior of routing nodes, dynamic routing protocols deployed and the network performance under SYN flood attack.

In the chapter, we performed measurement from three experiment scenarios. The scenarios involved dynamic routing protocol OSPF and IS-IS,

and also a combination of Cisco and Quagga routers. In the experiment we found that in most cases it is not possible to exploit the three-way handshake vulnerability similar to attacks in Chapter 2. However, flooding the network with SYN flood of sufficient strength (explicitly discussed in the chapter) can have a detrimental effect to the network performance. The flooding causes an increase in the packet loss, delay, instability of the dynamic routing protocols and to the extreme case driving the routing protocols out of service completely. In the study we also identified how an architectural design issue in Cisco 2600 series routers, used in our experiments, contributes to the instability of the deployed routing protocols. Lastly we recognize the fact that the results obtained can also be achieved using flooding packets from any other protocol. This is quite different in Chapter 2, where TCP packets only could achieve the results.

5.3 Modeling

Lastly, in Chapter 4, we delve into queuing theory to develop a model that estimates the loss connection probability during a SYN flood attack. In theory we model the 3-way handshake module of a host as a queuing system. From the system we identify the distribution of the arrival rate of both attack packets and regular request, and their respective service time's distribution in the system.

The comparison between the model and an experiment validated the model as an upper bound. The results from the model were consistently higher at all arrivals as compared to those derived from the experiment set-up. The upper bound model grows more accurate at a high packet rate. Our upper bound model did not compare well to the selected models for comparison. In our designed model simplicity is a crucial factor, contrary to the models found in the literature used for the comparison.

5.4 Future Work

The results obtained in our work are encouraging and leave a broad avenue to explore for further research works. Therefore, we suggest the following works to be considered in future studies.

- There is a need to develop a better, simple and accurate model that predicts important characteristics of DoS. These models are often essential in network security applications, for example, Intrusion Detection Systems(IDS). A complex solution usually makes deployment expensive and often needs custom tailored hardware.
- The cascading failures in networks, in general this behavior is attributed to an overload of the network the leads to a collapse of the net-

work. On of the means to induce an overload on networks is through the TCP SYN flooding. The literature contains a rich collection research based on mathematical analysis of such phenomenon. It will be of research interest to replicate the behaviour through an experimental set-up.

- The three-way hand-shake exploited in TCP is still used for the design of newer protocols in the Internet, one of the example is creating adjacency for neighboring routers running IS-IS protocol. It is an interesting research endeavor to study how the security lessons learnt from the TCP counterpart were considered in the overall design of the protocol.

Appendix A

Packet Generator

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <netdb.h>
#ifndef __USE_BSD
#define __USE_BSD
#endif
#define __FAVOR_BSD /* use BSD'ish TCP/UDP header definitions */

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <time.h>
#include <netinet/in_systm.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <signal.h>
#include <arpa/inet.h>

int src_port, initsport;
int start_seq;
int start_ip_id;
int i=1;

struct itimerval tout_val;

struct psd_tcp {
struct in_addr src;
struct in_addr dst;
```

```
unsigned char pad;
unsigned char proto;
unsigned short tcp_len;
struct tcphdr tcp;
};



unsigned short in_cksum(unsigned short *addr, int len)
{
int nleft = len;
int sum = 0;
unsigned short *w = addr;
unsigned short answer = 0;

while (nleft > 1) {
sum += *w++;
nleft -= 2;
}

if (nleft == 1) {
*(unsigned char *) (&answer) = *(unsigned char *) w;
sum += answer;
}

sum = (sum >> 16) + (sum & 0xFFFF);
sum += (sum >> 16);
answer = ~sum;
return (answer);
}


unsigned short in_cksum_tcp(int src, int dst, unsigned short *addr, int len)
{
struct psd_tcp buf;

memset(&buf, 0, sizeof(buf));
buf.src.s_addr = src;
buf.dst.s_addr = dst;
buf.pad = 0;
buf.proto = IPPROTO_TCP;
buf.tcp_len = htons(len);
memcpy(&(buf.tcp), addr, len);
return in_cksum((unsigned short *)&buf, 12 + len);
```

```
}


void run(int k)
{
struct ip ip;
struct tcphdr tcp;
int sd;
const int on = 1;
struct sockaddr_in sin;


//srand(getpid());
u_char *packet;
packet = (u_char *)malloc(60);

ip.ip_hl = 0x5;
ip.ip_v = 0x4;
ip.ip_tos = 0x0;
ip.ip_len = sizeof(struct ip) + sizeof(struct tcphdr);
ip.ip_id = htons((unsigned short)start_ip_id); /* 16 bits */
ip.ip_off = 0x0;
ip.ip_ttl = 64;
ip.ip_p = 6;
ip.ip_sum = 0x0;
ip.ip_src.s_addr = inet_addr("192.168.1.12");
ip.ip_dst.s_addr = inet_addr("192.168.1.3");
ip.ip_sum = in_cksum((unsigned short *)&ip, sizeof(ip));

memcpy(packet, &ip, sizeof(ip));

tcp.th_sport = htons((unsigned short)src_port);
tcp.th_dport = htons(80);
tcp.th_seq = htonl(start_seq); /* 32 bits ..*/
tcp.th_off = sizeof(struct tcphdr) / 4;
tcp.th_flags = TH_SYN;
tcp.th_win = htons(32768);
tcp.th_sum = 0;
tcp.th_sum = in_cksum_tcp(ip.ip_src.s_addr, ip.ip_dst.s_addr, (unsigned short *)&tcp
memcpy((packet + sizeof(ip)), &tcp, sizeof(tcp));

if ((sd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0) {
perror("raw socket");
exit(1);
```

```
}

if (setsockopt(sd, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on)) < 0) {
perror("setsockopt");
exit(1);
}


memset(&sin, 0, sizeof(sin));
sin.sin_family = AF_INET;
sin.sin_addr.s_addr = ip.ip_dst.s_addr;



if (sendto(sd, packet,ip.ip_len, 0, (struct sockaddr *)&sin, sizeof(struct sockaddr)
   perror("sendto");
   exit(1);
}

src_port = (initsport + start_seq) % 65536;
start_seq = start_seq + 1;
start_ip_id = start_ip_id +1;


free(packet);



}


int main(int argc, char **argv)
{

initsport = src_port = 1024 + (rand()%2000);
start_seq = rand();
start_ip_id = rand();

tout_val.it_interval.tv_sec = 1;
tout_val.it_interval.tv_usec = 0;
tout_val.it_value.tv_sec = 1;
tout_val.it_value.tv_usec = 0;

setitimer(ITIMER_REAL,&tout_val,0);
```

```
signal(SIGALRM,run);

while(1)
{
;
}

return 0;


}
```

Appendix B

TCP client

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <netdb.h>
#include <unistd.h>

void error(char *msg)
{
    perror(msg);
    exit(0);
}

int main(int argc, char *argv[])
{
    int sockfd, portno, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;

    char buffer[256];
    if (argc < 3) {
       fprintf(stderr,"usage %s hostname port\n", argv[0]);
       exit(0);
    }
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");
    server = gethostbyname(argv[1]);
    if (server == NULL) {
```

```
        fprintf(stderr,"ERROR, no such host\n");
        exit(0);
    }

    /* initialization of the serv_addr structure */
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
         (char *)&serv_addr.sin_addr.s_addr,
         server->h_length);
    serv_addr.sin_port = htons(portno);


    if (connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr)) < 0)
        error("ERROR connecting, connect failed");



    close(sockfd);

    printf("The connection closed successful\n");

    return 0;
}
```

Appendix C

Curriculum Vitae

Cliff N. Maregeli was born in Dar-es-salaam, Tanzania, at April 13, 1980. He obtained bachelors degree from the Faculty of Electrical Engineering at Dar-es-salaam University in 2005, in Electrical Engineering majoring in Computer Engineering.
His graduation project involved the design of an academic content management system.

The project involved the use of UML version 1.5 object oriented designs, using PHP, MySQL and Cascading Style Sheets (CSS) to produce a course content management system for the Faculty of Engineering at the University of Dar-es-salaam. In the project he was supervised by Justinian Anatory, then an assistant lecturer at the Faculty of Electrical Engineering, University of Dar-es-salaam. After graduation he joined Vodacom Tanzania, a cellular company, as an IT systems engineer from August 2005 to September. During his tenure at Vodacom (2005-2008), he was involved in management of the Voice over IP (VoIP) for the corporate offices and the contact centre, Value Added Systems (VAS) and Post-paid billing platform databases.

In September 2008, he joined the TU Delft University, in the Netherlands, to pursue a Masters Diploma in computer engineering. For his masters thesis he worked on a study for TCP SYN attacks and their effects on networks carrying the attacks. He is a graduate student from the TU Delft University.

Bibliography

[1] B. Forouzan. *TCP/IP Protocol Suite*, McGraw Hill, 1221 Avenue of the Americas, New York, third edition, 2006.

[2] D. Schuehler, and J. Lockwood. *TCP-Splitter: A TCP/IP Flow Monitor in Reconfigurable Hardware*, Proceedings of the 10^{TH} Symposium on High Performance Interconnects Hot Interconnects (HotI'02), 2002.

[3] B Al-Duwairi, and G. Manimaran. *Distributed packet pairing for reflector based DDoS attack mitigation*, March 1, 2006.

[4] F. Lau, S. Rubin, M. Smith , and L. Trajkovic. *Distributed denial of service attacks*, In Proceedings of Systems, Man, Cybernetics IEEE International Conference, pages 2275 - 2280 vol.3, October 2000.

[5] S. Kumar. *Smurf-based Distributed Denial of Service (DDoS) Attack Amplification in Internet*, Second International Conference on Internet Monitoring and Protection,2007.

[6] J. Lemon. *Resisting SYN flood DoS attacks with a SYN cache*, FreeBSD project.

[7] T. Dubendorfer, A. Wagner, and B. Plattner. *An Economic Damage Model for Large-Scale Internet Attacks*, Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises,2004, pages 223-228.

[8] S. Bellovin. *Security Problems in the TCP/IP Protocol Suite*, Computer Communication Review_vol. 19 no.2, April 1989, pages 32-48.

[9] J. Moy. *OSPF version 2*, RFC 1247, July 1991.

[10] H. Gredler, and W. Goralski. *The Complete IS-IS Routing Protocol*, Springer, 2005.

[11] J. Postel. *Internet Protocol*, RFC 791, September 1981.

[12] W. Eddy. *TCP SYN Flooding Attacks and Common Mitigations*, RFC 4987, August 2007.

[13] C. Schuba, I. Karsul, M. Kuhn, E. Spafford, A Sundarama, and D Zamboni. *Analysis of a Denial of Service Attack on TCP*, Proceedings of the 1997 IEEE Symposium on Security and Privacy,pg 208,1997

[14] B. Ziegler. *Hacker Tangles Panix Web Site*, Wall StreetJournal, September 12, 1996.

[15] M. Jensen, N. Gruschka, and N. Luttenberger. *The Impact of Flooding Attacks on Network–based Services*, Proceedings of the 2008 Third International Conference on Availability, Reliability and Security, pages 509-513, 2008.

[16] T. Nakashima, and T. Sueyoshi. *Performance Estimation of TCP under SYN Flood Attacks*, Proceedings of the First International Conference on Complex, Intelligent and Software Intensive Systems, pages 92-99, 2007.

[17] D. Moore, G. Voelker, and S. Savage. *Inferring Internet Denial-of-Service Activity.*, ACM Transactions on Computer Systems (TOCS), Volume 24, issue 2, pages 115-139, May 2006.

[18] Cisco Systems Inc. *Defining Strategies to Protect Against TCP SYN Denial of Service Attacks*, September 1996.

[19] Cisco Systems Inc. *Internetworking Technology Handbook*, `http://www.cisco.com/en/US/docs/internetworking/technology/\discretionary{-}{}{}handbook/ito_doc.html`.

[20] A. Kuzmanovic, and E. Knightly. *Low-Rate TCP-Targeted Denial of Service Attacks and Counter Strategies*, IEEE/ACM Transactions on Networking (TON), Volume 14, Issue 4, August 2006.

[21] Computer Emergency Response Team (CERT). *IP Spoofing Attacks and Hijacked Terminal Connections*, Jan 1995.

[22] IP Business news. *FCC Investigates Comcast Traffic Shaping*, `http://www.ipbusinessmag.com/departments/article/id/348`.

[23] K. Goh, B. Kahng, and D. Kim. *Universal Behavior of Load Distribution in Scale-Free Networks*, Phys.Lev.Lett, volume 87, December 12, 2001.

[24] C. Lin, J. Liu, H. Huang and T. Yang. *Using Adaptive Bandwidth Allocation Approach to Defend DDoS Attacks*, In Proceedings of Multimedia and Ubiquitous Computing Conference, pages 176 - 181, April 2008.

[25] H. Wang, D. Zhang, and K. Shin. *SYN–dog: Sniffing SYN Flooding Sources*, Proceedings for 22nd International Conference of Distributed Computing Systems (ICDCS), July 2002.

[26] Y. Bouzida, F. Cuppens, and S. Gombault. *Detecting and Reacting against Distributed Denial of Service Attacks*, In the Proceedings of Communications, June 2006, pages 2394-2400.

[27] P. Ferguson, and D. Senie. *Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing*, RFC 2827, Jan 1998.

[28] V. Velasco. *Introduction to IP Spoofing*, SANS Institute, November 2000.

[29] J. Lemon. *Resisting SYN flood Dos Attacks with SYN cache*, Proceedings of the BSDCon 2002 Conference, Feb 2002.

[30] J. Lemon. *Syncookies, FreeBSD System Manager's Manual*, FreeBSD-6.1, 2006.

[31] C. Perkins. *IP Mobility Support*, RFC 2002, October 1996.

[32] M. Hadley, E. Rescorla *Internet Denial-of-Service Considerations*, RFC 4732 November 1996.

[33] D. Yuan, and J. Zhong *A lab Implementation of Sync Flood Attack and Defence.*

[34] K. Elleithy, D. Blagovic, W. Cheng, and P. Sideleau. *Denial of Service Attacks Techniques: Analysis, Implementation and Comparison*, Journal of Systematics, Cybernatics and Informatics.

[35] CERT Coordination Center. *Denial of Service attacks*, `http://www.cert.org/tech_tips/denial_of_service.html`.

[36] P. Boyle. *Intrusion Detection FAQ: Distributed Denial of Service Attack Tools: trinoo and wintrinoo.*, SANS Institute, April 17, 2007.

[37] D. Dittrich. *The "Tribe Flood Network" distributed denial of service attack tool*, University of Washington, October 21, 1999.

[38] A. Adnan, M. Alam, and A. Aktaruzzaman. *TCP SYN Flood DoS Attack Experiments in Wireless Network.*

[39] David Dittrich. *The "stacheldraht" distributed denial of service attack tool*, University of Washington, December 31, 1999.

[40] S. Dietrich, N. Long, and D. Dittrich. *Analysis of "Shaft" a distributed denial of service tool*, University of Washington, December 31, 1999.

[41] J. Barlow, and W. Thrower. *TFN2K Analysis, revision 1.3*, March 7, 2000.

[42] P. Crucitti, V. Latora, and M. Marchiori. *Model for Cascading Failures in Complex Metworks*, The American Physical Society, 2004.

[43] A. Motter, and Y Lai. *Cascade-Based Attacks on Complex Networks*, The American Physical Society, 2002.

[44] D. Williams. *Australian SaaS Recruitment Portal Under DDoS Attack*, November 16, 2009.

[45] R. Chang. *Defending against Flooding-Based Distributed Denial-of-Service Attacks: A Tutorial*, IEEE Communications Magazine, October 2002.

[46] K. Nichols, S. Blake, F. Baker, and D. Black. *Definition of the Differentiated Services Field (DS Field)*, December 1998.

[47] P. Almquist, and F. Kastenholz. *Towards Requirements for IP Routers*, RFC 1716, November 1996.

[48] Bennahum, D. *PANIX ATTACK*, `http://memex.org/meme2-12.html`, October 1996.

[49] P. Ferguson, and D. Senie. *Defeating Denial of Service Attacks which employ IP Source Address Spoofing*, RFC 2827, May 2000.

[50] J. Mirkovic, amd P. Reiher. *A Taxonomy of DDoS Attack and DDoS Defense Mechanisms*, 2004.

[51] CERT Coordination Center. *Similar Attacks Using Various RPC Services*, `http://www.cert.org/incident_notes/IN-99-04.html`.

[52] NISCC Technical Note 06/02. *Response to Distributed Denial of Service (DDoS) Attacks.*

[53] C. Adams, and J. Gilchrist. *The CAST-256 Encryption Algorithm*, June 1999.

[54] P. Van Mieghem. *Perfomance Analysis of Communications Networks and Systems*, Cambridge University Press. 2006.

[55] M. Newman. *The mathematics of network.*

[56] J. Wu, Z. Gao, and H. Sun. *Cascade breakdown in scale-free networks with community structure*, December 19, 2006.

[57] R. Albert, H. Jeong, and A. L. Barabasi. *Nature*, London, 1999.

[58] M.E.J Newman and M. Girvan. *Finding and Evaluating Community Structure in Networks*, Phys.Lev.Lett, 2004.

[59] R. Callon. *Use of OSI IS-IS for Routing in TCP/IP and Dual Environments*, December 1990.

[60] `http://en.wikipedia.org/wiki/E-mail_bomb`

[61] M. Landler, and J. Markoff. *Digital Fears Emerge After Data Siege in Estonia*, May 29, 2007.

[62] A. Aissani. *Queueing Analysis for Networks Under DoS Attack*, ICCSA 20008, Part II, LNCS 5073, pp. 500-513, 2008.

[63] M. Long, C. Wu, and J. Hung. *Denial of Service Attacks on Network-Based Control Systems: Impact and Mitigation*, November 26, 2004.

[64] A. Willig. *A Short Introduction to Queueing Theory*, July 21, 1999.

[65] W. Stewart. *Probability, Markov Chains, Queues and Simulation*, Princeton University Press, 2009.

[66] B. Lewis and D. Berg. *Multithreaded Programming with PTHREADS*, Sun Microsystems Press, 1998.

[67] L. Foster, J. Cao, Cleveland W, D. Lin, and D. Sun. *Internet traffic tends towards Poisson and independent as the load increases*, Springer Heidelberg, 2003.

[68] A. Basu and J.G. Riecke. *Stability Issues in OSPF Routing*

[69] Quagga official website. `http://www.quagga.net`

[70] V. Eramo, M. Listanti, and A. Cianfrani. *Switiching Time Measurement and Optimization Issues in GNU Quagga Routing Software.* IEEE Globecom, 2005

[71] A. Bianco, J. Finochietto, G. Galante, M. Mellia, and F. Neri. *Open-Source PC-Based Software Routers: A Viable Approach to High-Perfomance Packet Switching* Springer-Veralag Berlin Heidelberg 2005.

[72] F. Baker. *RFC - Requirement for IP Version 4 Routers*, Network Working Group, June 1995.

[73] J. Case, M. Fedor, M. Schosffstall, and J. Davin. *RFC 1157 - Simple Network Management Protocol (SNMP). May 1990*

[74] `http://www.dpstele.com/layers/l2/snmp_l2_tut_part1.php`

[75] `http://oss.oetiker.ch/rrdtool/`

[76] `http://www.cacti.net/`

[77] http://www.techworld.com.au/article/212755/cisco_ios_vs_juniper_junos_technical_differences

[78] http://www.velocityreviews.com/forums/t372212-which-os-is-behind-ios.html

[79] http://www.blackhat.com/presentations/bh-europe-08/FX/Whitepaper/bh-eu-08-fx-WP.pdf

[80] http://www.menog.net/menog-meetings/menog4/presentations/MENOG4-ISIS-Tutorial.pdf

[81] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. *Address Allocation for Private Internets*, Network Working Group, Request for Comments: 1918, February 1996.

[82] J. Fusco. *The Linux Programmers's Toolbox*, Prentice Hall, 2007.

[83] D. Kirkland. *TCP protocol*, http://manpages.ubuntu.com/manapages/aunty/man7/tcp.7.html

[84] J. Postel *RFC793 - Transmission Control Protocol*, DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION, September 1981.

[85] S. Deering *RFC 1112 - Host Extension for IP Multicasting*, Network Working Group Request for Comments: 1112, August 1989.

[86] W. Stevens *RFC2001 - TCP Slow Start, Congestion Avoidance, Fast Retransmit*, Network Working Group Request for Comments: 2001, January 1997.

[87] H Wang, D Zhang, and K Shin. *Detecting SYN Flooding Attacks*, In Proceedings of the IEEE Infocom, September 1, 2008.

[88] T Schroeder. *Scalable Web Server Clustering Technologies*, IEEE Network, 2000.

[89] P. Van Mieghem. *Data Communication Networking*, Techne Press, Amsterdam, 2006.

[90] D. Boteanu, J. M. Fernandez, J. McHugh, and J. Mullins. *Queue Management as a DoS counter-measure?*, Springer, Heidelberg, 2007.