

PROJECT REPORT

DEFENDING SYN ATTACK IN TCP USING CRYPTOGRAPHY

SUBMITTED IN PARTIAL FULFILMENT OF THE DEGREE OF
BACHELOR OF TECHNOLOGY

by

Akhlesh Gupta
Dua Gaurav Ashok
Gunjan Kumar Gupta

Under the guidance of
Dr. M. P. Sebastian



2004

Department of Computer Engineering
National Institute of Technology, Calicut

National Institute of Technology, Calicut
Department of Computer Engineering

Certified that this Project entitled

DEFENDING SYN ATTACK IN TCP USING CRYPTOGRAPHY

is a bonafide work carried out by

Akhlesh Gupta
Dua Gaurav Ashok
Gunjan Kumar Gupta

*in partial fulfilment of their
Bachelor of Technology Degree
under our guidance*

Dr. M. P. Sebastian
*Assistant Professor
Dept. of Computer Engineering*

Dr. V.K. Govindan
*Professor and Head
Dept. of Computer Engineering*

Acknowledgement

Words will not suffice to express fully, our deep sense of gratitude to Dr. M.P. Sebastian, Professor, Dept. of Computer Science Engineering, for the guidance and support extended by him, throughout the course of this work. He has not only been our mentor, but also our main source of inspiration.

We express our wholehearted thanks to all our friends, for their suggestions of immense value and encouragement through out the course of this work.

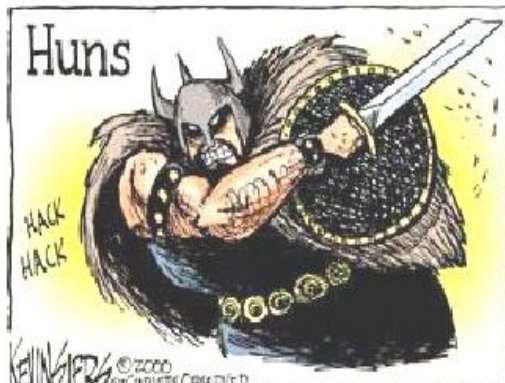
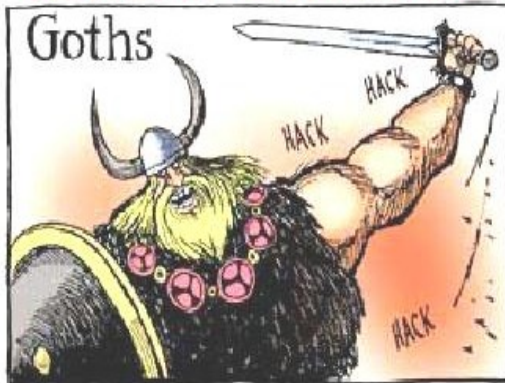
Akhlesh Gupta
Dua Gaurav Ashok
Gunjan Kumar Gupta

Abstract

In the existing TCP three way handshake process, *SYN Flood* attack is the most common Denial of Service attack. In this attack, server is continuously being bombarded with SYN packets from attacker, for each SYN packet the server will acknowledge the request but instead of acknowledging back the server, attacker will simply drop the ACK packets and continue sending SYN packets. Thus the server locks up space in the partial connection queue for the unacknowledged SYN packets until the packet is timed out. If the rate of arrival of SYN packets is much more than the rate of time out of SYN packets in the partial connection queue plus the rate of ACK received from the client, then server will not be able to serve all the new requests and some of the legitimate requests will not get service.

Our project is based on restricting the *SYN Flood* attack by using cryptography. In our solution we will use two modes Normal TCP mode and Cryptographic TCP mode. In Normal TCP mode the server will behave like existing TCP. When the partial connection queue becomes full, server will switch from Normal TCP mode to Cryptographic TCP mode. It will return back to Normal TCP mode only when the partial connection queue has any vacancy. Thus, by using Cryptography we give service to all authenticated clients whatever be the size of the partial connection queue. It gives an edge in the performance of TCP as all legitimate clients will get service and any attacker which uses spoofing will not be allowed to lock up resources. But there is a disadvantage that there is no scope for retransmission from the server side during handshaking.

BRINGING CIVILIZATION TO ITS KNEES...



Contents

1 Introduction	5
1.1 What is Denial of service attack?	5
1.2 Effects of DoS attack	6
1.3 History Attacks.	6
1.4 Generic DoS attacks.	7
1.4.1 Smurf attack.	7
1.4.2 SYN Flood attack	7
1.4.3 UDP Flood attack	7
1.4.4 Teardrop attack.	8
1.5 What is SYN Flood attack?.	8
1.6 Motivation	8
2 Security considerations in TCP	9
2.1 TCP header format.	9
2.2 Three way handshake in TCP.	11
2.3 State Transition Diagram.	12
2.4 Weakness of the TCP protocol	12
2.5 SYN Flood attack in TCP	14
2.6 Severity of SYN attack	15
2.7 Existing Security Mechanisms.	16
3 Design Principles of Cryptographic TCP	17
3.1 Handshake in Cryptographic TCP	17
3.2 Algorithm for handshake	19
3.3 SHA-1 Algorithm	20
3.4 State transition diagram	21
3.5 Advantages	23
3.6 Disadvantages	23
4 Implementation	25
5 Testing and Results	26
6 Conclusions	27
References	29
A	30
B	32

Chapter 1

Introduction

In the present situation the most effective attack on the TCP is the *SYN flood* attack. A *SYN flood* is a Denial of Service attack that takes advantage of the TCP three way handshake protocol. A SYN is a type of TCP packet sent to initiate a connection with a listening TCP port. The port responds with a SYN/ACK to the initiating port, and allocates resources to the new connection. When a corresponding ACK packet is received on the listening port, then the connection is established with the client.

A SYN flood occurs when one or more listening TCP ports are sent large numbers of SYN packets. Such attacks could take various forms, most of which do not adversely affect the attacked system. However, the most potentially harmful attack sends SYN packets in which the client address refers to a system which does not exist. In this case, SYN packets remain in the TCP partial connection queue for each listening port that is attacked, unable to complete because the SYN/ACK cannot be routed to a bogus address. If the queues are too small and packets awaiting response remain on the queues, the TCP stack refuses to accept any connections until the bogus packets have timed out.

1.1 What is Denial of Service attack?

A denial of service (DoS) attack is a malicious attempt by one or many users to limit or completely disable the availability of a service. They cost businesses millions of pounds each year and are a serious threat to any system or network. These costs are related to system downtime, lost revenues, and the labor involved in identifying and reacting to such attacks. DoS attacks were theorized years ago, before the mass adoption of current Internet protocols. DoS is still a major problem today and the Internet remains a fragile place.

A large number of known vulnerabilities in network software and protocols exist; meaning DoS can be achieved in a number of ways,

- Sending enough data to consume all available network bandwidth (Bandwidth Consumption)
- Sending data in such a way as to consume a resource needed by the service (Resource Starvation)
- Exercising a software *bug* causing the software running the service to fail (Programming Flaws)
- Malicious use of the Domain Name Service (DNS) and Internet routing protocols

Many DoS attacks exploit inherent weaknesses in core Internet protocols. This makes them practically impossible to prevent, since the protocols are embedded in the underlying network technology and adopted as standards worldwide. Today, even the best countermeasure software can only provide a limiting effect on the severity of an attack. An ideal solution to DoS will require changes in the security and authentication of these protocols.

In order to launch some DoS attacks, the programmer must be able to form *raw* packets. Using raw packets, the header information and data can be manipulated to form any kind of packet sequence. Hence techniques such as *IP Spoofing* and malformed ICMP Ping requests can be used.

1.2 Effects of DoS attack

It is often much easier to disrupt the operation of a network or system than to actually gain access. There are a large number of DoS attacks in existence, all targeting vulnerabilities in different services or systems. In general they attempt to shut down or seriously slow down a service provided by a computer system. The effects can range from going unnoticed to disastrous depending on the scale of the attack and the service being targeted.

With the adoption of the Supervisory Control and Data Acquisition Service (SCADA) network in the USA, the risk of a DoS attack could be catastrophic. The SCADA network is an interconnected ribbon of computer systems used for maintaining the nations infrastructure, including power, water and utilities.

1.3 Reported attacks

DoS first received large scale public attention in February 2000 [1]. Major Internet sites including CNN, Yahoo, and Amazon suffered a distributed attack over a period of several days. After several months of investigation, law enforcement officials arrested a 15-year-old Canadian youth who used the alias *Mafiaboy* [2] and charged him with perpetrating the attacks. In January 2001, the youth pleaded guilty to 56 criminal counts relating to the incident. CNN and other victims claim the attack caused damages totaling \$1.7 billion.

Since then many other attacks have been reported in the media. However it is very likely many large, well known companies do not report DoS attacks, in an effort to protect their corporate image of a secure business. The recent attack on the Al-Jazeera website [3] just a few months ago highlights the fact that DoS is still the tool of choice for many individuals.

1.4 Generic DoS attacks

We described below some widely known basic denial of service attack methods that are employed by the attack daemons.

1.4.1 Smurf attack

Smurf attack involves an attacker sending a large amount of Internet Control Message Protocol (ICMP) echo traffic to a set of Internet Protocol (IP) broadcast addresses. The ICMP echo packets are specified with a source address of the target victim (spoofed address). Most hosts on an IP network will accept ICMP echo requests and reply to them with an echo reply to the source address, in this case, the target victim. This multiplies the traffic by the number of responding hosts. On a broadcast network, there could potentially be hundreds of machines to reply to each ICMP packet. The process of using a network to elicit many responses to a single packet has been labeled as an *amplifier* [4]. There are two parties who are hurt by this type of attack: the intermediate broadcast devices (amplifiers) and the spoofed source address target (the victim). The victim is the target of a large amount of traffic that the amplifiers generate. This attack has the potential to overload an entire network.

1.4.2 SYN Flood attack

SYN Flood attack is also known as the Transmission Control Protocol (TCP) SYN attack, and is based on exploiting the standard TCP three-way handshake. The TCP three-way handshake requires a three-packet exchange to be performed before a client can officially use the service. A server, upon receiving an initial SYN (synchronize/start) request from a client, sends back a SYN/ACK (synchronize/acknowledge) packet and waits for the client to send the final ACK (acknowledge). However, it is possible to send a barrage of initial SYN's without sending the corresponding ACK's, essentially leaving the server waiting for the non-existent ACK's [5]. Considering that the server only has a limited buffer queue for new connections, SYN Flood results in the server being unable to process other incoming connections as the queue gets overloaded [6].

1.4.3 UDP Flood attack

UDP Flood attack is based on UDP echo and character generator services provided by most computers on a network. The attacker uses forged UDP packets to connect the echo service on one machine to the character generator (chargen) service on another machine. The result is that the two services consume all available network bandwidth between the machines as they exchange characters between themselves. A variation of this attack called ICMP Flood, floods a machine with ICMP packets instead of UDP packets.

1.4.4 Teardrop attack

Teardrop attack, a type of denial of service attack, exploits the way that the Internet Protocol (IP) requires a packet that is too large for the next router to handle be divided into fragments. The fragment packet identifies an offset to the beginning of the first packet that enables the entire packet to be reassembled by the receiving system. In the teardrop attack, the attacker's IP puts a confusing offset value in the second or later fragment. If the receiving operating system does not have a plan for this situation, it can cause the system to crash.

1.5 What is SYN Flood attack?

TCP SYN flooding is an instance of the flooding attacks [7]. Under this attack, the victim is a host and usually runs a Web server. A regular client opens a connection with the server by sending a TCP SYN segment. The server allocates buffer for the expected connection and replies with a TCP ACK segment. The connection remains half-open (backlogged) till the client acknowledges the ACK of the server and moves the connection to the established state. If the client does not send the ACK, the buffer will be deallocated after an expiration of a timer. The server can only have a specific number of half-open connections after which all requests will be refused. The attacker sends a TCP SYN segment pretending a desire to establish a connection and making the server reserves buffer for it. The attacker does not complete the connection. Instead, it issues more TCP SYN's, which lead the server to waste its memory and reach its limit for the backlogged connections. Sending such SYN requests with a high rate keeps the server unable to satisfy connection requests from legitimate users.

1.6 Motivation

Presently, TCP protocol has many inherent flaws which make it possible for attacker to deteriorate the performance of the server using many DoS attacks. SYN attack is one of the burning problems of the computer world. During SYN attack server's resources is exhausted and it is not in a position to serve all the requests. Our aim is to recover the server from SYN attack without much degradation of the server's performance. We aim to achieve this without using extra hardware or software with minimal changes in the existing TCP. In our implementation we use cryptographic technique to achieve our goal.

Chapter 2

Security considerations in TCP

This chapter deals with the basic concepts of the TCP protocol. We have described the TCP header format, three way handshake process, and flaws in the handshaking process.

2.1 TCP packet header format

The Transport Control Protocol (TCP) is connection oriented and reliable, in-sequence delivery transport protocol. It provides full duplex stream of data octets (8-bit bytes). It is the main protocol for the internet and is widely used in extra- and intranets. Most today's services on internet relay on TCP. For example mail (SMTP, port 25), old insecure virtual terminal service (telnet, port 23), file transport protocol (FTP, port 21) and most important in this case also is the hyper text transfer protocol (HTTP, 80) better known as the world wide web services (WWW). Almost everything uses TCP somehow to do their communications over the network - at least the interactive ones. This takes us to the point. What if these services are denied for some reason? SYN flood is one example to this. Let's look at the format of tcp packet.

The underlying IP packet contains the source IP (internet protocol) address and target IP address. To grant simultaneous access to the TCP module, TCP provides an interface called a port. Ports are used by the kernel to identify network processes from each other. They are strictly transport layer entities. Together with an IP address, a TCP port provides an endpoint for network communications. At any given moment all Internet connections can be described by four numbers. The source IP address and source port and the destination IP address and destination port completes this. Different services are bound to well-known ports so that they may be located on a standard port on different systems. [8]

TCP packet contains unique sequence and acknowledgement numbers compared to other packets in the same connection. The Initial Sequence Number (ISN) is random integer between 0 and 4,294,967,295. After each sent packet the sequence number counter is incremented by one. Data offset tells the current. This enables the right ordering of the packets if they get scrambled in the network. From the figure you can see that there is these URG, SYN, ACK, RST, PSH and FIN bits.

Source Port							
Destination Port							
Sequence Number							
Acknowledgement Number							
Header Length	Reserved	URG	ACK	PSH	RST	SYN	FIN
Window							
Checksum							
Urgent Pointer							
Options							
User Data							

Figure 2.1: TCP header

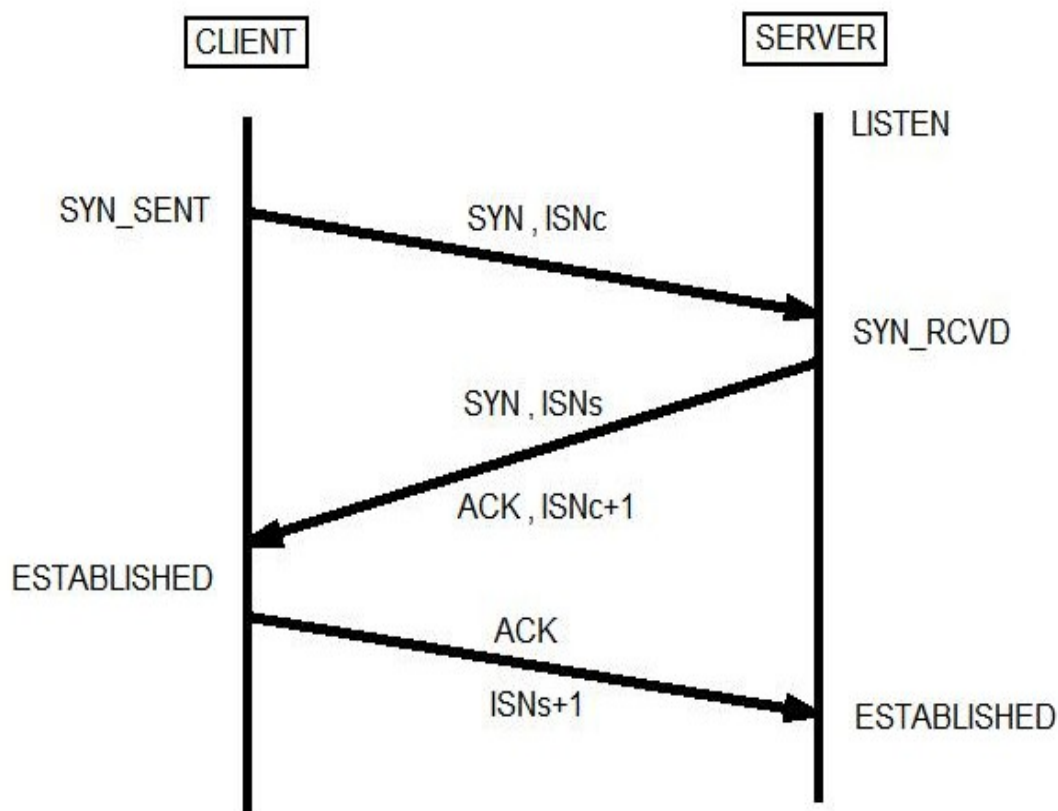
- **URG** means out of band data. For example in the telnet session if you press ctrl-c tcp stack will send a packet, which has this flag set.
- **SYN** bit has meaning only when establishing connection e.g. in the handshaking procedure. Both sides of the connection need to send this special packet with SYN flag on.
- When the **ACK** flag is on the Acknowledgement field in the tcp packet contains the number of the next acknowledgeable tcp packet with this sequence number. This bit is on almost in every packet. ACK flag tells to the target machine that the sending machine has approved all packets with sequence number below the Ack number in the packet.
- If the reset flag (**RST**) is on then the connection is destroyed and all data structures in memory for the connection must be freed.
- With interactive connections **PSH** (push) flag is used to gain rapid and smooth interaction. The packet is not queued but rather sent as soon as possible. Interactive programs should thus use this flag.

- **FIN** flag tells to the target machine that it should not take anymore data packets from the sending machine. E.g. the sending machine tells that it won't send anymore packets but can still receive packets by himself.

2.2 Three way handshake in TCP

Before any data can be transferred, a connection has to be established between the two processes. One of the processes (usually the server) issues a passive OPEN call, the other an active OPEN call. The passive OPEN call remains dormant until another process tries to connect to it by an active OPEN.

On the network, three TCP segments are exchanged (see Figure 2.2):



ISNc: Initial sequence number of client
ISNs: Initial sequence number of server

Figure 2.2: Three way handshaking

1. CLIENT picks an initial sequence number (ISNc) and sends a segment to SERVER containing: SYN_FLAG=1, ACK_FLAG=0, and SEQ=ISNc.
2. When SERVER receives the SYN, it chooses its initial sequence number (ISNs) and sends a TCP segment to CLIENT containing: SYN_FLAG=1,

ACK_FLAG=1, SEQ=ISNs, ACK=(ISNc+1). At this point SERVER allocates resources for new TCP connection.

3. When CLIENT receives SERVER's response, it acknowledges SERVER's choice of an initial sequence number by sending a dataless third segment containing: SYN_FLAG=0, ACK_FLAG=1, SEQ=ISNc+1, ACK=(ISNs+1) (data length = 0). At this point CLIENT allocates resources for the new TCP connection.
4. Data transfer may now begin.

Note: The sequence number used in SYN segments are actually part of the sequence number space. That is why the third segment that Client sends contains SEQ=(ISNc + 1). This is required so that we don't get confused by old SYNs that we have already seen. To ensure that old segments are ignored, TCP *ignores* any segments that refer to a sequence number outside of its receive window. This includes segments with the SYN bit set.

2.3 State Transition Diagram

Different states of the TCP server and client are sketched out in the Figure 2.3. The state diagram illustrates only state changes, together with the causing events and resulting actions, but addresses neither error conditions nor actions which are not connected with state changes. This diagram is only a summary and must not be taken as the total specification [11].

2.4 Weakness of the TCP protocol

There are many weaknesses in the TCP protocol and we are going to get deeper with the problem in the handshaking transaction. Consider again the handshaking. What if the machine A sends a fake packet to machine B. Fake because the source address (should be the B's IP address), is unreachable host in this network segment. Target machine B accepts the SYN packet and tries to send SYN/ACK packet to the fake address. The answer will never get to the target. Request for Comments file number 1122 (rfc1122) gives good advices:

Address Validation

- Reject OPEN call to invalid IP address x
- Reject SYN from invalid IP address x
- Silently discard SYN to bcast/mcast addr x

Here the *x* means that it **MUST** be implemented. Unfortunately as the flood is possible with fake source address (demonstrated below) this address validation schema hasn't been used (except the third one). The fact that two first advices are not followed it is possible to do SYN flood DoS attack.

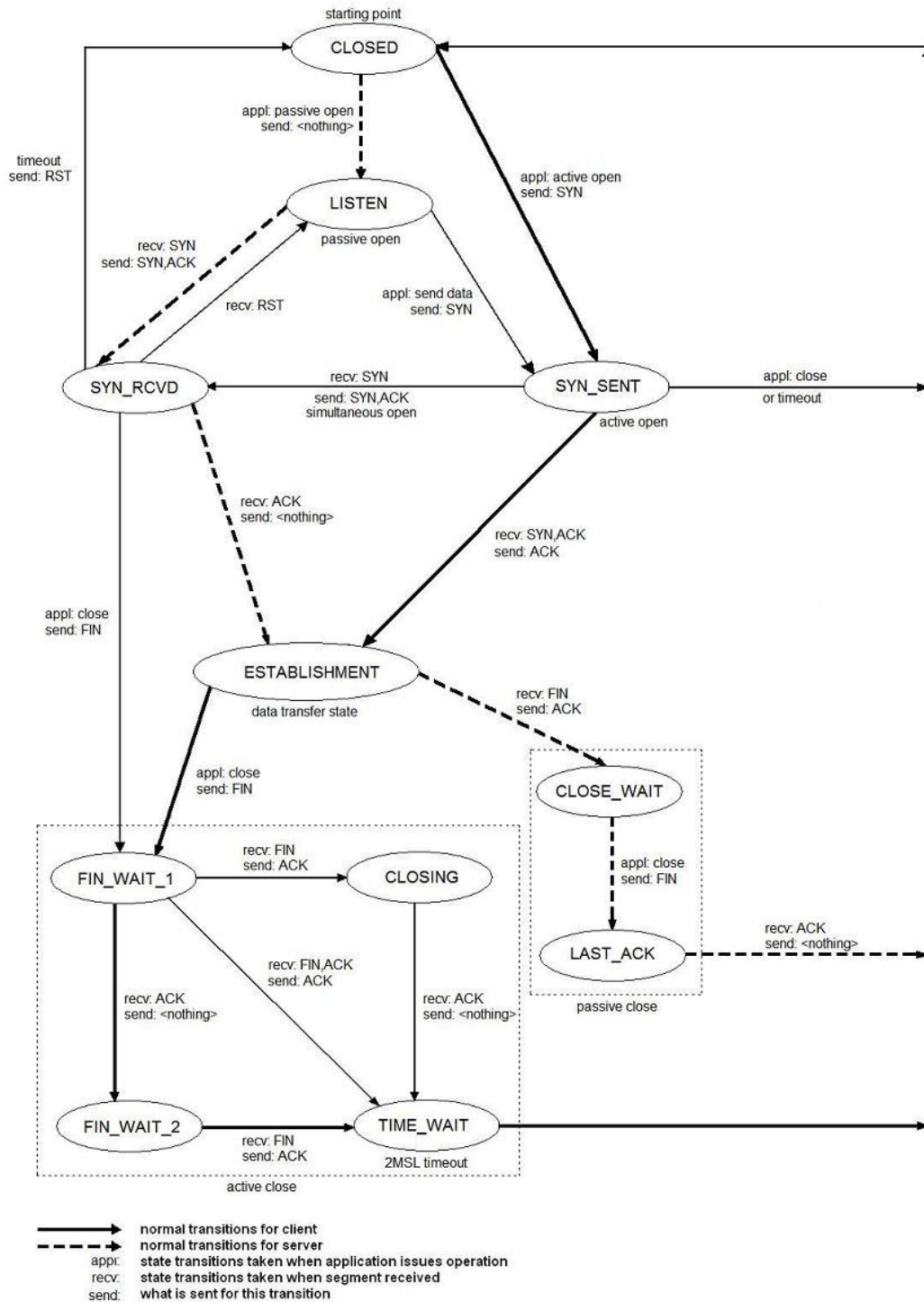


Figure 2.3: State Transition Diagram of TCP

Using a spoofed IP address not in use on the Internet, an attacker sends multiple SYN packets to the target machine. For each SYN packet received, the target machine allocates resources and sends an acknowledgement (SYN-ACK) to the source IP address.

Because the target machine doesn't receive a response from the attacking machine, it attempts to resend the SYN-ACK five times, at 3-, 6-, 12-, 24-, and 48-second intervals, before deallocating the resources 96 seconds after attempting the last retry. If you add it all together, you can see that the target machine allocates resources for more than 3 minutes to respond to just one SYN attack.

There is also quite similar DoS attack called RST flood which isn't covered in our project. By sending RST packets with correct sequence numbers (packets can be sniffed from the network) the activating (in the handshaking phase) tcp connection can be torn down quite effectively. The purpose and effect of this kind of attack is similar to syn flood DoS attack.

2.5 SYN Flood attack in TCP

Although this mechanism works for all valid TCP requests, attackers can leverage weaknesses in this system to create a DoS condition. The problem occurs due to the fact that most systems allocate a finite number of resources when setting up a 'potential' connection or a connection that has not yet been established. Although most systems can sustain hundreds of concurrent connections to the same port (e.g. port 80 for http), it may only take a dozen or so 'potential' connection requests to exhaust all resources allocated to setup a new connection. Thus SYN Flooding is a resource starvation attack.

When a SYN Flood attack is initiated the following sequence occurs:

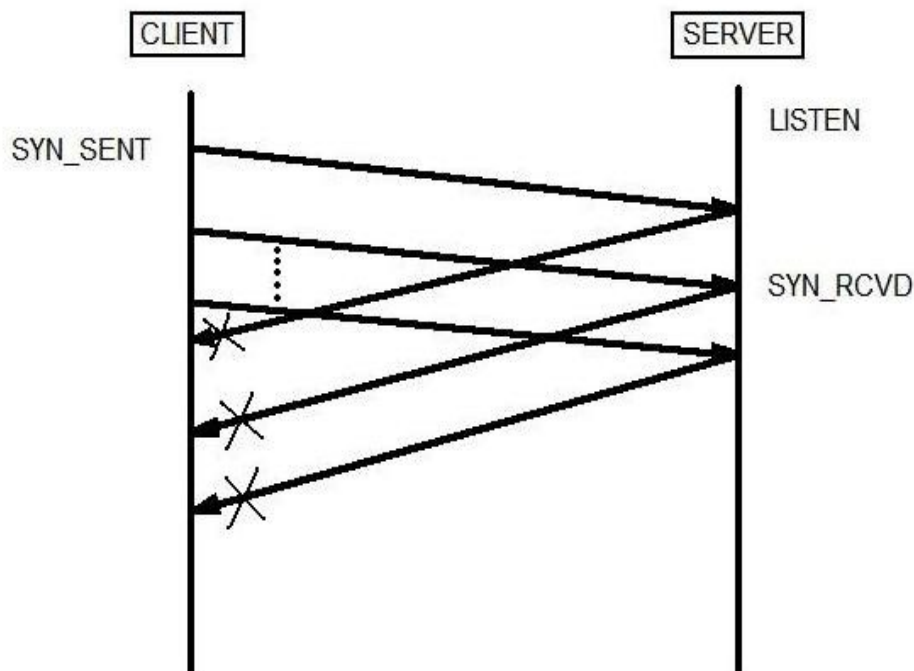


Figure 2.4: SYN Flood attack

1. CLIENT picks an initial sequence number (ISNc) and sends a segment to SERVER containing: SYN_FLAG=1, ACK_FLAG=0, and SEQ=ISNc. The client initiating the attack spoofs, or changes, the source IP address in its SYN messages to that of a currently unreachable host.
2. When SERVER receives the SYN, it chooses its initial sequence number (ISNs) and sends a TCP segment to CLIENT containing: SYN_FLAG=1, ACK_FLAG=1, SEQ=ISNs, ACK=(ISNc+1). *At this point SERVER allocates resources for new TCP connection.*
3. If the spoofed IP address is reachable then the host that receives the server's SYN-ACK will respond with a RST (Reset) message. A RST is transmitted when a host receives a packet that does not appear to be correct for the referenced connection. This would defeat the attack since a RST message from an active host owning the spoofed IP address would cause the backlogged connection to be removed from the server's pending TCP connection data structure.
4. If the spoofed IP address is unreachable then this will result in a half open connection at the server side, leading to locking up of resources for the request until time out occurs. This means that the victim server will not be able to accept legitimate connections from other clients until the pending TCP connection data structure is emptied. Depending on the server's TCP/IP implementation, a worst case scenario could result in the exhaustion of the server's memory and a potential system crash. It is important to note that in a TCP SYN Flooding attack the actual service is not damaged; only the ability to provide the service to legitimate clients is impeded.

The connection queue is small; attackers may only have to send a few spoofed SYN packets every 10secs to completely disable a port. The victim system will never be able to clear the backlog of half open connections, before receiving a new spoofed SYN packet.

This attack has the advantage (for the attacker) in that it requires very little bandwidth to disable a port. It is also a stealth attack, since the source SYN packet contains a spoofed IP address.

2.6 Severity of SYN attack

As with many other DoS attacks the SYN flood doesn't do any physical damage for the information in the machine. Nor does it make any damage to physical devices. The natures of DoS attacks are to deny something from users or other machines/processes. It may not sound very severe but when you think it more deeply it is very unpleasant effect in the growing and more commercial internet. The www browsers are used almost for every access to internet resources nowadays. WWW services are used more and more. Little by little comes the electronic cash possibilities and other growing services. Browser is the main tool in the internet communications. For example with syn flood you can deny access to the port 80 where the http server resides in a vulnerable machine. The mail server, ftp, telnet and ssh. The list is quite long.

2.7 Existing Security mechanisms

There are several existing security mechanisms which deal with SYN attack. Some of the mechanisms are,

Firewall: This is a preventive mechanism which tries to thwart attacks before they harm the system. Firewall uses filtering and trace back as the main strategy. It scans each packet which passes through the firewall. This increases packet processing time, and also overhead on the server increases. But in our solution, overhead on the server increases only when it experiences attack.

Dynamic Queue size: In the event of an attack, you should ensure that legitimate requests remain in a queue long enough to receive responses and get passed to the established connections queue. The higher the partial connection queue limit value, the less likely legitimate packets will be dropped. But the queue size is a bottleneck as it cannot be increased indefinitely. Using cryptographic technique we create a feeling of infinity size queue without actually consuming any extra resources.

SYN cookie: SYN cookies do not store any state on the machine, but keeps all state regarding the initial TCP connection in the network, treating it as an infinitely deep queue. This is done by use of a cryptographic function to encode all information into a value that is sent to the client with the SYN, ACK and returned to the server in the final portion of the 3 way handshake. In this SYN cookie mechanism, cryptographic function is used to encode information for all packets, which decreases the performance of server. But we use cryptographic function only when server experiences attack, thus saving the CPU cycles. We have increased security of the cryptographic function by using a random number in calculating hash value. SYN cookie mechanism is not able to serve other TCP options but we have an edge over this mechanism as we can serve them in Normal TCP mode (as described in section 3.1).

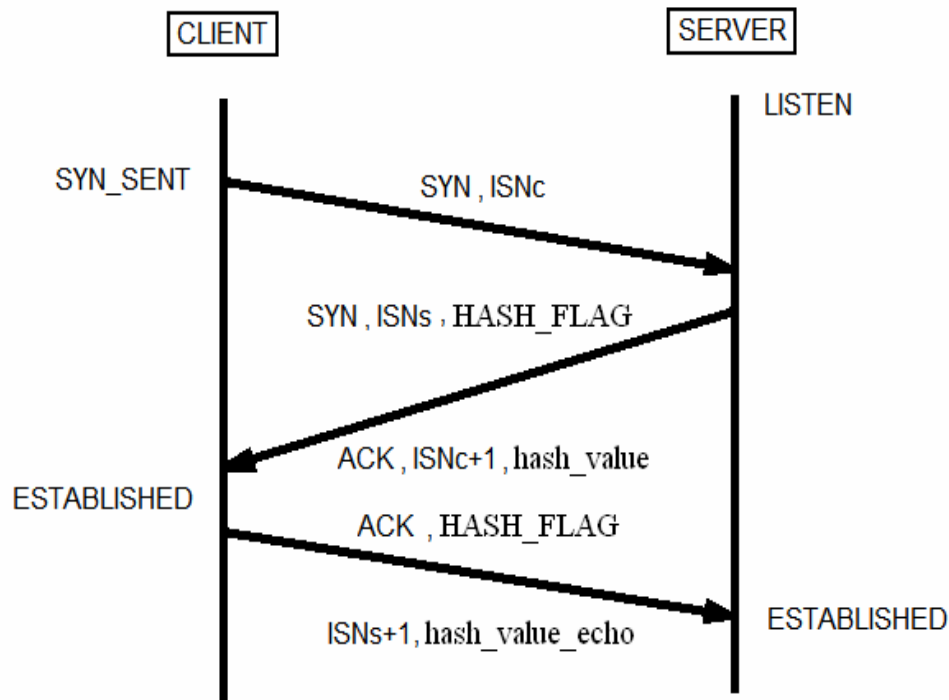
Chapter 3

Design Principles of Cryptographic TCP

In the 3-way handshaking process during TCP connection establishment, server allocates resources before the client does. But in the proposed method server allocates resources after the resource allocation by the client.

3.1 Handshake in Cryptographic TCP

Large SYN queues and random early drops make SYN flooding more expensive but don't actually solve the problem. We are using cryptographic technique to solve the problem.



ISNc: Initial sequence number of client

ISNs: Initial sequence number of server

Figure3.1: Cryptographic handshaking

In our solution we are considering two modes of handshake depending upon the load on the queue, one mode is normal TCP and another is cryptographic TCP. We switch from normal TCP mode to cryptographic TCP mode when there is no space left in queue and move back to normal TCP mode when there is any vacancy in queue. A connection which starts in any mode will continue to be in that mode itself until the connection is established.

Steps involved in cryptographic handshaking are as follows:

1. CLIENT picks an initial sequence number (ISNc) and sends a segment to SERVER containing: SYN_FLAG=1, HASH_FLAG=0, ACK_FLAG=0, and SEQ=ISNc.
2. When SERVER receives the SYN, it chooses its outgoing sequence number (ISNs) as a one way hash of the incoming information [9] and sends a TCP segment with the hash value in data part to CLIENT containing: SYN_FLAG=1, HASH_FLAG=1, ACK_FLAG=1, SEQ=ISNs, ACK=(ISNc+1). *At this point SERVER remains in LISTEN state.*

If the partial connection queue becomes full i.e. when the server is experiencing a SYN attack, we don't have to drop connections. Instead server sends back a SYN+ACK, exactly as if the SYN queue had been larger. With the difference that we create the outgoing sequence number as a one-way hash of the incoming information (IP addresses, port number, and sequence number), followed by a secret key, and a random number that changes every minute using formula,

$$\text{SHA1}(\text{secret_key}, \text{server_addr}, \text{server_port}, \text{client_addr}, \text{client_port}) + \text{ISNc} + \text{random_number}$$

Server's sequence number will be the last 32 bits of the hash value calculated by the above formula.

3. When CLIENT receives SERVER's response, it acknowledges SERVER's choice of an initial sequence number; echoing data as third segment containing: SYN_FLAG=0, HASH_FLAG=1, ACK_FLAG=1, SEQ = ISNc+1, ACK=(ISNs+1). *At this point CLIENT goes to ESTABLISHED state.*

When client receives server's SYN-ACK, first check the HASH_FLAG. If the flag is not set then it responds like normal TCP, otherwise it copies the data part to the new packet with the HASH_FLAG and ACK_FLAG set and sends it back to server.

4. When SERVER receives CLIENT's acknowledgement, it validates hash value echoed in the packet i.e. client is validated. *At this point SERVER goes to ESTABLISHED state.* Data transfer may now begin.

When the client's SYN-ACK is received we check the HASH_FLAG first. If it is not set then we will check its entry in the partial connection queue. If the entry is present

(i.e. TCB already exists) then establish the connection according to normal TCP, as this client had requested during normal TCP mode. Otherwise reject this request as a bogus client request. If the HASH_FLAG is set then we will recalculate the hash value for last few random numbers using this formula,

SHA1 (secret_key, server_addr, server_port, client_addr, client_port) + (ISNc - 1) + random_number

Compare the hash value echoed by client with the calculated hash value. If the hash value matches then the client is legitimate and a connection is established, with server changing its state from LISTEN to ESTABLISHED state. A TCB is created for this connection now.

We are taking a random number which will be changed after every 60 seconds. We will keep track of last three random numbers to calculate hash value. Instead of using random number to calculate message digest we just add to it. Advantage of this technique is that we don't have to calculate message digest again as the random number changes.

3.2 Algorithm for handshake

Client side:

Client sends a SYN packet to server.

If the client receives SYN acknowledged packet then,

 If hash flag is set then,

 create a new packet,

 set the HASH, ACK flag and echo the hash value in the data part to the server.

 else

 set the ACK flag and send the packet to server.

Server side:

If the packet has only SYN flag set then,

 If the partial connection queue has vacant space then,

 make an entry in the queue,

 create TCB for this request,

 create a new packet,

 set the SYN and ACK flag of the packet and send it to client.

 else

 calculate the hash value,

 create a new packet,

 set the SYN, HASH, ACK flag of the packet and copy the hash value in the data part and send it to client.

else

 If ACK flag is set in the packet then,

 If HASH flag is set in the packet then,

 recalculate hash value,

```

        compare it with echoed hash value,
        If hash value are same then,
            establish the connection with the client.
        else
            drop the packet.
    else
        search the entry for this request in the partial connection
        queue,
        If the entry is found then,
            establish the connection with the client.
        else
            drop the packet.

```

3.3 SHA-1 Algorithm

Pseudo code for the SHA-1 algorithm follows [10]:

Initialize variables:

```

a = h0 = 0x67452301
b = h1 = 0xEFCDAB89
c = h2 = 0x98BADCFE
d = h3 = 0x10325476
e = h4 = 0xC3D2E1F0

```

Pre-processing:

```

paddedmessage = (message) append 1
while length (paddedmessage) < 512n - 64:
    paddedmessage = paddedmessage append 0
paddedmessage = paddedmessage append (length (message) in 64-bit format)

```

Process the message in successive 512-bit chunks:

```

while 512-bit chunk(s) remain(s):
    break the current chunk into sixteen 32-bit words w (i), 0 <= i <= 15

```

Extend the sixteen 32-bit words into eighty 32-bit words:

```

for i from 16 to 79:
    w (i) = (w (i-3) xor w (i-8) xor w (i-14) xor w (i-16)) leftrotate 1

```

Main loop:

```

for i from 0 to 79:
    temp = (a leftrotate 5) + f (b, c, d) + e + k + w (i) (all addition is mod  $2^{32}$ )

```

where:

(0 ≤ i ≤ 19): f(b, c, d) = (b and c) or ((not b) and d), k = 0x5A827999

(20 ≤ i ≤ 39): f(b, c, d) = (b xor c xor d), k = 0x6ED9EBA1

(40 ≤ i ≤ 59): f(b, c, d) = (b and c) or (b and d) or (c and d), k =
0x8F1BBCDC

(60 ≤ i ≤ 79): f(b, c, d) = (b xor c xor d), k = 0xCA62C1D6

e = d

d = c

c = b leftrotate 30

b = a

a = temp

h0 = h0 + a

h1 = h1 + b

h2 = h2 + c

h3 = h3 + d

h4 = h4 + e

digest = hash = h0 append h1 append h2 append h3 append h4

Note: Instead of the formulation from FIPS PUB 180-1 shown, the following may be used for improved efficiency:

(0 ≤ i ≤ 19): f(b, c, d) = (d xor (b and (c xor d)))

(40 ≤ i ≤ 59): f(b, c, d) = (b and c) or (d and (b or c))

3.4 State Transition Diagram

The rules regarding the initiation and termination of a TCP connection implementing cryptographic handshaking can be summarized in a state transition diagram [11]. State transition diagram of Normal TCP is same as that of existing TCP. Normal TCP mode works same as existing TCP. Here we have marked the normal client transitions with a solid arrow and normal server transition with a dashed arrow.

In *Normal TCP* mode, client side and the server side are as described below (Figure 2.3):

- Towards the client side, client first sends the packet to the server with the SYN flag set and goes to the SYN_SENT state. After receiving SYN acknowledgement from the server it changes its state from SYN_SENT to ESTABLISHED.
- Towards the server side, after receiving the packet from client with the SYN flag set the server changes its state from passive open to SYN_RCVD. When it receives packet from the client with the ACK flag set it goes to ESTABLISHED state.

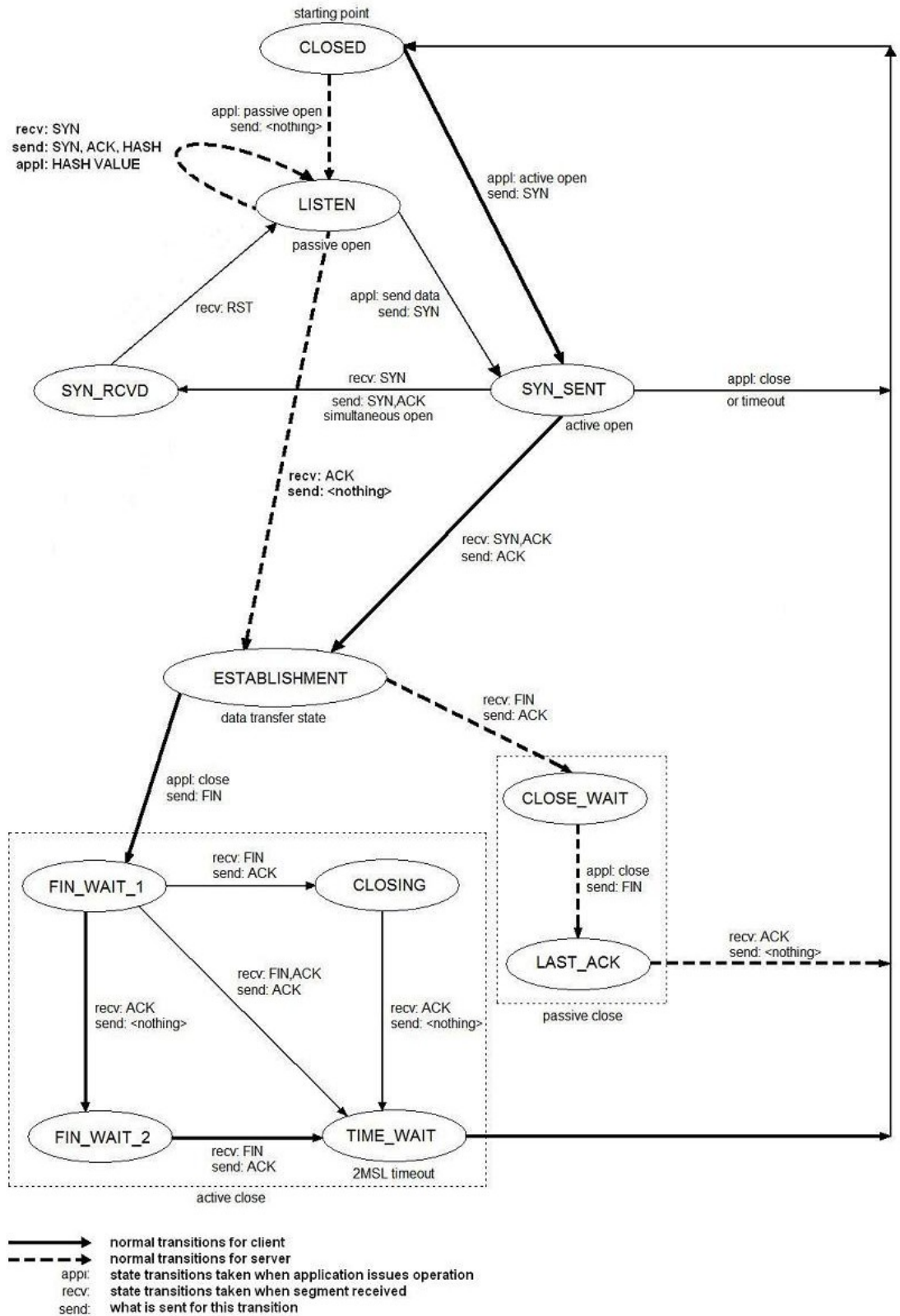


Figure 3.2: State transition diagram of Crypt TCP mode

In *Crypt TCP* mode, client and the server side are as described below (Figure 3.2):

- Towards the client side, client first sends the packet to the server with the SYN flag set and goes to the SYN_SENT state. After receiving the packet with HASH, SYN and ACK flag set from the server it changes its state from SYN_SENT to ESTABLISHED. It also echoes the data part of the packet which contains the hash value sent by the server.
- Towards the server side, after receiving the packet from client with the SYN flag set the server remains in the same LISTEN state. When it receives packet from the client with the ACK and HASH flag set it goes to ESTABLISHED state after comparing the hash value sent by the client otherwise it drops the packet.

The two transitions leading from the ESTABLISHED state are for termination of a connection. The ESTABLISHED state is where data transfer can occur between the two ends in both directions.

We've collected the four boxes in the lower left of this diagram within a dashed box and leveled it *active close*. Two other boxes (CLOSE_WAIT and LAST_ACK) are collected in a dashed box with the level *passive close*.

The state CLOSED is not really a state, but is the imaginary starting point and ending point for the diagram. The state diagram from LISTEN to SYN_SENT is legal but is not supported in Berkley-derived implementations.

3.5 Advantages

- a) It avoids SYN DoS attack on the server. In the proposed method the client is validated and made to allocate resources before the server, thus the server will allocate resources only for legitimate clients. If a client is using IP spoofing for the SYN DoS attack then server will not allocate resources unless the client is confirmed to be legitimate.
- b) There is no degradation in performance when there is no SYN attack on the server; instead it behaves as the normal TCP.
- c) Even if the queue size is less than the legitimate requests made, still our solution will serve all the requests as it is not in the case of normal TCP where requests are discarded when the queue is full.
- d) We are able to serve all kinds of TCP options (e.g. T/TCP) in Normal TCP mode.

3.6 Disadvantages

- a) The performance of the server decreases when it is under SYN attack, because we need to calculate hash value of the client state which takes more CPU time. But recent processors have higher speed thus making the CPU time, taken for calculating hash value, negligible as compared to network speed.

- b) In our solution we cannot serve other TCP options when we are in Crypt TCP mode. But we can overcome this problem by reserving some percentage of the queue for it.
- c) When SYN, ACK arrives at a client but the return ACK is lost, this result in a disparity about the established state between the client and server. Resulting in client waiting for server's response.

Chapter 4

Implementation

We have implemented the project in ns-2 (network simulator) [12]. The output is shown graphically considering client server architecture in network animator (nam). We have used Tcl language in writing code for network instance and C++ for the core simulator.

Network Simulator (ns) is an object oriented simulator, written in C++, with an OTcl interpreter as a front end. The simulator supports a class hierarchy in C++ (also called the compiled hierarchy), and a similar class hierarchy within the OTcl interpreter (also called the interpreted hierarchy). The two hierarchies are closely related to each other; from the user's perspective, there is a one-to-one correspondence between a class in the interpreted hierarchy and one in the compiled hierarchy. *ns* uses two languages, C++ and OTcl because C++ is fast to run but slower to change, making it suitable for detailed protocol implementation. OTcl runs much slower but can be changed very quickly (and interactively), making it ideal for simulation configuration. *ns* (via tclcl) provides glue to make objects and variables appear on both languages.

In *ns* simulator Full TCP considers handshaking during connection setup. Inheriting Full TCP we have made three new agents [13]:

- a) **Original TCP Agent:** In Original TCP we have implemented partial connection queue which was not present in Full TCP. Thus we are able to show connection drop during the handshake process. In Full TCP there is no provision for multi-client connection but in Original TCP we have simulated multi-client connection using two agents for each connection.
- b) **Crypt TCP Agent:** In Crypt TCP we are using a reserved bit from the tcp header for the Hash flag. This flag is set whenever the handshake is done in cryptographic mode; server will calculate hash value using SHA1 hash algorithm and handshake will proceed as described in section 3.1. Both the modes of handshake are implemented in this agent.
- c) **Attacker TCP Agent:** In Attacker TCP we have implemented client side in such a way that it continues sending SYN packets and will drop all incoming packets.

Chapter 5

Testing and Results

In our project we have shown the results for two test cases for each test case we are comparing the performance of TCP and Cryptographic TCP.

- a) **Case 1:** Some clients are trying to make Tcp connection on the server out of which one is an attacker, which is continuously bombarding SYN packets on the server; while other clients are legitimate clients. Partial connection queue size is greater than the number of clients requesting for the connection at a particular moment.

In *TCP*, some of the clients are denied from tcp connection with the server. Attacker is sending SYN packets continuously to the server which fills up the partial connection queue, giving less chance for legitimate clients to make a connection. This situation is shown in Appendix A.1.

In *Cryptographic TCP*, number of connections made by the server at an instant is not restricted by the size of the partial connection queue. If the queue becomes full then we change the mode from the *Normal Tcp* to *Crypt Tcp*. Any requests coming in a mode are served in the same mode itself. In Normal Tcp mode the requests for tcp connections are served in the same manner as they are done in the existing Tcp. Any request coming when the queue is full are served under Crypt Tcp mode, in this mode the request, instead of being dropped as in the existing Tcp are acknowledged by the server as described in the section 3.1. Thus in our proposed solution all the legitimate clients are able to make the connection and are not affected by the presence of attacker. This situation is shown in Appendix A.2.

- b) **Case 2:** All the clients are legitimate and are trying to establish tcp connection with the server at the same instant. Partial connection queue size on the server side is less than number of client's request for tcp connection at an instant.

In *TCP*, all the requests coming after the queue is full are dropped. Thus all the requests are not served. This situation is shown in Appendix B.1.

In *Cryptographic TCP*, as number of connections made by the server at an instant is not restricted by the size of the partial connection queue, thus all the client's request are successfully served. This situation is shown in Appendix B.2.

Performance of server

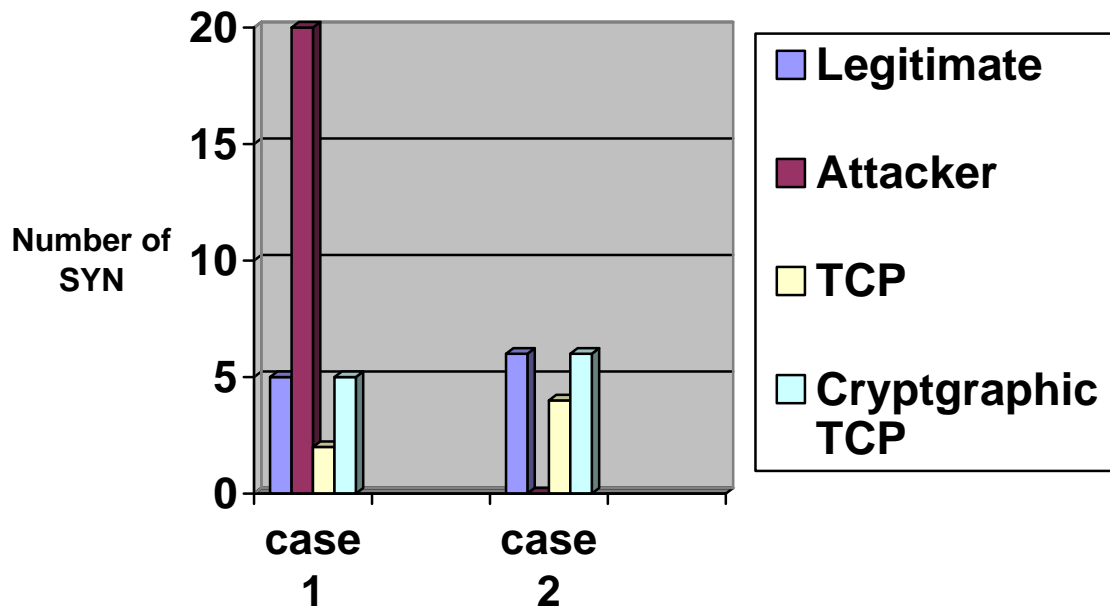


Figure 5.1: Bar chart showing performance of server

The results of the implementation are shown in the bar chart. Performance of one sample server is compared between TCP and Cryptographic TCP with the partial connection queue size of four.

Chapter 6

Conclusions

Denial of service attack has become increasingly popular due to easy accessibility, and the fact that little programming knowledge is required to launch an attack. These attacks are among the most vicious because they quickly consume all network resources on even the largest hosts, rendering them useless. We have studied various aspects of SYN attack and its effects on the server. This attack exploits a small flaw in the TCP protocol, which can be overcome using authentication of the client. SYN attack has been simulated on the server under various conditions and network topologies. SYN attack is simulated on the modified server, which uses our cryptographic technique (SHA-1 one way hash algorithm) of authenticating the client before giving service. Thus server is able to overcome the SYN attack. The other existing method for combating with SYN flood attack is SCTP implementation. But it calculates hash value for each SYN packet whether it is attacked or not. Rather in our solution we calculate hash value only when partial connection queue becomes full. Thus load on CPU in our solution is far less than that in SCTP implementation.

Bibliography

- [1] CNN.com, "The denial of service aftermath, Feb 2000,"
<http://www.cnn.com/2000/TECH/computing/02/14/dos.aftermath.idg/index.html>
- [2] Philip Preville, The Montreal Mirror Online Archive, "On the trail of Mafiaboy,"
Jun 2000, <http://www.montrealmirror.com/ARCHIVES/2000/022400/news1.html>
- [3] Paul Roberts, IDG News Service, "Al-Jazeera Sites Hit With Denial-of-Service
Attacks," Mar 2003, <http://www.nwfusion.com/news/2003/0326aljahobbl.html>
- [4] C. A. Huegen, "The latest in denial of service attacks: 'Smurfing' description
and information to minimize effects," Feb. 2000,
<http://users.quadranner.com/chuegen/smurf.cgi>.
- [5] S. Bellovin, "Security problems in the TCP/IP protocol suite," *Comput. Commun.
Rev.*, vol. 19, no. 2, pp. 32-48, Apr. 1989.
- [6] Cisco Systems, Inc., "Defining strategies to protect against TCP SYN denial of
service attacks," July 1999, <http://www.cisco.com/warp/public/707/4.html>.
- [7] Ahsan Habib, Mohamed M. Hefeeda, and Bharat K. Bhargava., "Detecting Service
Violations and DoS Attacks," CERIAS and Department of Computer Sciences,
Purdue University, West Lafayette, 2003.
- [8] Postel, J., "Transmission Control Protocol", RFC 793, USC/Information Sciences
Institute, September 1981.
- [9] <http://cr.yp.to/syncookies/idea>
- [10] <http://en.wikipedia.org/wiki/Pseudocode>
- [11] W.R. Stevens, "TCP/IP Illustrated," Volume I, Pearson Education Asia, 2001.
- [12] <http://www.isi.edu/nsnam/ns/>
- [13] <http://nile.wpi.edu/NS/>

APPENDIX A

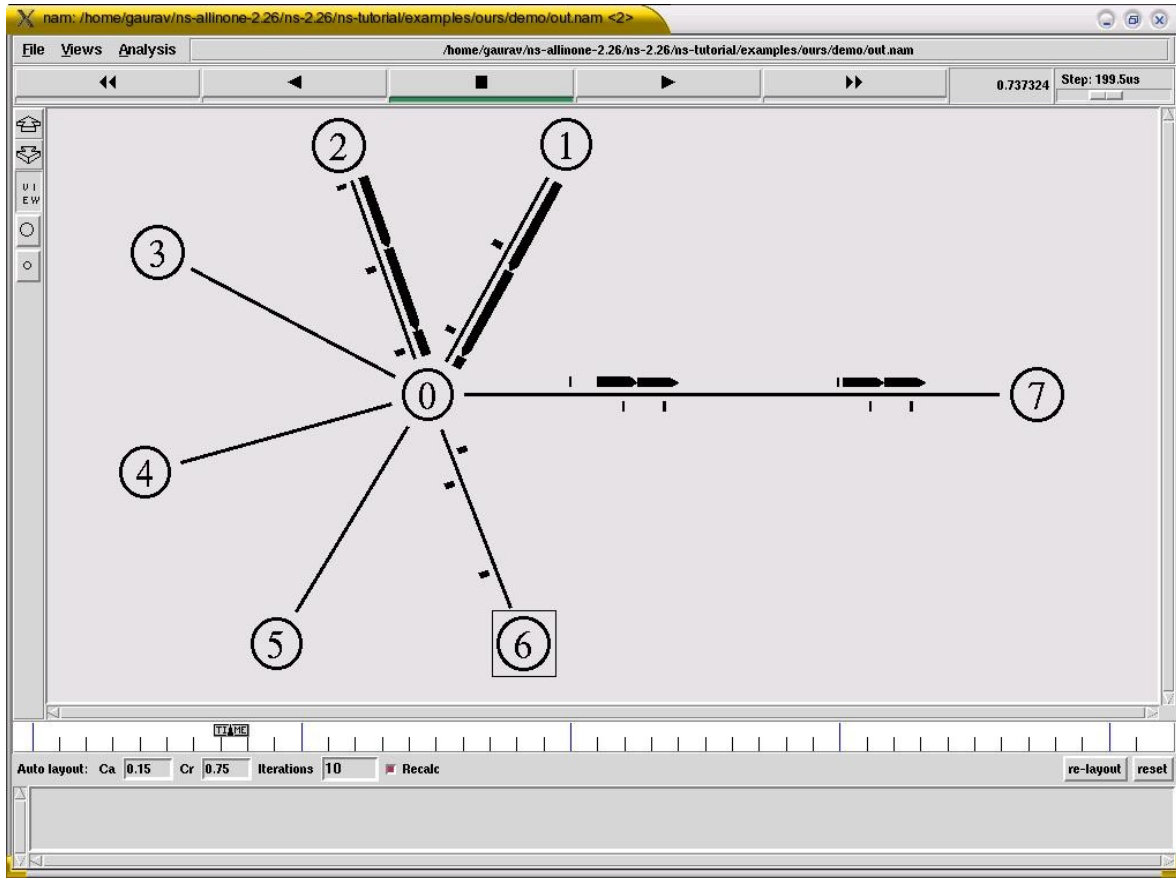


Figure A.1: Attack on TCP server

In Figure A.1 nodes 1, 2, 3, 4 and 5 represent legitimate clients. Node 6 represents attacker and node 7 represents server. Due to partial connection queue size of 4 in this example, only client 1 and client 2 are successful to make a connection with the server. When other clients send SYN packets, they are dropped because the queue is full with the attacker's SYN packets. This is the case with existing TCP.

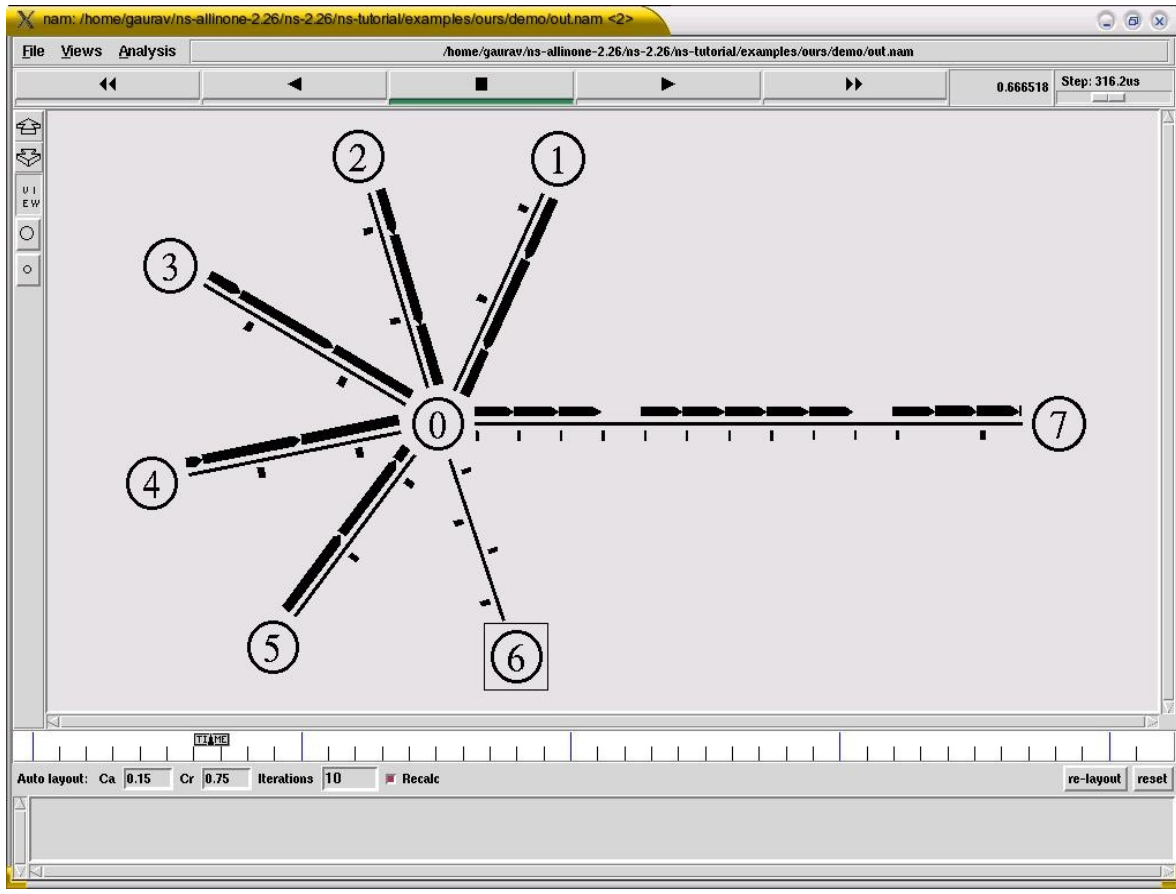


Figure A.2: Attack on Cryptographic TCP server

In Figure A.2, nodes 1, 2, 3, 4 and 5 represent legitimate clients. Node 6 represents attacker and node 7 represents server. Here partial connection queue size is 4 but when the queue becomes full by the SYN packets from the attacker, switching from Normal TCP mode to Crypt TCP mode takes place. And all clients are able to make a connection with the server.

APPENDIX B

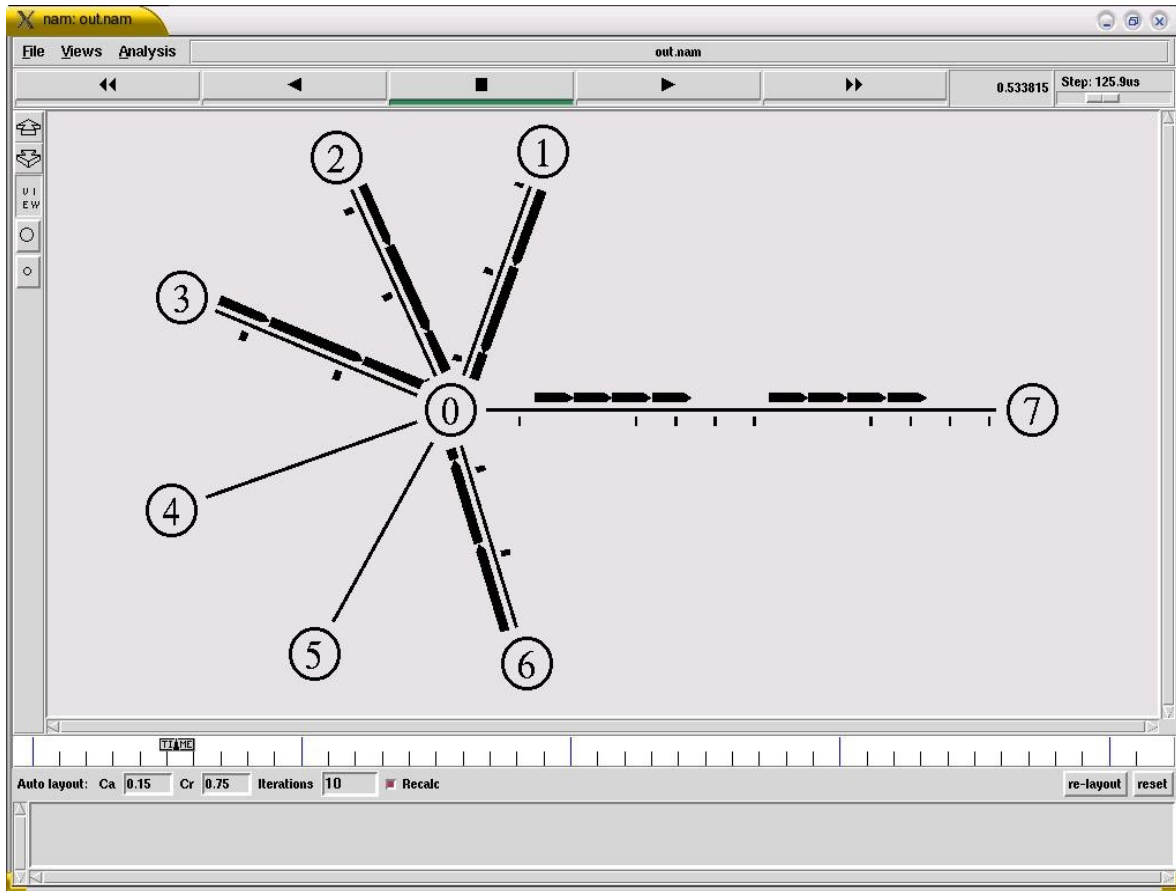


Figure B.1: Overload on TCP server

In Figure B.1 nodes 1, 2, 3, 4, 5 and 6 represent legitimate clients and node 7 represent server. Size of the partial connection queue is 4, thus when all try to make tcp connection with the server at the same instant only 4 are able to get service. Rests of the SYN packet which reach server when the partial connection queue becomes full are dropped out. This is the case with existing TCP.

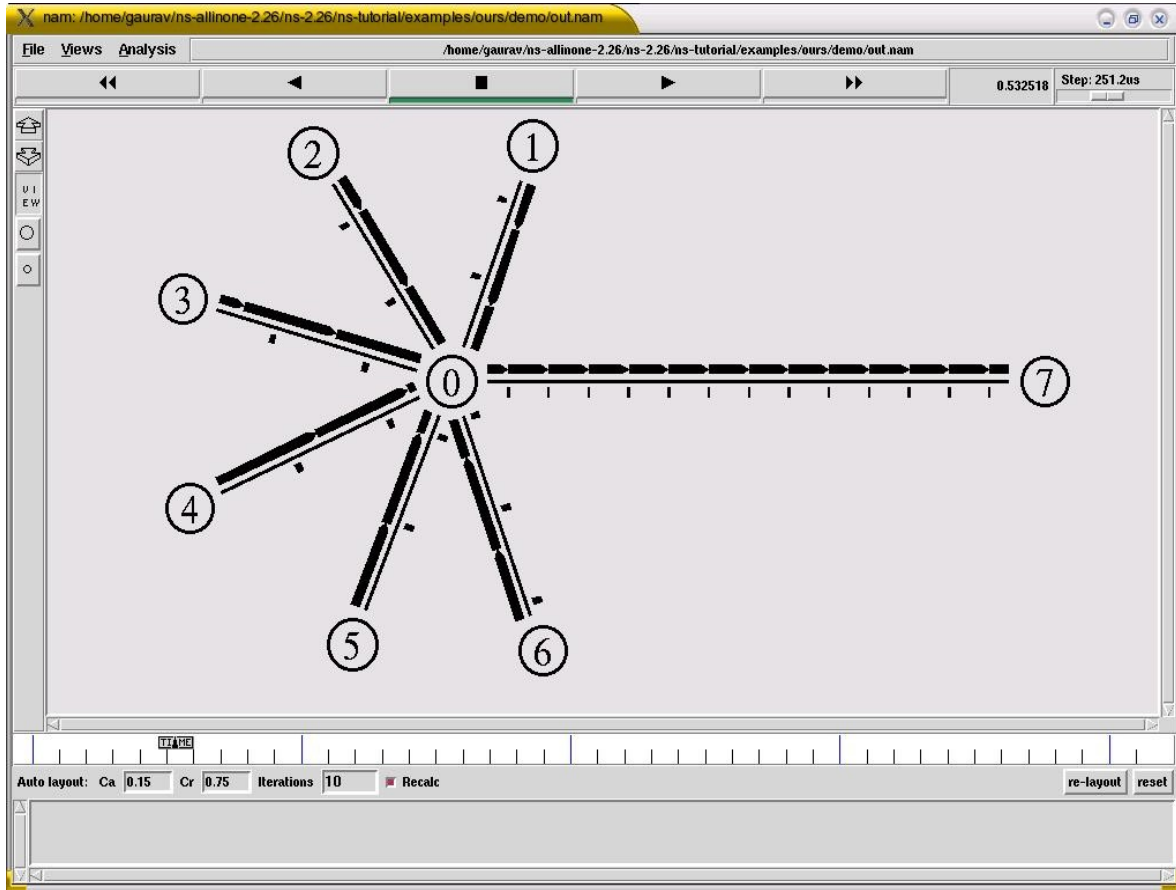


Figure B.2: Overload on Cryptographic server

In Figure B.2 node 1, 2, 3, 4, 5 and 6 represent legitimate clients and node 7 represent server. Size of the partial connection queue is 4, thus when all try to make TCP connection with the server at the same instant, all are able to get service. SYN packets which reach server when the partial connection queue becomes full are served in Crypt TCP mode.