Denial of Service Attacks (DoS)

Denial of Service (DoS) Attacks

Goal: take large service/network/org offline by overwhelming it with network traffic such that they can't process real requests

How: find mechanism where attacker doesn't spend a lot of effort, but requests are difficult/expensive for victim to process

Types of Attacks

DoS Bug: design flaw that allows one machine to disrupt a service. Generally a protocol asymmetry, e.g., easy to send request, difficult to create response. Or requires server state.

DoS Flood: control a large number of requests from a botnet or other machines you control

DoS Opportunities at Every Layer

Link Layer: send too much traffic for switches/routers to handle

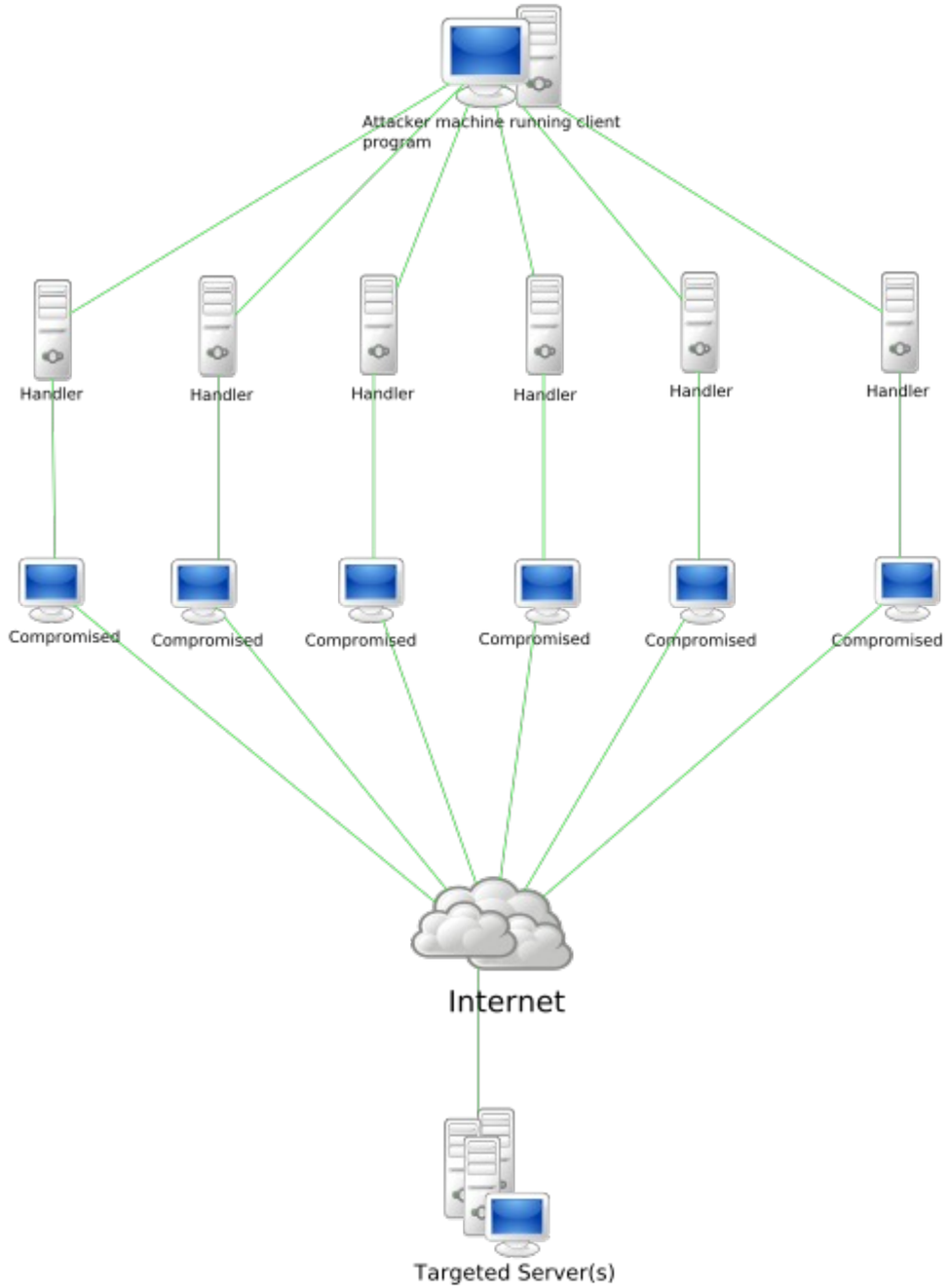
TCP/UDP: require servers to maintain large number of concurrent connections or state

Application Layer: require servers to perform expensive queries or cryptographic operations

Distributed DoS (DDos)

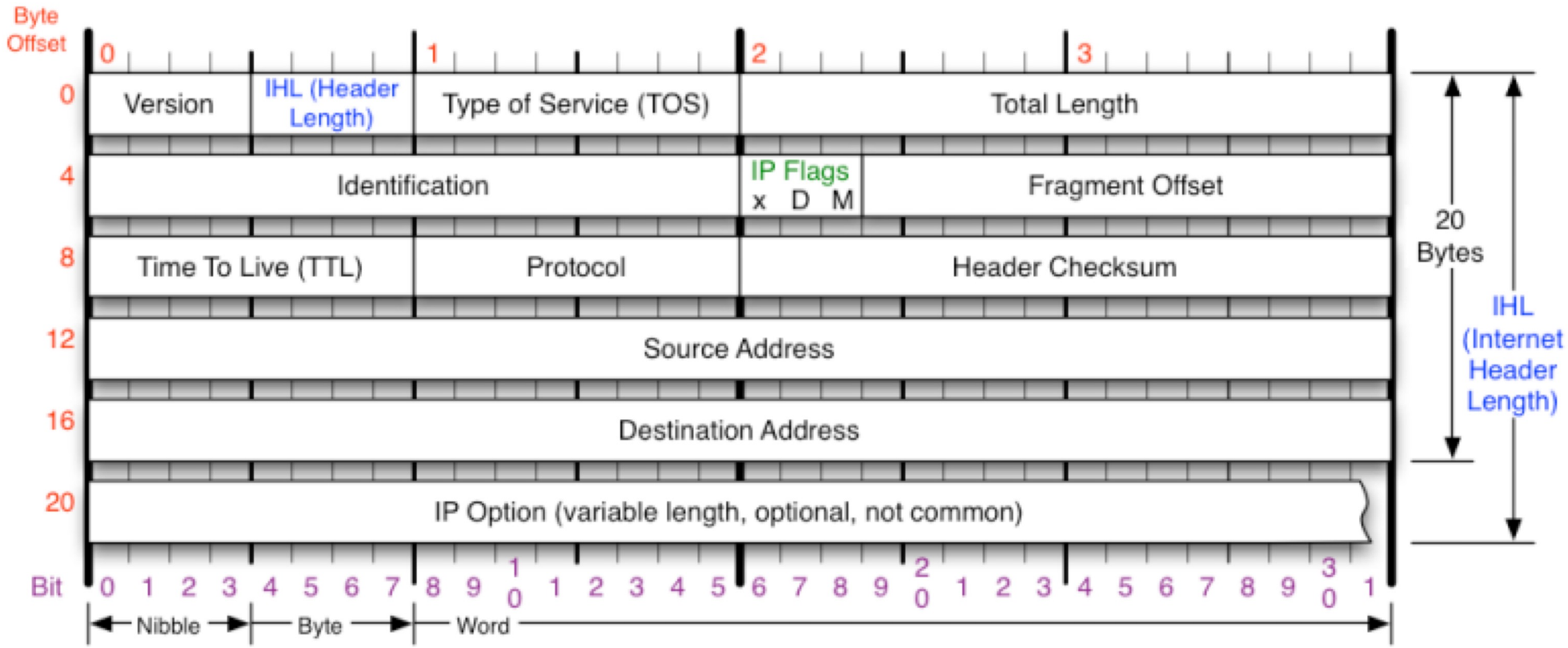
Incoming traffic flooding the victim originates from **many** different sources

Botnet


Attacker machine running client program
Handler
Handler
Handler
Handler
Handler
Handler
Compromised
Compromised
Compromised
Compromised
Compromised
Compromised
Internet
Targeted Server(s)

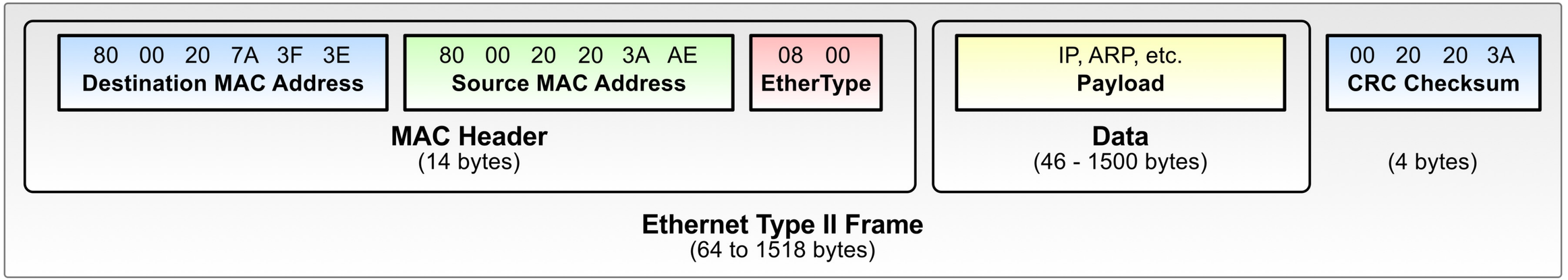
IPv4 Header

Instruct routers and hosts what to do with a packet
All values are filled in by the sending host



Ethernet

Most common Link Layer Protocol. Let's you send packets to other local hosts.



At layer 2 (link layer) packets are called *frames*

MAC addresses: 6 bytes, universally unique

EtherType gives layer 3 protocol in payload
- 0x0800: IPv4
- 0x0806: ARP
- 0x86DD: IPv6

From Packets to Streams

Most applications want a stream of bytes delivered reliably and in-order between applications on different hosts

Transmission Control Protocol (TCP) provides…

- Connection-oriented protocol with explicit setup/teardown
- Reliable in-order byte stream
- Congestion control

Despite IP packets being dropped, re-ordered, and duplicated

TCP Sequence Numbers

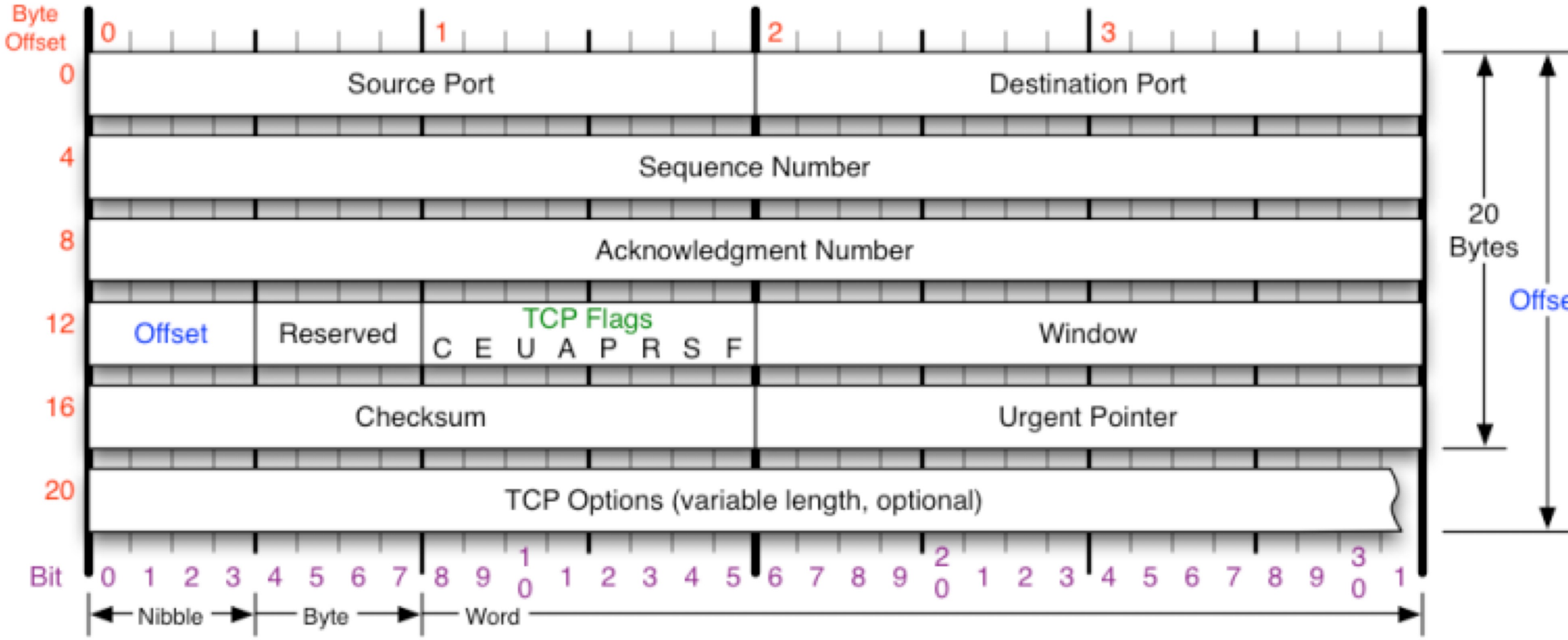
Two data streams in a TCP session, one in each direction

Bytes in data stream numbered with a 32-bit *sequence number*

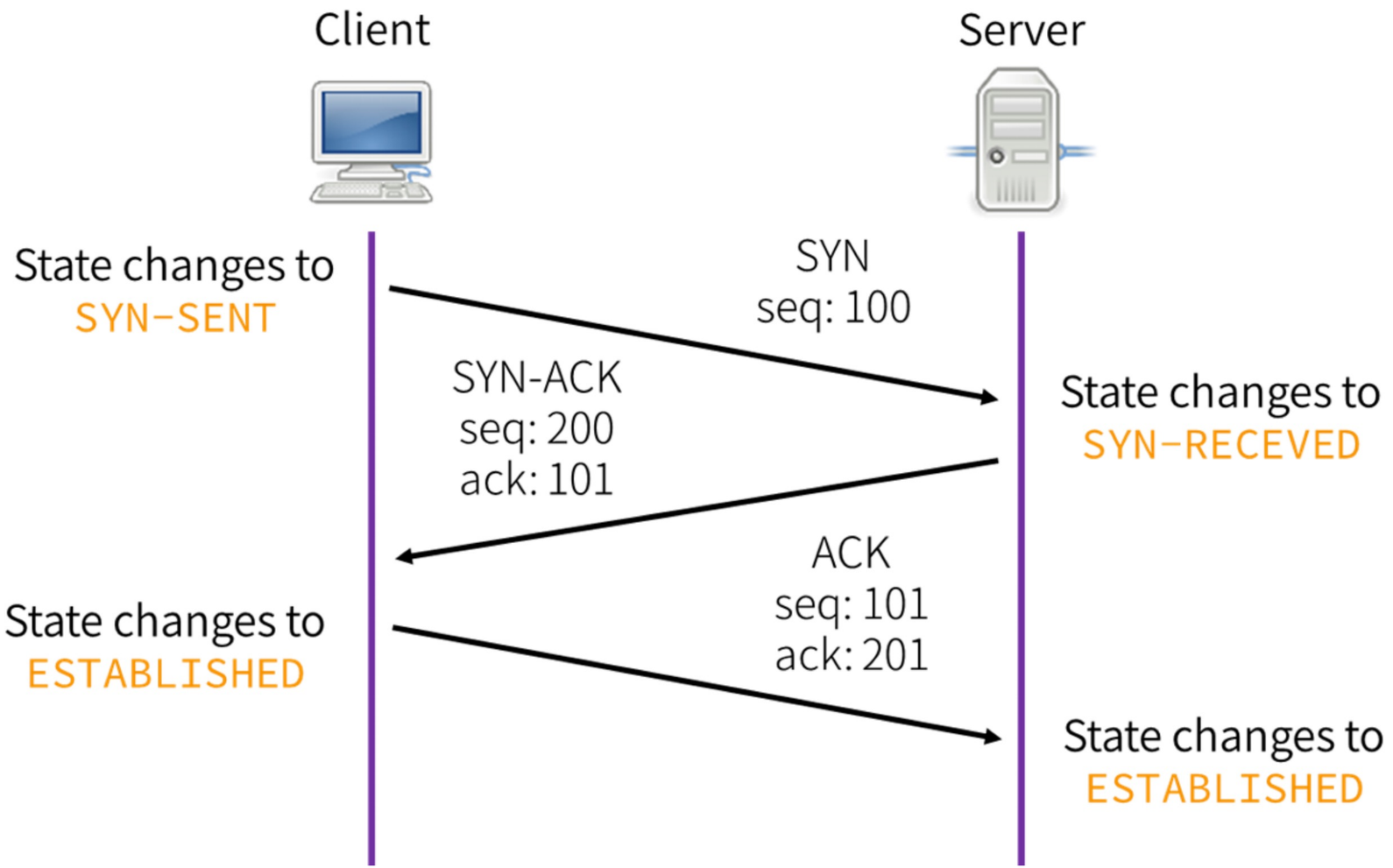
Every packet has sequence number that indicates where data belongs

Receiver sends acknowledgement number that indicates data received

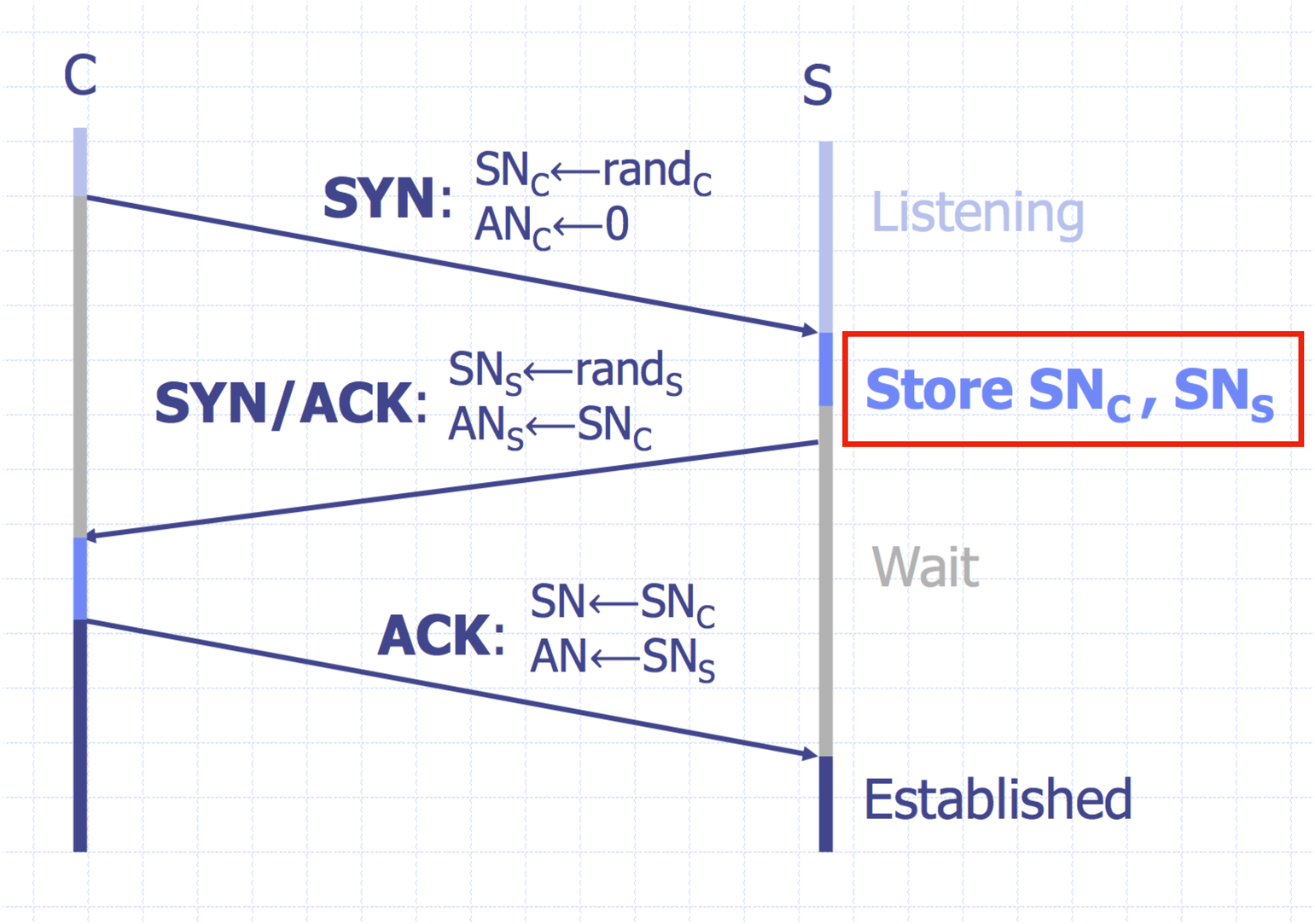
TCP Packet


Byte Offset
0
1
2
3
0
Source Port
Destination Port
4
Sequence Number
8
Acknowledgment Number
12
Offset
Reserved
TCP Flags
C E U A P R S F
Window
16
Checksum
Urgent Pointer
20
TCP Options (variable length, optional)
20 Bytes
Offse
Bit
0 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 20 1 2 3 4 5 6 7 8 9 30 1
Nibble
Byte
Word

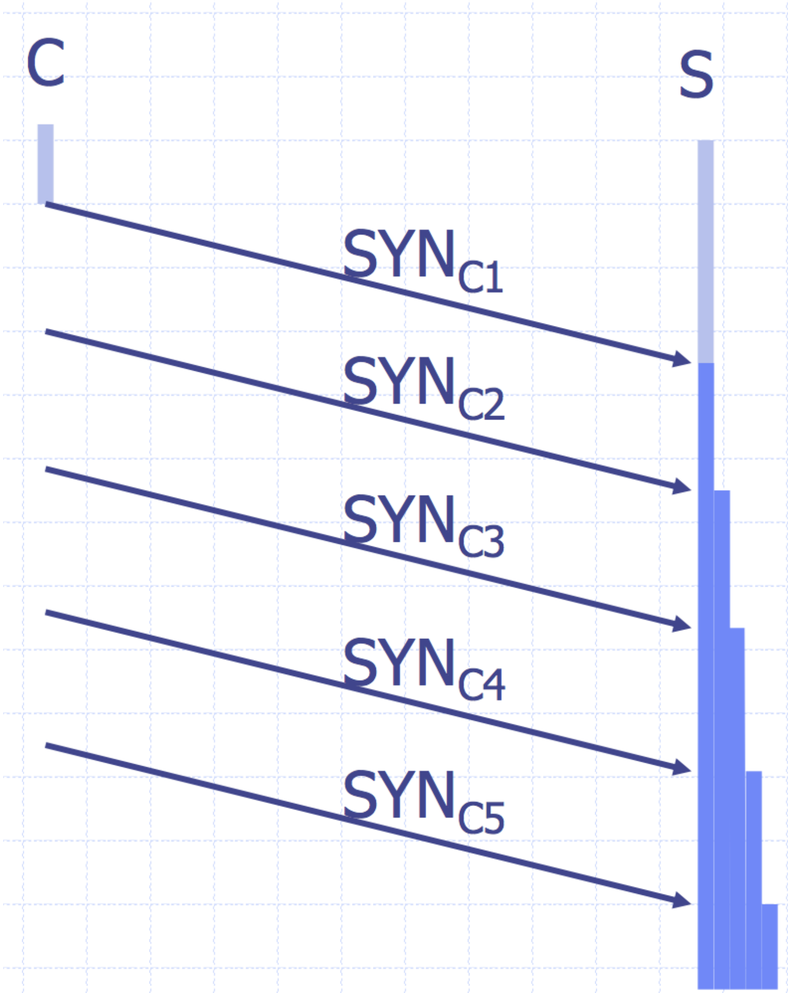
TCP Three Way Handshake



TCP Handshake



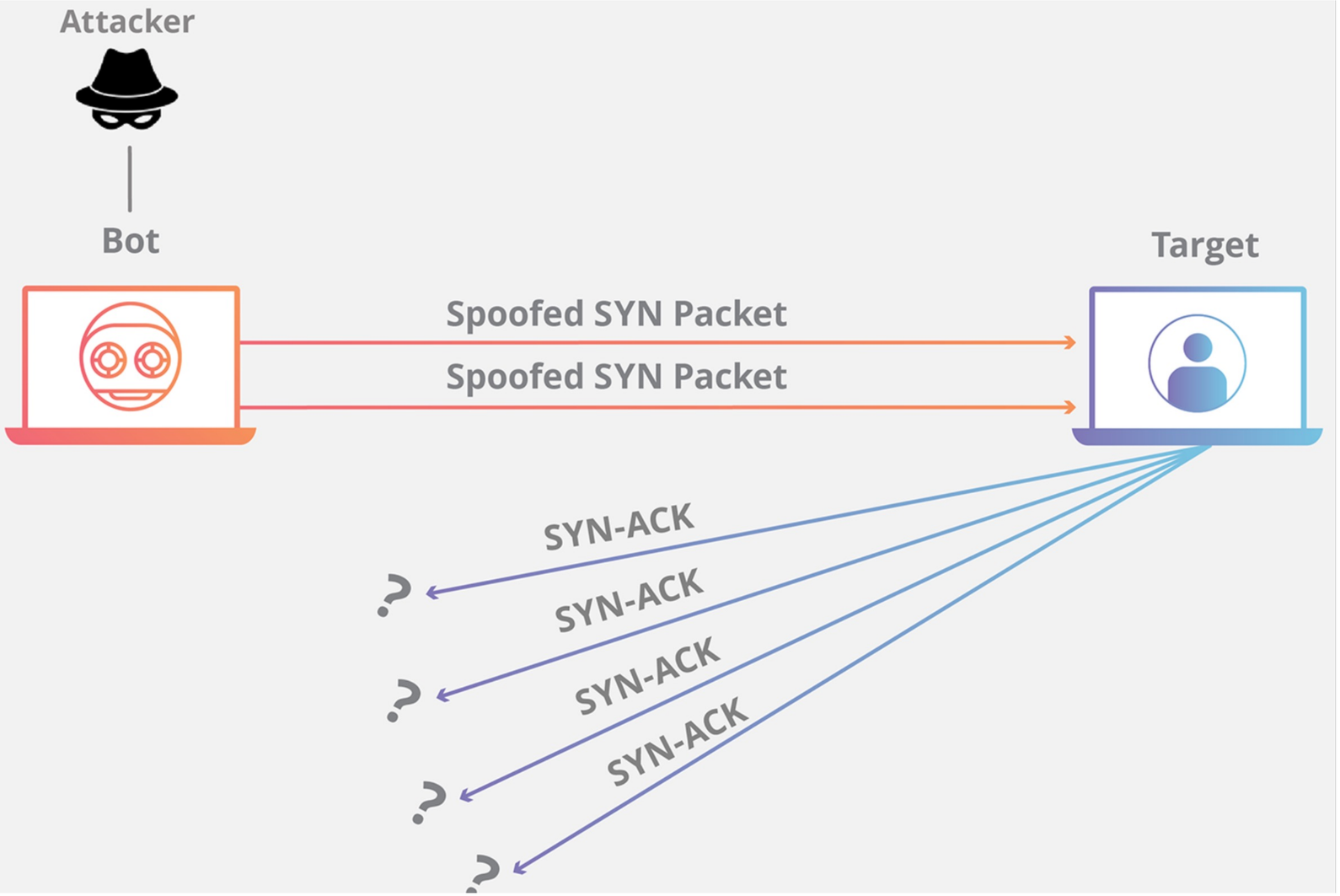
SYN Floods



Single machine:

- SYN Packets with **random source IP addresses**
- Fills up backlog queue on server
- No further connections possible

SYN Floods


Attacker
Bot
Target
Spoofed SYN Packet
Spoofed SYN Packet
SYN-ACK
SYN-ACK
SYN-ACK
SYN-ACK
?
?
?
?

Core Problem

Problem: server commits resources (memory) before confirming identify of the client (when client responds)

Bad Solution:

- Increase backlog queue size
- Decrease timeout

Real Solution: Avoid state until 3-way handshake completes

SYN Cookies

Idea: Instead of storing SN_C and SN_S…
send a cookie back to the client.

$L = MAC_{key}(SAddr, SPort, DAddr, DPort, SN_C, T)$

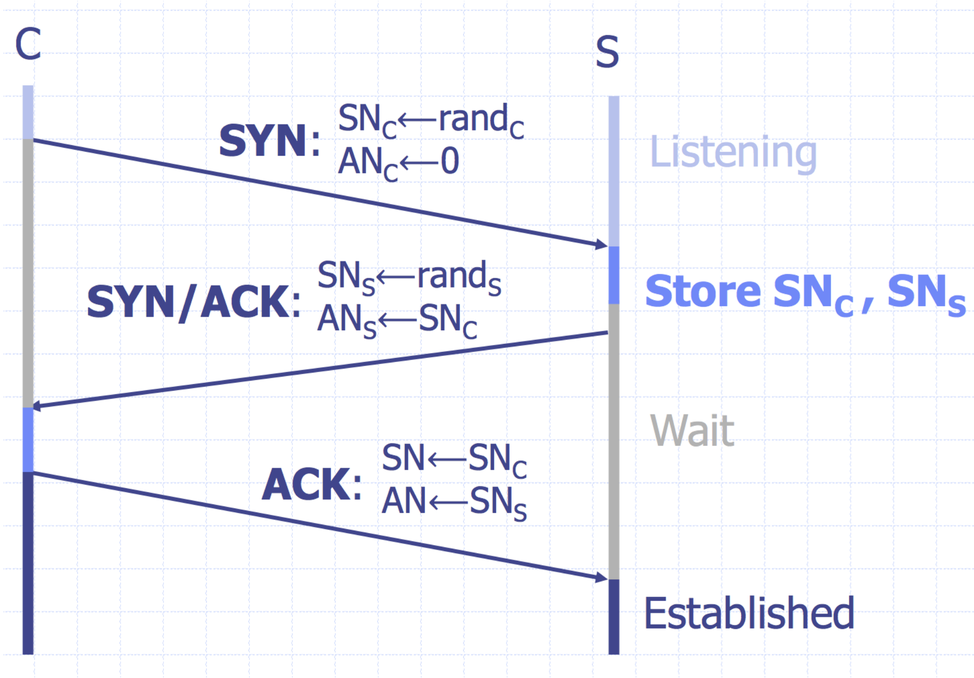
key: picked at random during boot

T = 5-bit counter incremented every 64 secs.

$SN_S = (T \| mss \| L)$

Honest client sends ACK ($AN=SN_S$, $SN=SN_C+1$)

Server allocates space for socket only if valid SNs



Server does not save state
(loses TCP options)

Moving Up Stack: GET Floods

Command bot army to:

* Complete real TCP connection
* Complete TLS Handshake
* GET large image or other content

Will bypass flood protections…. but attacker can no longer use random source IPs

Victim site can block or rate limit bots