

TCP/IP Overview

CYBR371: System and Network Security, (2024/T1)

Arman Khouzani, Mohammad Nekooei
Slides modified from "Masood Mansoori"

13 March, 2024

Victoria University of Wellington – School of Engineering and Computer Science

TCP/IP Basics



How Internet works

Protocol: Agreement on how to communicate.

Internet is based on the **TCP/IP** protocol.

Transmission Control Protocol / Internet Protocol (TCP/IP) is a suite of many protocols for transmitting information on a network.

- Often referred to as a “protocol stack”.

OSI and TCP/IP Models

OSI reference model: divides the communication functions used by two hosts into seven separate layers.

TCP/IP has its own stack of protocols.

OSI Layers	TCP/IP Layers
Application	Application
Presentation	
Session	
Transport	Transport
Network	Internet
Data Link	
Physical	Network Interface

Note: Direct or strict comparisons of the OSI and TCP/IP models should be avoided, because the layering in TCP/IP is not a principal design criterion.

TCP/IP Protocols

	TCP/IP layers	TCP/IP Protocols
Application Specific Semantics	Application	HTTP, DNS, BGP, NTP, SMTP, IMAP, FTP, NFS, SNMP
E2E communication between processes; Adds ports/reliability	Transport	TCP, UDP
Adds global addresses; Requires routing	Network	IP, ICMP
Adds framing & destination; Still assumes shared link. Broadcasts on shared link	Network Interface	Ethernet, 802.11 (wifi), High-Level Data Link Control (HDLC), Asynchronous Transfer Mode (ATM), PPP

- THE NETWORK IS DUMB.
- **End-hosts** are the periphery (users, devices).
- **Routers** and **switches** are Intermediary devices that:
 - Route (figure out where to forward).
 - Forward (actually send).

Principles of IP:

- The routers **have no knowledge of ongoing connections** through them.
- They do “**destination-based**” routing and forwarding.
 - Given the destination IP address in the packet, send it to the “next hop” that is best suited to help ultimately get the packet there.

Types of Addresses in Internet

- **Media Access Control** (MAC) addresses in the network access layer.
 - Associated with network interface card (NIC).
 - 48 bits or 64 bits.
- **IP addresses** for the network layer.
 - 32 bits for IPv4, and 128 bits for IPv6.
 - e.g., **128.3.23.3**
- **IP addresses + ports** for the transport layer.
 - e.g., **128.3.23.3:80**
- **Domain names** for the application/human layer.
 - e.g., **ecs.wgtn.ac.nz**

Routing and Translation of Addresses

Translation between **IP addresses and MAC** addresses.

- Address Resolution Protocol (**ARP**) for IPv4.
- Neighbour Discovery Protocol (**NDP**) for IPv6.

Routing with IP addresses

- TCP, UDP, IP for routing packets, connections.
- Border Gateway Protocol (**BGP**) for routing table updates.

Translation between **IP addresses and domain names**

- Domain Name System (**DNS**).

Types of Addresses in Internet

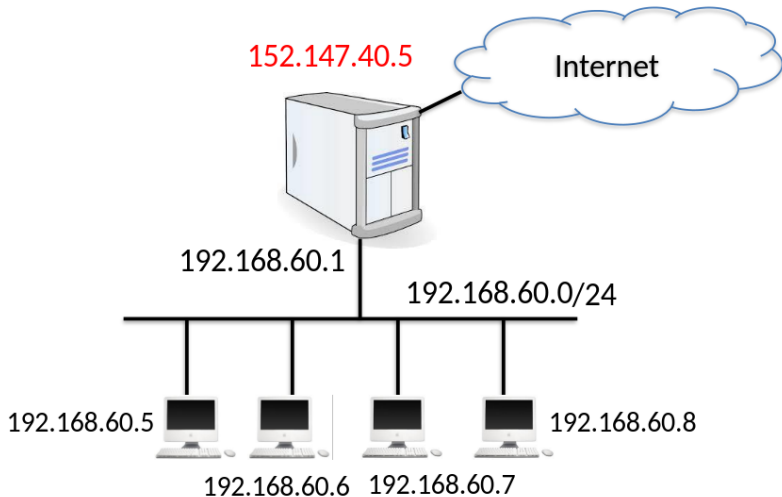
Private IP address

- Private addresses are not routable over the internet
 - **10.0.0.0/8 (10.x.x.x)**
 - **172.16.0.0/12 (172.16.x.x)**
 - **192.168.0.0/16 (192.168.x.x)**
- ▶ **NAT (Network Address Translation)**: the process of replacing a private IP address to a public IP address and vice-versa.

Loopback Address

- **localhost, Interface lo**
 - **127.0.0.0/8 (127.x.x.x)**
 - Commonly used **127.0.0.1**

Network Address Translation (NAT)

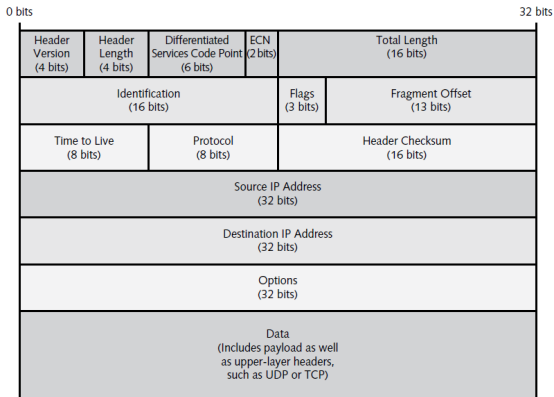


Networking

Data is transmitted in *small chunks*.

- At Level 3 these chunks are called **packets**.
- At Level 2 these chunks are called **dataframe**.

A packet/frame has 2 primary subdivisions: Header and Data.

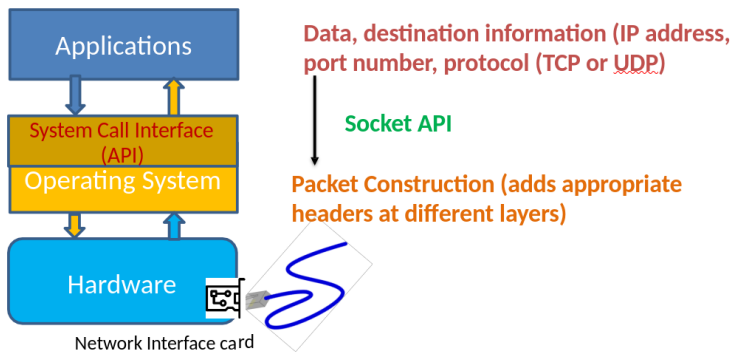


- Anyone can send to any port on any host.
- No check on authenticity of IP address.
- Network packets are not private (Intermediate networks cannot be trusted).
- TCP state is easy to guess.

Sending Packets

Creation of packets is handled by the OS.

In our programs, we specify the data that needs to be sent; the packets are then created by the OS and sent over the network to the destination.



Example Program to Send a Packet

SendPkt.py

```
import socket
IP = "127.0.0.1"
PORT = 9090
data = b'Hello World!'

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

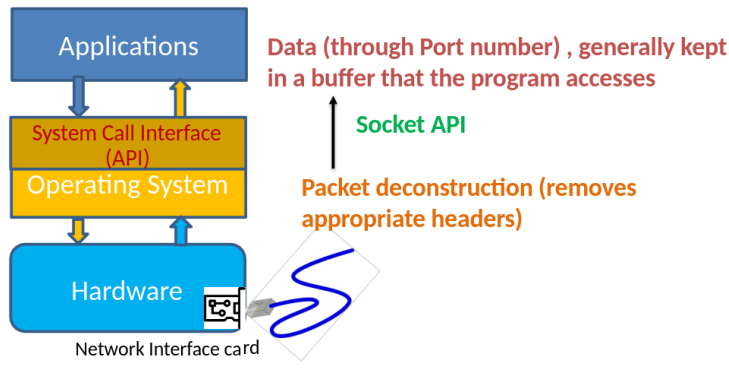
```
$ python SendPkt.py
```

```
$ nc -lvp 9090
```

Receiving Packets

Packets go through the network routers and eventually reach the destination IP address.

Packet at the destination goes through different layers, Data link, IP, Network layer; and finally data is handed over to the application (through the socket).



e.g. Program to receive a packet

ReceivePkt.py

```
import socket

IP = "0.0.0.0"
PORT = 9090
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(IP, PORT)

While True:
    data, (ip, port) = sock.recvfrom(1024)
    print("Sender: {} and Port: {}".format(ip, port))
    print("Received Message: {}".format(data))
```

```
$ nc -u <IP address> -p 9090
```

Why did we not bind the client with a port number?



**Protocols, Vulnerabilities and
Attacks By Attack Surface (i.e.
Layers)**

Attack Types

“Most” attacks on the Network Interface and the Network layer are **DoS** and **Spoofing** attacks.

DoS: Resource exhaustion which leads to lack of availability.

- Categorisation by Volume (exhausting bandwidth):
 - Volumetric, e.g., ICMP Flood, UDP Flood
 - Protocol/Application (misusing a protocol or an application to disrupt or exhaust the target's resources)
 - e.g. protocol: SYN Flood, Ping of Death, Smurf Attack, Fragmentation Attacks
 - e.g. application: HTTP Flood, Slowloris
- Categorisation by resource disparity (attack v defence):
 - Symmetric
 - Asymmetric (substantial damage with minimal resources)
- Categorisation by Direction:
 - Direct
 - Reflected

Fundamental skills that lots of network attacks depend on.

- **Sniffing:** (a.k.a **snooping**) *tapping* each packet as it flows across the network; i.e., it is a technique in which a user sniffs data belonging to other users of the network.
- **Spoofing:** *forging* a packet, to put some fake information in a packet and send it out.

Many LAN networks work with a **broadcast medium**.

- Packets on the wire are *heard* by all machines in that network.
- If the destination address matches with the machine's address, it accepts it; otherwise, it rejects it.

Addresses:

- Layer 2: MAC address: **identifies a machine on a network**.
- Layer 3: IP address: **identifies a network**.

Layer 2: How do we tell the NIC card to accept all the packets irrespective of what it is programmed to receive (specified by the destination MAC address)?

- Set NIC card in *promiscuous mode*.

Layer 3: Checks destination IP address: not for me → drops.

- However, many OS provide **raw socket** type:
 - in **normal socket**, packets get passed through the TCP/IP protocol stack: each layer strips the corresponding headers, the application gets the data.
 - in **raw sockets**, the packets are passed directly by the OS to the application, it *includes all the headers*.

Packet Sniffing: Programmes/Libraries

Although we can write our own sniffer programs,

- it will be time consuming (involves low level programming)
- not portable

Sniffing API:

- **PCAP** (Packet Capture API)
 - **libcap** in linux, **WinPcap** and **Npcap** in Windows
 - Written in C. Other languages offer wrappers.
 - Widely used by many tools:
 - **Wireshark**
 - **tcpdump**
 - **Scapy**
 - **McAfee**
 - **Nmap**
 - **Snort**, ...

Packet Sniffing using Scapy

Scapy

(<https://scapy.readthedocs.io/en/latest/introduction.html>)

- is built on top of Pcap.
- is a packet manipulation tool for computer networks, originally written in Python

Installation:

```
$ sudo pip3 install scapy
```

Importing **Scapy** in a python program:

```
from scapy.all import *
```

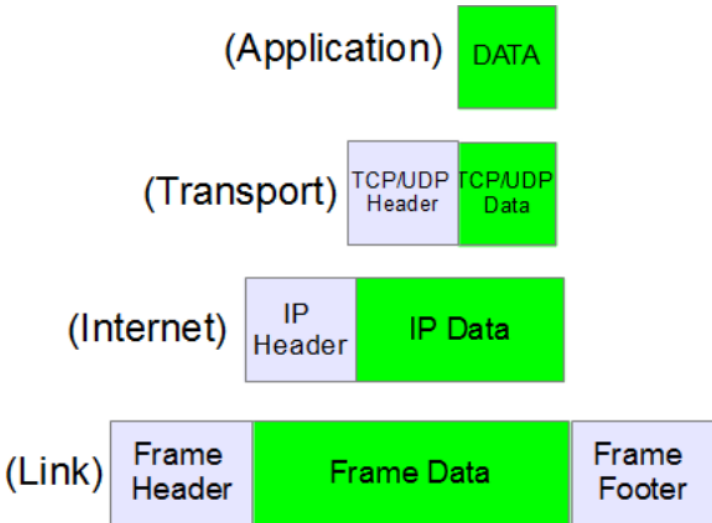
Scapy: Example

```
from scapy.all import *
pkt = sniff(iface='enp0s3',
            filter='icmp or udp',
            count=10)
pkt.summary()
```

Ways to display:

- **hexdump()**
- **pkt.show()**

Layers and Headers



What's in the packet

E.g.: the packet that we get through **Scapy** is an object of type Ethernet:

```
>>> pkt
<Ether type =IPv4 | <IP frag = 0 proto = udp | UDP | Raw
  load = 'hello' | >>>>
```

```
>>> pkt.payload
<IP frag = 0 proto = udp | UDP | Raw load = 'hello' |
>>>
```

```
>>> pkt.payload.payload
UDP | Raw load = 'hello' | >>
```

```
>>> pkt.payload.payload.payload
Raw load = 'hello' | >
```

Accessing Layers

```
>>> pkt.haslayer(UDP)
```

```
True
```

```
>>> pkt.haslayer(TCP)
```

```
0
```

```
>>> pkt.getlayer(UDP)
```

```
UDP | Raw load = 'hello' | >>
```

```
>>> pkt[UDP]
```

```
UDP | Raw load = 'hello' | >>
```

```
>>> pkt[Raw].load
```

```
b'hello'
```

Get Information of Protocol Classes

Get attribute names:

```
>>> ls(IP)
```

Get method names:

```
>>> help(IP)
```

Using Tools : Wireshark

Free and open source network protocol analyser.

Similar to **TCPDump** but has a graphical front-end:

The screenshot displays the Wireshark interface with a network capture of a DNS transaction. The packet list pane shows several packets, with packet 349 highlighted. The packet details pane shows the structure of the DNS query and response. The packet bytes pane shows the raw hex and ASCII data of the response.

No.	Time	Source	Destination	Protocol	Length	Info
343	05.142415	192.168.0.21	174.129.249.228	TCP	66	40555 → 80 [ACK] Seq=1 Ack=1 Min=5888 Len=0 TSval=491519346 TSecr=551811827
344	05.142715	192.168.0.21	174.129.249.228	HTTP	253	GET /clients/netflix/Flash/application.swf?flash_version=Flash_lite_2.1&w=1.5&r=
345	05.230738	174.129.249.228	192.168.0.21	TCP	66	80 → 40555 [ACK] Seq=1 Ack=188 Min=8804 Len=0 TSval=551811850 TSecr=491519347
346	05.240742	174.129.249.228	192.168.0.21	HTTP	820	HTTP/1.1 302 Moved Temporarily
347	05.241592	192.168.0.21	174.129.249.228	TCP	66	40555 → 80 [ACK] Seq=188 Ack=763 Min=7424 Len=0 TSval=491519446 TSecr=551811852
348	05.242532	192.168.0.21	192.168.0.1	DNS	77	Standard query 0x2188 A cdn-0.nflximg.com
349	05.276870	192.168.0.1	192.168.0.21	DNS	489	Standard query response 0x2188 A cdn-0.nflximg.com CNAME images.netflix.com.edge
350	05.277992	192.168.0.21	63.80.242.48	TCP	74	37863 → 80 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=491519482 TSecr=
351	05.297757	63.80.242.48	192.168.0.21	TCP	74	80 → 37863 [SYN, ACK] Seq=0 Ack=1 Min=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=329554130
352	05.298936	192.168.0.21	63.80.242.48	TCP	66	37863 → 80 [ACK] Seq=1 Ack=1 Min=5888 Len=0 TSval=491519502 TSecr=329554130
353	05.298667	192.168.0.21	63.80.242.48	HTTP	153	GET /us/mrd/clients/Flash/B14540.bun HTTP/1.1
354	05.318770	63.80.242.48	192.168.0.21	TCP	66	80 → 37863 [ACK] Seq=1 Ack=85 Min=5792 Len=0 TSval=329554151 TSecr=491519503
355	05.321733	63.80.242.48	192.168.0.21	TCP	1514	[TCP segment of a reassembled PDU]

Packet 349 details:

- Ethernet II, Src: GlobalStar (00:130:80:00:00:00), Dst: Vizio_14 (08:00:19:9d:14:b0e1)
- Internet Protocol Version 4, Src: 192.168.0.1, Dst: 192.168.0.21
- User Datagram Protocol, Src Port: 53 (53), Dst Port: 34036 (34036)
- Domain Name System (response)
 - Request ID: 0x2188
 - Transaction ID: 0x2188
 - Flags: 0x18 Standard query response, No error
 - Questions: 1
 - Answer RRs: 4
 - Authority RRs: 0
 - Additional RRs: 9
 - Queries
 - cdn-0.nflximg.com: type A, class IN
 - Answers
 - Authoritative nameservers

Packet bytes:

```
0020 00 15 00 35 04 f4 01 c7 83 ff 81 80 00 01 .....f.....
0020 00 04 00 09 00 09 05 63 64 00 2d 38 07 64 66 6c .....c dn-0.nfl
0040 76 09 6d 67 03 63 6f 6d 00 00 01 00 01 c0 8c 80 .....img.com.....
0060 05 00 01 00 00 05 29 00 22 06 69 6d 01 67 65 73 .....".images
0080 07 6e 65 74 66 6c 69 78 03 6f 6d 89 05 64 67 .....netflix.com.edg
00a0 05 73 75 69 74 65 03 6e 65 74 00 c0 2f 00 05 00 .....esuite.n et./...
```

Packet Spoofing

Recap: Sending a normal packet

```
import socket
IP = "127.0.0.1"
PORT = "9090"
data = b'Hello World !'
sock = socket(socket.AF_INET, socket.SOCK_DGRAM)
Sock.sendto(data, (IP, PORT))
```

Source IP? Source Port?

TCP/IP Protocol stack creates the packet by adding headers (by different layers). We generally set only a few attributes of headers (destination IP address, port number, some flags).

Packet Spoofing with Scapy

Constructing packets:

- We need to control the headers for packet snooping.
- raw socket sends the forged packets.

```
>>> a = IP(src='1.2.3.4', dst = '10.20.30.40')
>>> b = UDP(sport='1234', dport = '1020')
>>> c = "Hello World"
>>> pkt = a/b/c
>>> pkt.show()
```

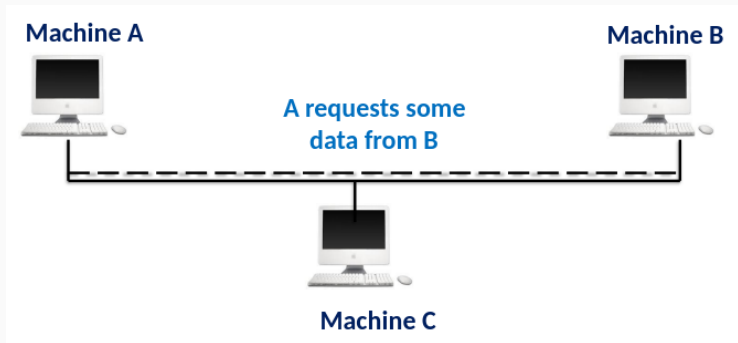
Spoofting ICMP Packet

```
from scapy.all import *  
  
ip = IP(src='1.2.3.4', dst='94.180.216.34')  
icmp = ICMP()  
pkt = ip/icmp  
send(pkt, verbose=0)
```


Spoofting UDP Packet

```
from scapy.all import *  
  
ip = IP(src='1.2.3.4', dst='94.180.216.34')  
udp = UDP(sport=9090, dport=9100)  
data = 'Hello! \n'  
pkt = ip/udp/data  
send(pkt)
```

Sniff Request and Spoof Reply



- Sniffing traffic, knows what is being requested by A.
- Sends data to A showing that the data has come from B.

Next: Physical Layer and Data Link Layer Attacks