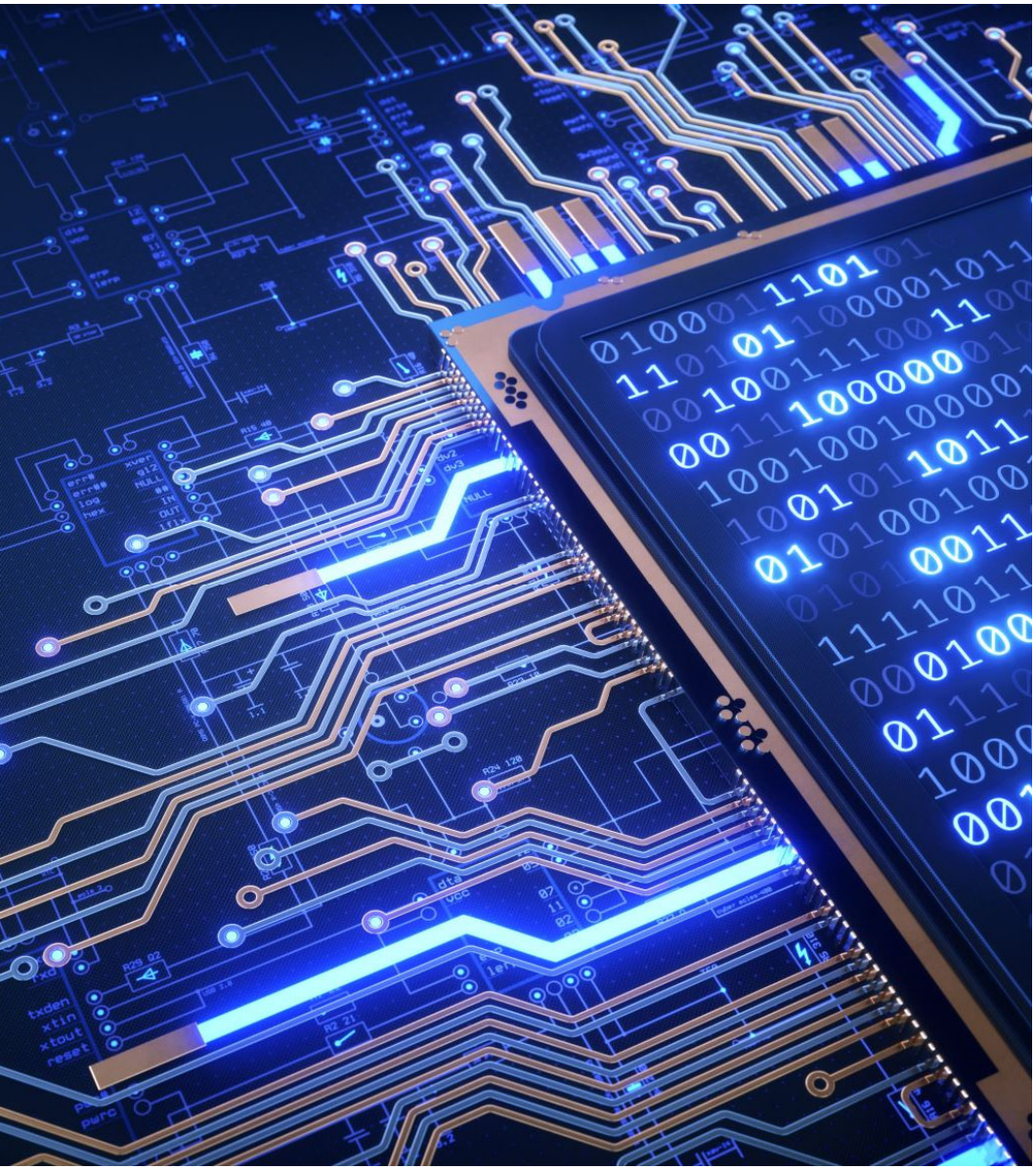
Network Traffic Analysis

Intro to techniques and tools

21 June 2024

Agenda

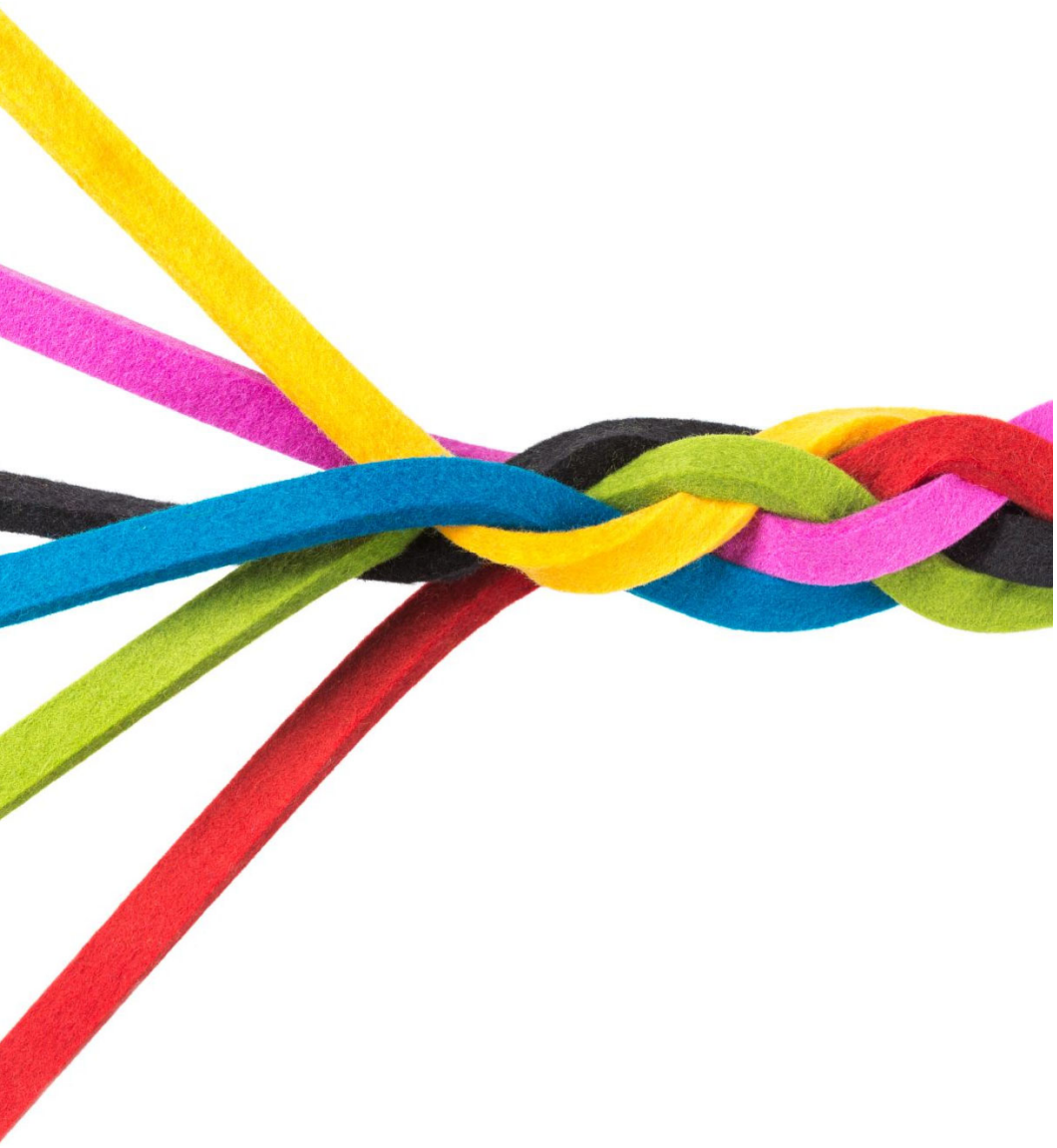
- Facts / Assumptions
- Why do we care?
- What is a packet?
- Tool Overview
- Exercises
- References



Facts & Assumptions

- You have used a computer before
- This is meant to be introductory level
- Network analysis is incredibly broad
- There will be no ML, AI, or IPv6
- If something is incorrect, let me know
- I will send you the slides

"Just remember that a CPU is literally a rock that we tricked into thinking," Someone



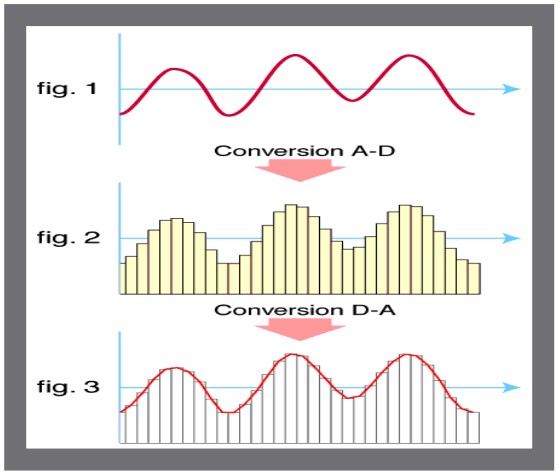
Why do we care about this?

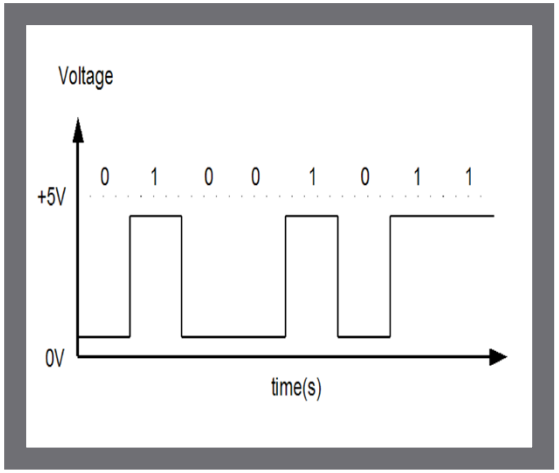
Regardless of work role, understanding network activity is critical for cyber operations.

- Identifying Malicious Activity
- Creating Malicious Activity
- Malware Analysis
- Network Performance Optimization
- Security Policy Enforcement

What is Network Traffic?

- **Analog**
  - Continuous Time and Continuous Amplitude
  - 2.4 GHz and 5 GHz (cycles per second)
  - Quantization error when converted to Digital
- **Digital**
  - Discrete Time and Discrete Amplitude
  - Copper Wire:
    - 0V encoded as a binary 0
    - 5V encoded as a binary 1





Why binary?

- **The Soviet Union experimented with "Base 3" Ternary Computers. A ternary bit is a "trit".**
- **Now that's something you know.**

- **Binary makes hardware easier.**
  - **We use a voltage threshold, such as 5V, to indicate a "1" value. Adding a 3rd voltage, such as 10V, to indicate a "2" would make engineering far more complicated.**

  - **Logic gates with 2 inputs are more efficient to implement.**

Length	Name	Example
1	Bit	0
4	Nibble	1011
8	Byte	10110101

What do we do with all these 1's and 0's?

Standardizing Data – Does it get more fun?

The internet works because of standards. Understanding the rules helps you identify malicious and abnormal activity.

Who manages these standards?
- **Internet Engineering Task Force (IETF)**
- **International Organization for Standardization (ISO)**
- **International Electrotechnical Commission (IEC)**

2121333

byte

string that consists of a number of bits, treated as a unit, and usually representing a character or a part of a character

Note 1 to entry: The number of bits in a byte is fixed for a given data processing system.

Note 2 to entry: The number of bits in a byte is usually 8.

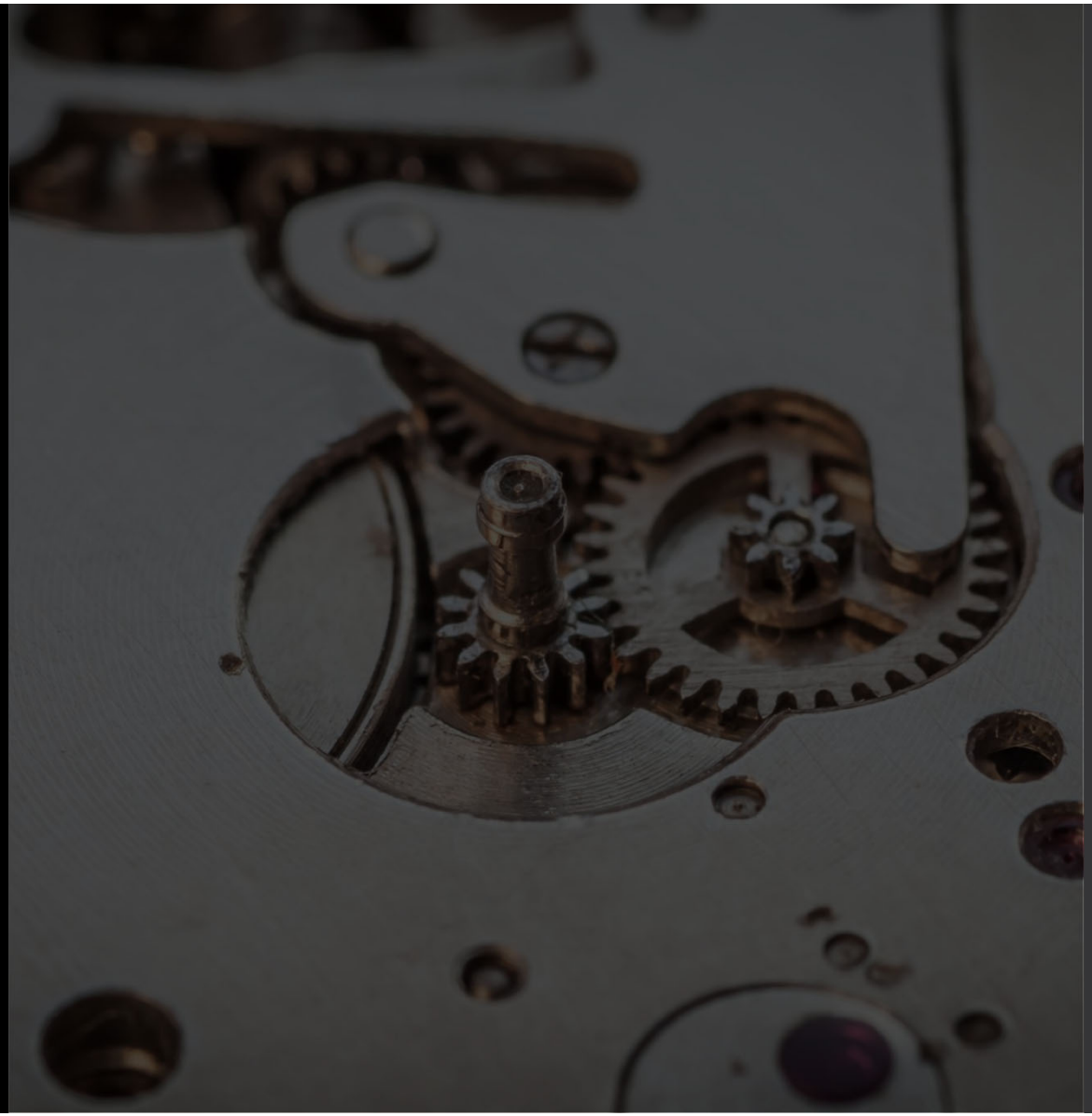
Note 3 to entry: byte: term and definition standardized by ISO/IEC [ISO/IEC 2382-1:1993; ISO/IEC 2382-4:1999].

Note 4 to entry: 01.02.09 (2382)

[SOURCE:ISO-IEC-2382-1 * 1993 * * *]

https://www.iso.org/obp/ui/en/#iso:std:iso-iec:2382:ed-1:v2:en

Every standard is made up, So, every rule can be broken.

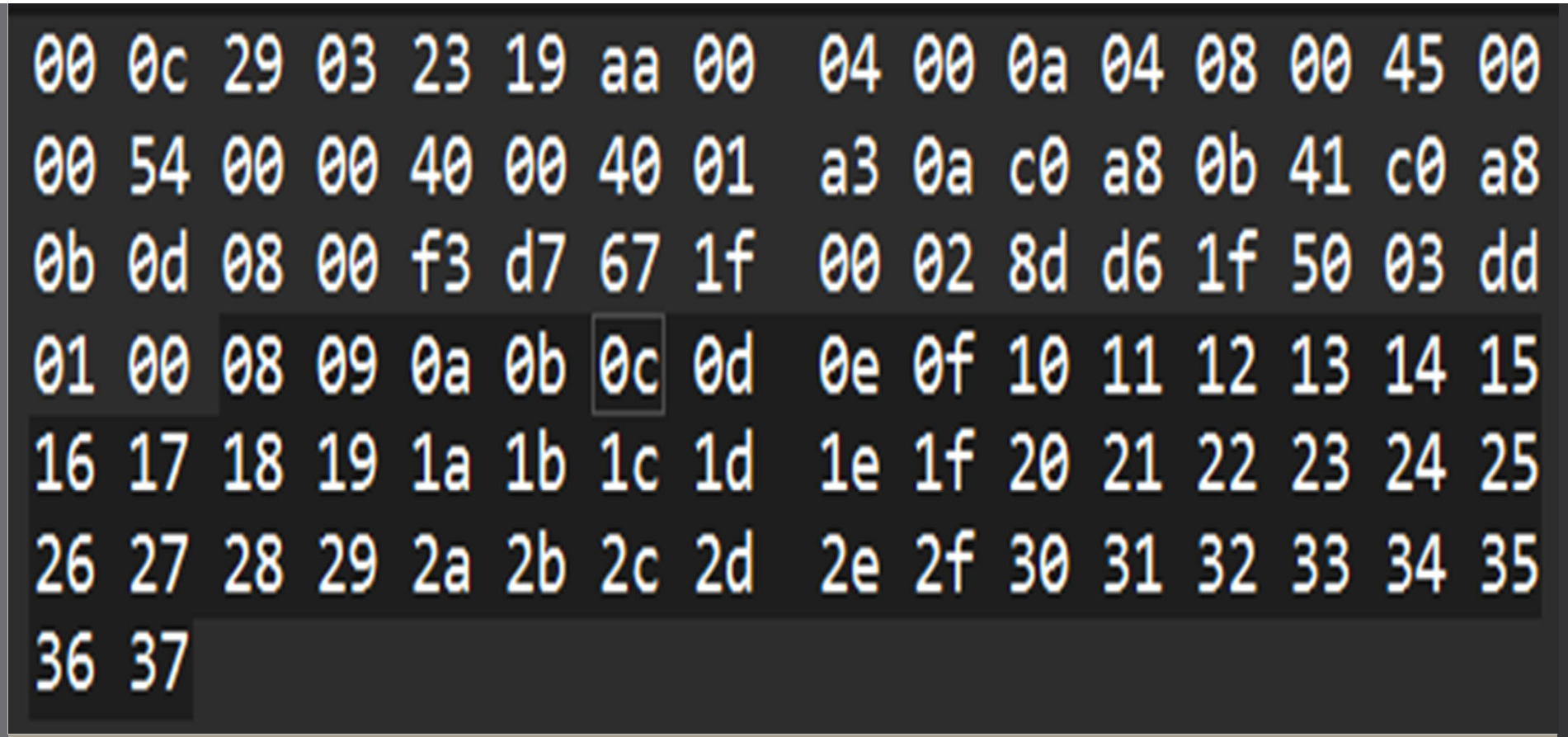


```
00000000 00001100 00101001 00000011 00100011 00011001 10101010 00000000
00000100 00000000 00001010 00000100 00001000 00000000 01000101 00000000
00000000 01010100 00000000 00000000 01000000 00000000 01000000 00000001
10100011 00001010 11000000 10101000 00001011 01000001 11000000 10101000
00001011 00001101 00001000 00000000 11110011 11010111 01100111 00011111
00000000 00000010 10001101 11010110 00011111 01010000 00000011 11011101
00000001 00000000 00001000 00001001 00001010 00001011 00001100 00001101
00001110 00001111 00010000 00010001 00010010 00010011 00010100 00010101
00010110 00010111 00011000 00011001 00011010 00011011 00011100 00011101
00011110 00011111 00100000 00100001 00100010 00100011 00100100 00100101
00100110 00100111 00101000 00101001 00101010 00101011 00101100 00101101
00101110 00101111 00110000 00110001 00110010 00110011 00110100 00110101
00110110 00110111
```

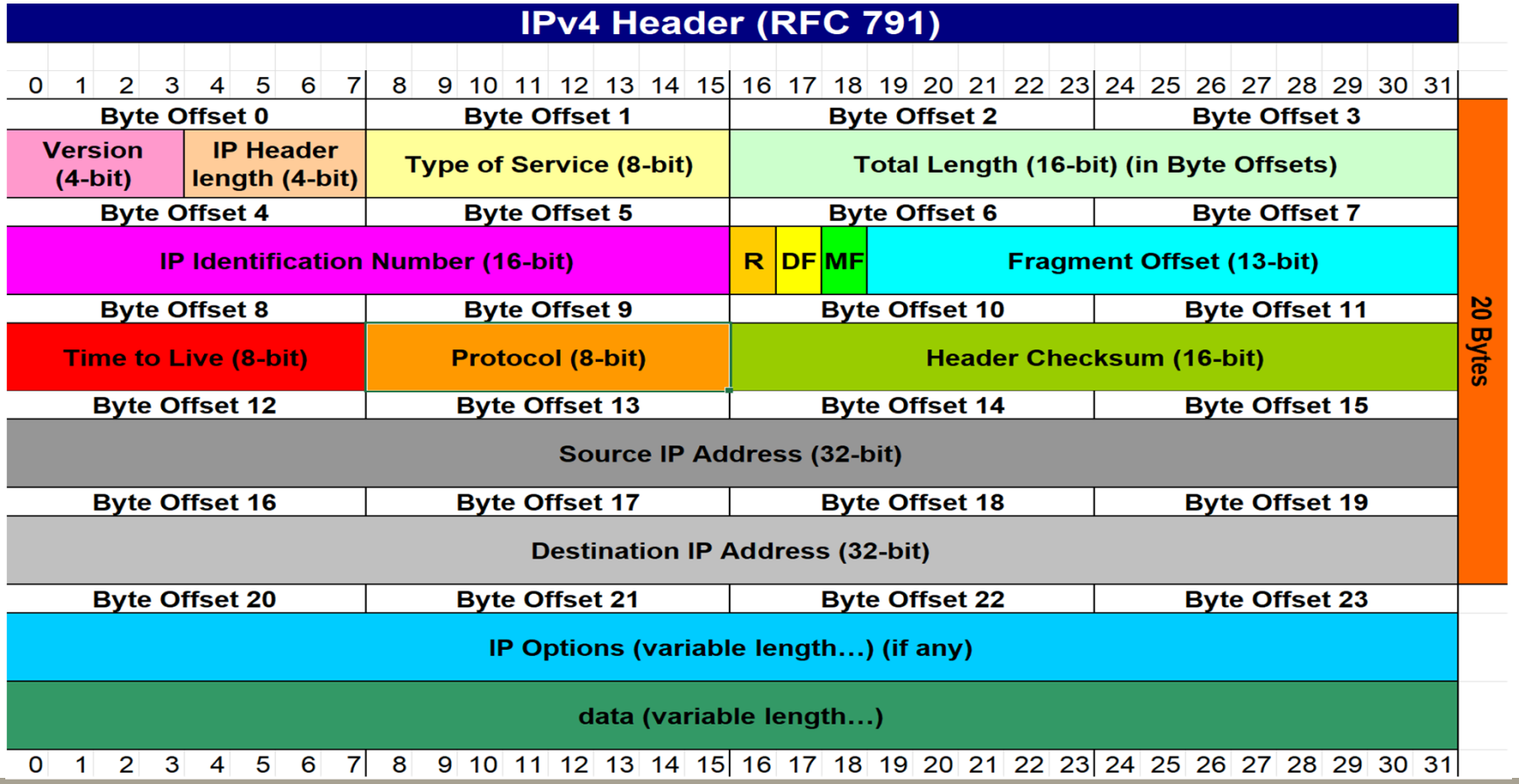
What is this nonsense.

```
00000000 00001100 00101001 00000011 00100011 00011001 10101010 00000000
00000100 00000000 00001010 00000100 00001000 00000000 01000101 00000000
00000000 01010100 00000000 00000000 01000000 00000000 01000000 00000001
10100011 00001010 11000000 10101000 00001011 01000001 11000000 10101000
00001011 00001101 00001000 00000000 11110011 11010111 01100111 00011111
00000000 00000010 10001101 11010110 00011111 01010000 00000011 11011101
00000001 00000000 00001000 00001001 00001010 00001011 00001100 00001101
00001110 00001111 00010000 00010001 00010010 00010011 00010100 00010101
00010110 00010111 00011000 00011001 00011010 00011011 00011100 00011101
00011110 00011111 00100000 00100001 00100010 00100011 00100100 00100101
00100110 00100111 00101000 00101001 00101010 00101011 00101100 00101101
00101110 00101111 00110000 00110001 00110010 00110011 00110100 00110101
00110110 00110111
```

An ICMP echo request – Obviously



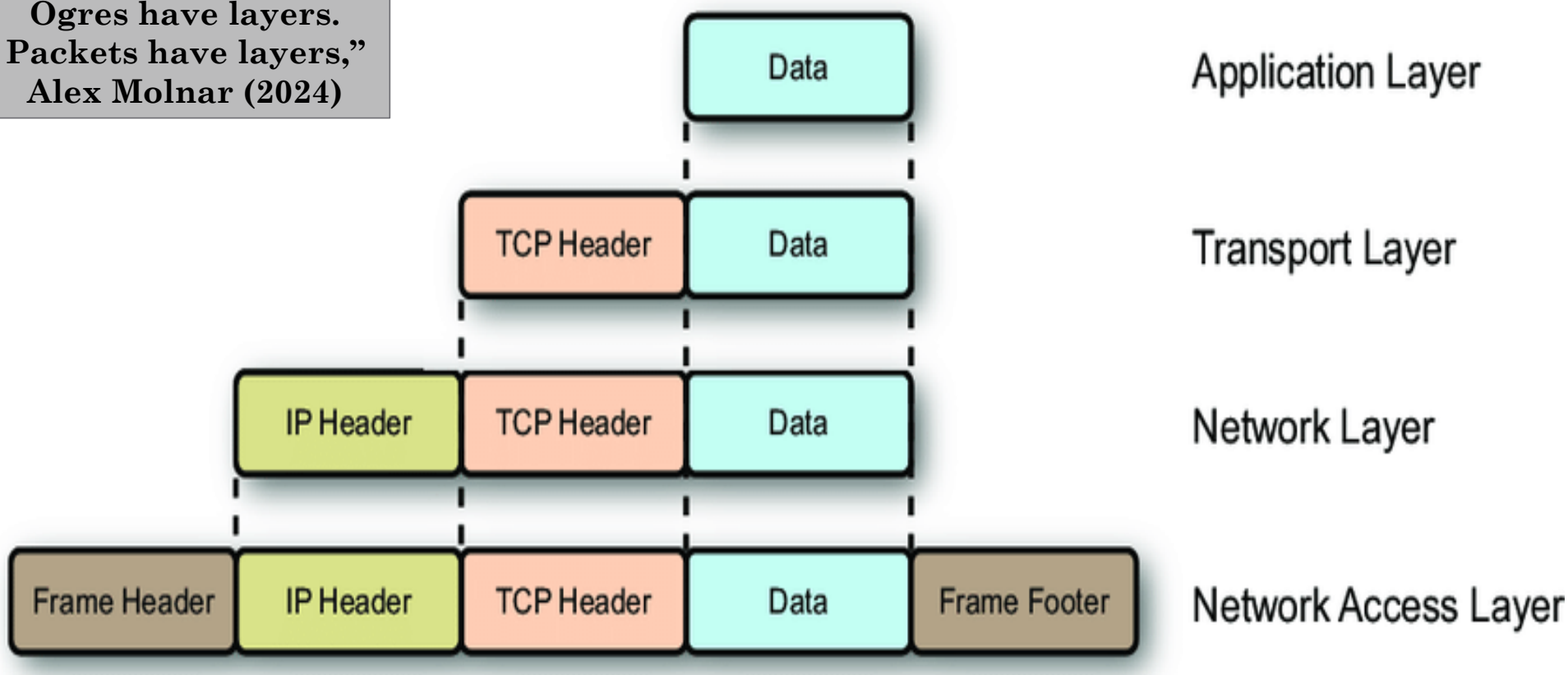
Each hexadecimal value = 4 bits

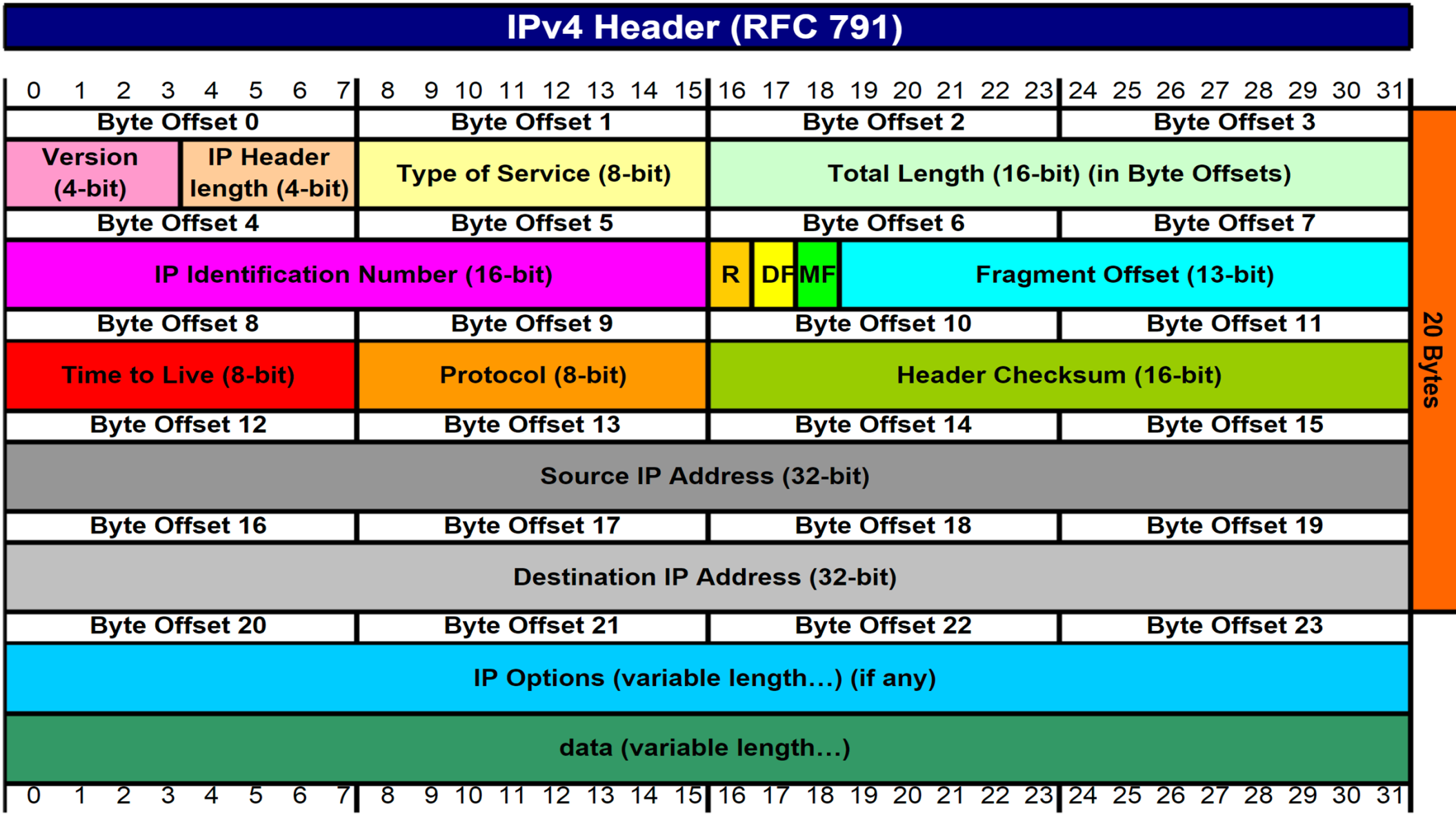


What is a packet? OOF

Packet Structure – Encapsulation

"Onions have layers. Ogres have layers. Packets have layers," Alex Molnar (2024)




IPv4 Header (RFC 791)
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
Byte Offset 0
Byte Offset 1
Byte Offset 2
Byte Offset 3
Version (4-bit)
IP Header length (4-bit)
Type of Service (8-bit)
Total Length (16-bit) (in Byte Offsets)
Byte Offset 4
Byte Offset 5
Byte Offset 6
Byte Offset 7
IP Identification Number (16-bit)
R
DF
MF
Fragment Offset (13-bit)
Byte Offset 8
Byte Offset 9
Byte Offset 10
Byte Offset 11
Time to Live (8-bit)
Protocol (8-bit)
Header Checksum (16-bit)
Byte Offset 12
Byte Offset 13
Byte Offset 14
Byte Offset 15
Source IP Address (32-bit)
Byte Offset 16
Byte Offset 17
Byte Offset 18
Byte Offset 19
Destination IP Address (32-bit)
Byte Offset 20
Byte Offset 21
Byte Offset 22
Byte Offset 23
IP Options (variable length…) (if any)
data (variable length…)
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
20 Bytes

- 192 .168 .1 .1 (ASCII)
- C0 .A8 .01 .01 (HEX)
- 11000000.10101000.00000001.00000001 (Binary)

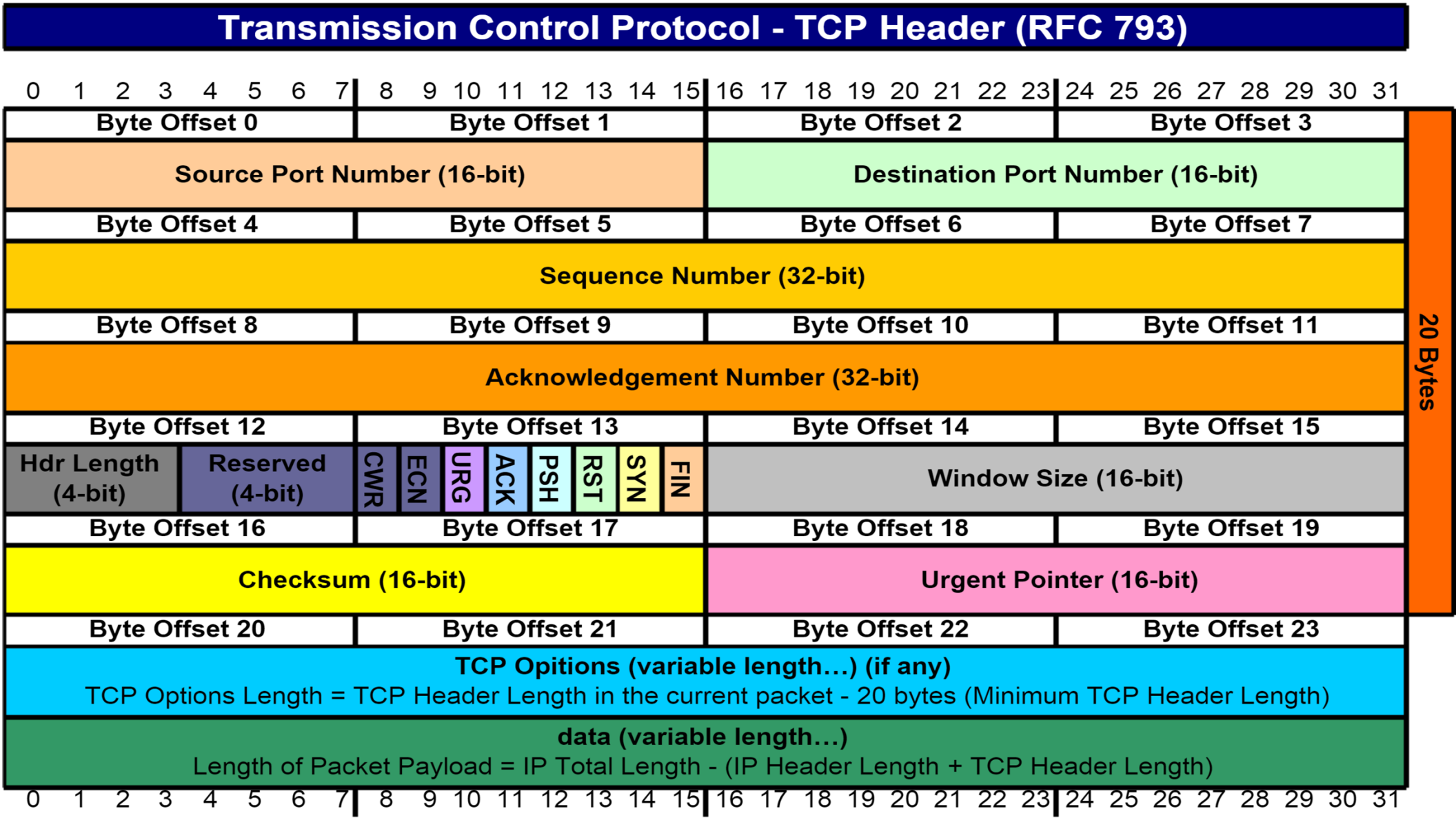
- **Protocol Types**:
  - **ICMP** (1)
  - **TCP** (6),
  - **UDP** (17)
  - **Etc**.

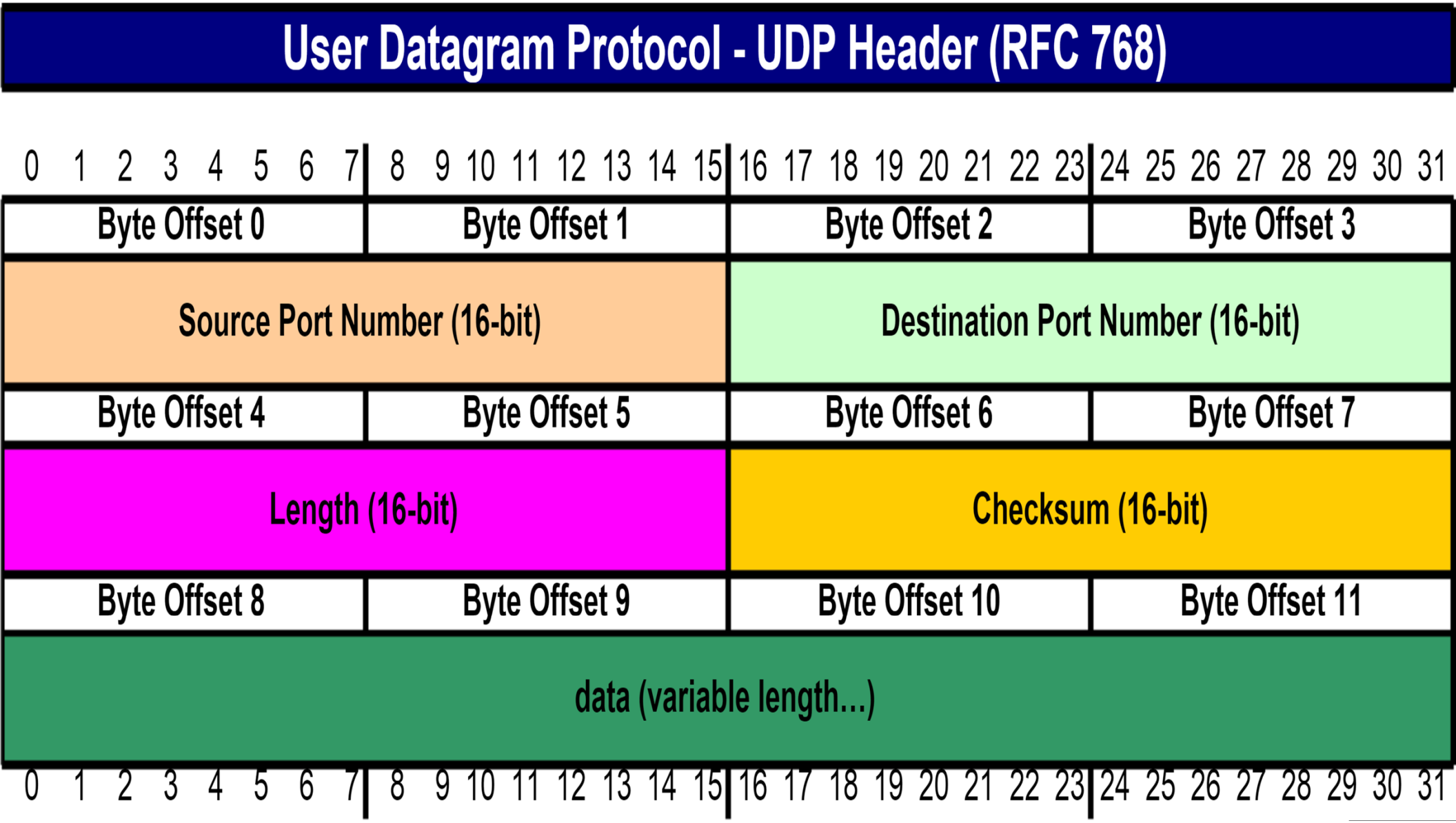
Fragmentation:

R: Reserved Bit

DF: Don't Fragment

MF: More Fragments


Transmission Control Protocol - TCP Header (RFC 793)
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
Byte Offset 0
Byte Offset 1
Byte Offset 2
Byte Offset 3
Source Port Number (16-bit)
Destination Port Number (16-bit)
Byte Offset 4
Byte Offset 5
Byte Offset 6
Byte Offset 7
Sequence Number (32-bit)
Byte Offset 8
Byte Offset 9
Byte Offset 10
Byte Offset 11
Acknowledgement Number (32-bit)
Byte Offset 12
Byte Offset 13
Byte Offset 14
Byte Offset 15
Hdr Length (4-bit)
Reserved (4-bit)
CWR
ECN
URG
ACK
PSH
RST
SYN
FIN
Window Size (16-bit)
Byte Offset 16
Byte Offset 17
Byte Offset 18
Byte Offset 19
Checksum (16-bit)
Urgent Pointer (16-bit)
Byte Offset 20
Byte Offset 21
Byte Offset 22
Byte Offset 23
20 Bytes
TCP Opitions (variable length…) (if any)
TCP Options Length = TCP Header Length in the current packet - 20 bytes (Minimum TCP Header Length)
data (variable length…)
Length of Packet Payload = IP Total Length - (IP Header Length + TCP Header Length)
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31


User Datagram Protocol - UDP Header (RFC 768)
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
Byte Offset 0
Byte Offset 1
Byte Offset 2
Byte Offset 3
Source Port Number (16-bit)
Destination Port Number (16-bit)
Byte Offset 4
Byte Offset 5
Byte Offset 6
Byte Offset 7
Length (16-bit)
Checksum (16-bit)
Byte Offset 8
Byte Offset 9
Byte Offset 10
Byte Offset 11
data (variable length...)
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31

What tools do we care about?

- TCPdump
- PyShark / Tshark
- Netsh
- Pktmon
- Scapy
- Wireshark

TCPDump

```
$ tcpdump
  -i (interfaces)
  -w (write to file)
  -r (read from file)
  -c (number of packets to collect)
  -A (prints in ASCII)
  -X (print in hex and ASCII)
  -v (verbose)
  -nn (prevents resolution)
  -tttt (human readable time)
  -l (line-buffered output – real-time processing)
```

TCPDump

- **Standard on most Linux distributions**
- **May need to bring your own binary**

- **Can I have one packet?**

```
$ tcpdump -i eth0 -c 1
```

- **Capture traffic from/to a specific IP**

```
$ tcpdump -i eth0 host 8.8.8.8 -w test.pcap
```

- **Capture traffic where only the SYN flag is set**

```
$ tcpdump -i eth0 ‘tcp[tcpflags] == tcp-syn’
$ tcpdump -i eth0 ‘[13] = 2’
```

TCPDump – Monitoring

- **Add colors, time formatting, and don't resolve ports**

```
$ sudo tcpdump -i eth0 -tttt -nn -l |
GREP_COLORS='mt=5;31' grep --color=always
"192.168.217.128" | ccze -A
```

- **Look for ASCII strings such as user, pass, and login**

```
$ sudo tcpdump -A -i eth0 'port http or
port ftp or port telnet' |
grep -i 'user\|pass\|login'
```

PyShark & Tshark

```
$ pip install pyshark
```

```python
import pyshark

# network interface to capture packets
interface = 'eth0'

# number of packets to capture
num_packets = 10

# start live capture
capture = pyshark.LiveCapture(interface=interface)
capture.sniff(packet_count=num_packets)

# Process each captured packet
for packet in capture:
    print('-' * 50)
    print(f"Packet Number: {packet.number}")
    if 'ip' in packet:
        print(f"Source IP: {packet.ip.src}")
        print(f"Destination IP: {packet.ip.dst}")
    if 'tcp' in packet:
        print(f"Source Port: {packet.tcp.srcport}")
        print(f"Destination Port: {packet.tcp.dstport}")
    print('-' * 50)
```

Netsh

- **Requires administrative privileges**

```
C:\> netsh interface show interface
C:\> netsh wlan show interfaces
C:\> netsh trace start capture=yes CaptureInterface="Wi-Fi"
C:\> netsh trace stop
```

Netsh – etl2pcapng

https://github.com/microsoft/etl2pcapng/releases

```
C:\> etl2pcapng in.etl out.pcapng
```

Netsh Scenarios

```
C:\> netsh trace show scenarios
```

```
LAN                     : Troubleshoot wired LAN related issues
Layer2                  : Troubleshoot layer 2 connectivity related issues
MBN                     : Troubleshoot mobile broadband related issues
NDIS                    : Troubleshoot network adapter related issues
NetConnection           : Troubleshoot network connection related issues
NetworkSnapshot         : Collect the current network state of the system
P2P-Grouping            : Troubleshoot Peer-to-Peer Grouping related issues
P2P-PNRP                : Troubleshoot Peer Name Resolution Protocol (PNRP) related issues
RemoteAssistance        : Troubleshoot Windows Remote Assistance related issues
Virtualization          : Troubleshoot network connectivity issues in virtualization environment
VPNServer               : Troubleshoot VPN related issues
WCN                     : Troubleshoot Windows Connect Now related issues
WFP-IPsec               : Troubleshoot Windows Filtering Platform and IPsec related issues
WLAN                    : Troubleshoot wireless LAN related issues
XboxMultiplayer         : Troubleshoot Xbox Live Multiplayer connectivity-related issues
```

Pktmon – Windows

- **Requires administrative privileges**

```
C:\> pktmon start --capture --comp nics --flags 0x1E
```

- **nics: monitor Network Interface Cards**
- **--flags 0x01: Capture packets**

```
C:\> pktmon stop
```

```
C:\> pktmon pcapng .\PktMon.etl –o test.pcapng
```

Scapy – Getting Started

```
$ pip install scapy
```

```
$ python
```

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

```
>>> packet = IP()/TCP()
```

```
>>> Ether()/packet
```

```
>>> ls(IP, verbose=True)
```

Scapy – Getting Started

```
$ scapy
```

Scapy – Interactive Mode

```
>>> lsc()
>>> pkt = sniff(count=1)
>>> type(pkt)
>>> pkt[0].summary()
>>> explore(scapy.layers.dhcp)
>>> ls(Ether)
>>> ls(IP)
>>> pkt[0].show()
>>> pkts = sniff(count=10)
>>> pkts.summary()
>>> pkts[5][<layer>].summary()
>>> pkts[5].command()
```

Scapy – Make & Send a Packet

```
>>> lsc()
>>> pkt = sniff(count=1)
>>> type(pkt)
>>> pkt[0].summary()
>>> explore(scapy.layers.dhcp)
>>> ls(Ether)
>>> ls(IP)
>>> pkt[0].show()
>>> pkts = sniff(count=10)
>>> pkts.summary()
>>> pkts[5][<layer>].summary()
>>> pkts[5].command()
```

```python
from scapy.all import sniff

# Apply to each packet
def summarize(packet):
    print(packet.summary())

# Capture packets with options
sniff(
    count=5,                    # Set to 0 to capture indefinitely
    iface='eth0',               # Network interface
    prn=summarize,              # Function to apply to each packet
    timeout=10                  # Stop sniffing after specified seconds
)
```

Scapy
– More options

```python
from scapy.all import Ether, IP, ICMP, send, sendp

# Create an IP packet with an ICMP layer
packet_IP = IP(dst="192.168.1.1") / ICMP()


# Operates at the Network Layer. Automatically determines appropriate
# interface based on routing table and destination IP
send(packet)

# Create an Ethernet frame with an IP and ICMP layer
packet_ETH = Ether() / IP(dst="192.168.1.1") / ICMP()

# Operated at the Data Link Layer. Requires user to specify
# the interface. Useful for low-level packet crafting.
sendp(packet, iface="eth0")
```

Scapy
– Send packets

```python
from scapy.all import sniff, wrpcap

# Packet list
packets = []

def packet_callback(packet):
    packets.append(packet)

# Capture 10 packets on the 'eth0' interface and store them in the list
sniff(count=10, iface="eth0", prn=packet_callback, store=1)

# Write the captured packets to a pcap file
wrpcap("captured_packets.pcap", packets)
```

Scapy – Write to pcap

```python
from scapy.all import rdpcap

# Read packets from the pcap file
packets = rdpcap("captured_packets.pcap")

# Print a summary of each packet
for packet in packets:
    print(packet.summary())
```


Scapy
– Read from pcap

```python
from scapy.all import IP, ICMP, sr1
import time
import random

def quiet_ping_sweep(ip_range):
    for ip in ip_range:
        packet = IP(dst=ip) / ICMP()
        response = sr1(packet, timeout=1, verbose=False)
        if response:
            print(f"{ip} is alive")
        time.sleep(random.uniform(1, 5)) # randomize delay between 1 and 5 seconds

ip_range = [f"192.168.1.{i}" for i in range(1, 11)]
random.shuffle(ip_range)  # Randomize the order of targets
quiet_ping_sweep(ip_range)
```


Scapy – Ping Sweep

Because you have to perform unnecessarily complicated host discovery in any cyber security class

Wireshark – Setup

Disclaimer: Some PCAPs contain Windows-based malware, so don't download files contained in the traffic.

```
$ sudo apt-get install wireshark
```

- **Wireshark Configuration:**
  - Edit → Preferences → Fonts and Colors
  - Help → About Wireshark
  - Edit → Configuration Profiles →  → Ok
  - "Right Click" the following columns → Remove this Column
    - Frame Number, Protocol, Length
  - "Right Click" any column → Column Preferences

Displayed	Title	Type
✓	Time	UTC date, as YYYY-MM-DD, and time
✓	Src	Src addr (unresolved)
✓	Sport	Src port (unresolved)
✓	Dst	Dest addr (unresolved)
✓	Dport	Dest port (unresolved)
✓	Info	Information

  - Align port columns to the left
  - View → Time Display Format → Seconds and UTC Date / Time of Day

- **Wireshark Configuration:**
  - "Right Click" any column → Column Preferences
    - http.host or tls.handshake.extensions_server_name

  - Basic Web Filter:
    - This displays HTTP URLs and HTTPS domains
    - Also ignore Simple Service Discovery Protocol

    - Save filter → + → Basic for label
  - Web Filter and SYN flags

Capture Filters	Display Filters
tcp port 80	tcp.port == 443
host 192.168.1.100	ip.src == 192.168.1.100
net 192.168.1.0/24	ip.addr ==192.168.1.0/24
ether host aa:bb:cc:dd:ee:ff	eth.addr == aa:bb:cc:dd:ee:ff
tcp[tcpflags] & tcp-syn != 0	tcp.flags.syn == 1
not host 192.168.1.1	http.request.method == “GET”
not broadcast and not multicast	dns.qry.name contains “example.com”

Wireshark – Capture & Display

Exercises

Basic:
https://github.com/alexjmolnar/network-analysis

Unit42: (These PCAPs contain malware)
First:
https://unit42.paloaltonetworks.com/january-wireshark-quiz/
https://unit42.paloaltonetworks.com/january-wireshark-quiz-answers/
Second:
https://unit42.paloaltonetworks.com/wireshark-quiz-icedid/
https://unit42.paloaltonetworks.com/wireshark-quiz-icedid-answers/
PCAP Github: https://github.com/pan-unit42/Wireshark-quizzes/