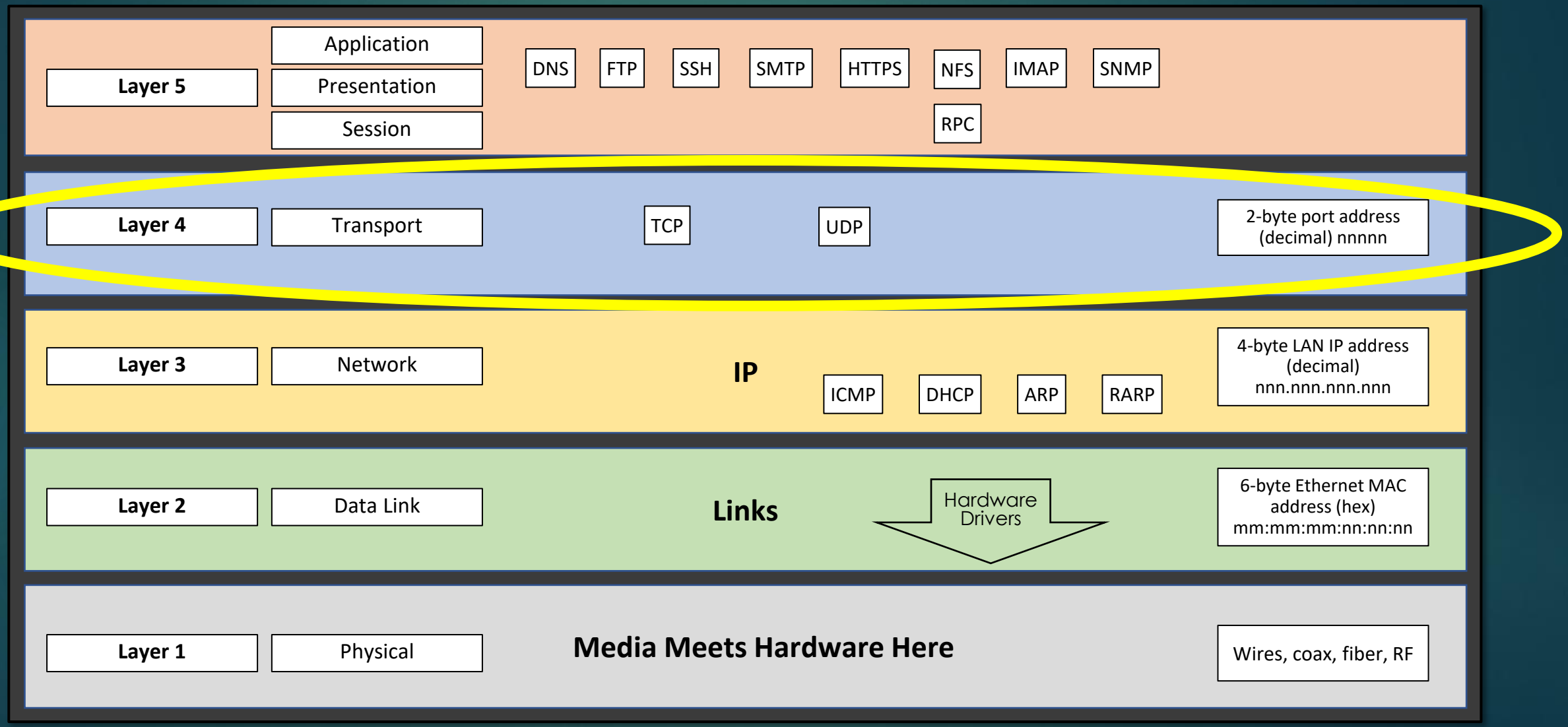
# IT/CYBR 4323, Chapter 17:

# TCP


- Slides written by Mr. Donald Privitera with Mr. Gennadiy Kemelmakher to accompany An Introduction to Computer Networks written by Peter L Dordal, http://intronetworks.cs.luc.edu/
- Kennesaw State University
- Marietta Campus
- College of Computing and Software Engineering
- IT Department
- dprivit2@kennesaw.edu, gkemelma@kennesaw.edu

# 5-Layer TCP/IPv4 Network Stack



# Topics

- TCP Basics
- Header
- Handshake
- Hardware Assistance
- Coding in Python

# TCP Basics

TCP is…

- **Stream-oriented**.
- **Connection-oriented.**
- **Reliable.**
- Based on **sliding windows.**
- Suitable for **request/reply** protocols (factoring in overhead).
- Has a concept of **socketpair** which creates exclusivity for that stream.
- Heavily influenced by the **End-to-End** design principle.
- **Segments** are the data portions of TCP packets.

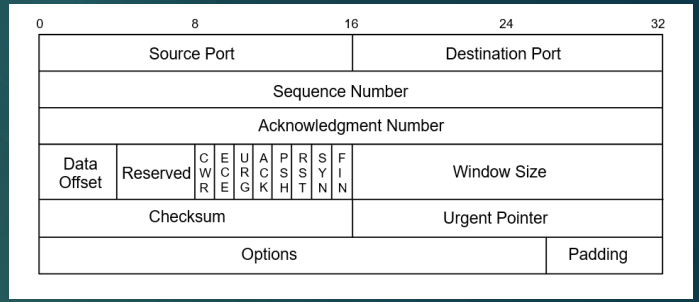
# TCP IP Header

| Offsets | Octet | 0 | | | | | | | | 1 | | | | | | | | 2 | | | | | | | | 3 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Octet | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | Source port | | | | | | | | | | | | | | | | Destination port | | | | | | | | | | | | | | | |
| 4 | 32 | Sequence number | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 64 | Acknowledgment number (if ACK set) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | 96 | Data offset | | | | Reserved<br>0 0 0 | | | | NS | CWR | ECE | URG | ACK | PSH | RST | SYN | FIN | Window Size | | | | | | | | | | | | | | |
| 16 | 128 | Checksum | | | | | | | | | | | | | | | | | Urgent pointer (if URG set) | | | | | | | | | | | | | | |
| 20 | 160 | Options (if data offset > 5. Padded at the end with "0" bits if necessary.) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ⋮ | ⋮ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 60 | 480 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

https://en.wikipedia.org/wiki/Transmission_Control_Protocol

Peter L Dordal, An Introduction to Computer Networks

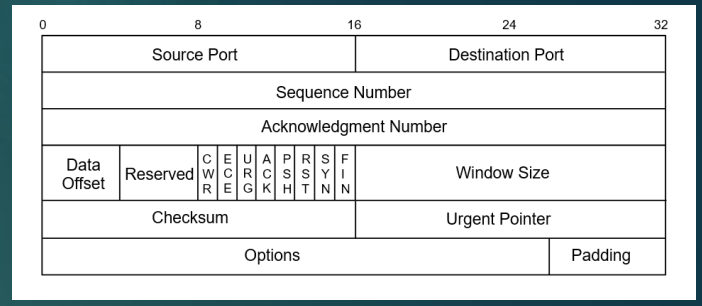
# TCP IP Header




- **Sequence** and **Acknowledgement** numbers are for numbering data. Typically, 1024-byte blocks of data are sent and these numbers are incremented by that amount. These are relative to the **Initial Sequence Number (ISN)**, a random 32-bit number.
- Acknowledgements are **cumulative** acking all bytes < N.
- **Data Offset** (4 bits) = size of TCP header in 32-bit words
- **Flags** (9 bits)
- **Window Size** (16 bits) = size of receive window in (size units)
- **Urgent Pointer** (16 bits)
- **Options** (0-10 32-bit words) = max seg size, win scale, SACK, etc.

Peter L Dordal, An Introduction to Computer Networks

# TCP IP Header Flags




- **SYN**: for SYNchronize; marks packets that are part of the new-connection handshake.
- **ACK**: indicates that the header Acknowledgment field is valid; that is, all but the first packet.
- **FIN**: for FINish; marks packets involved in the connection closing.
- **PSH**: for PuSH; marks "non-full" packets that should be delivered promptly at the far end.
- **RST**: for ReSeT; indicates "reset the connection" for various error conditions.
- **URG**: for URGent; part of a now-seldom-used mechanism for high-priority data.
- **CWR** and **ECE**: part of the Explicit Congestion Notification (ECN) mechanism.
- **NS**: ECN-nonce for concealment protection

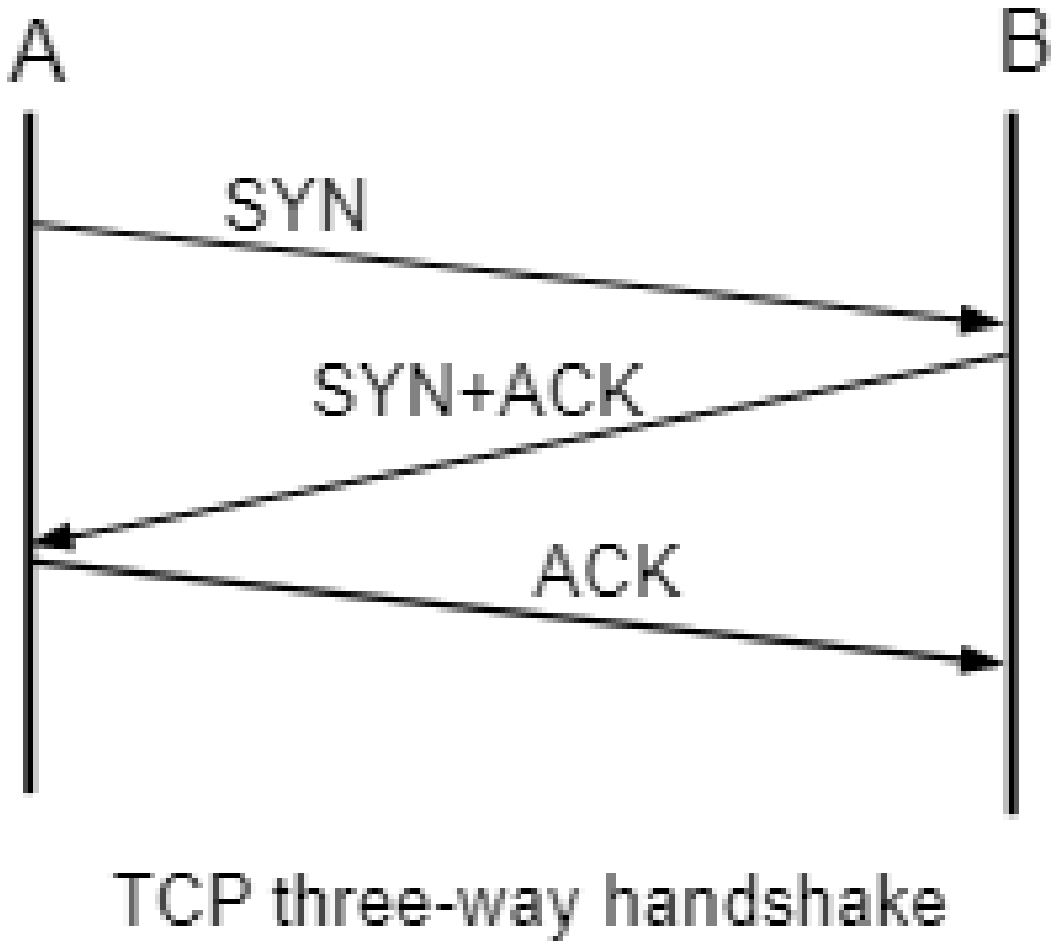
Peter L Dordal, An Introduction to Computer Networks

# TCP 3-Way Handshake

TCP connections are established via an exchange known as the three-way handshake. If A is the client and B is the listening server, then the handshake proceeds as follows:

- A sends B a packet with the SYN bit set (a SYN packet);
- B responds with a SYN packet of its own; the ACK bit is now also set;
- A responds to B's SYN with its own ACK;
- Normally, the three-way handshake is triggered by an application's request to connect; data can be sent only after the handshake completes. This means a one-RTT delay before any data can be sent.

Peter L Dordal, An Introduction To Computer Networks



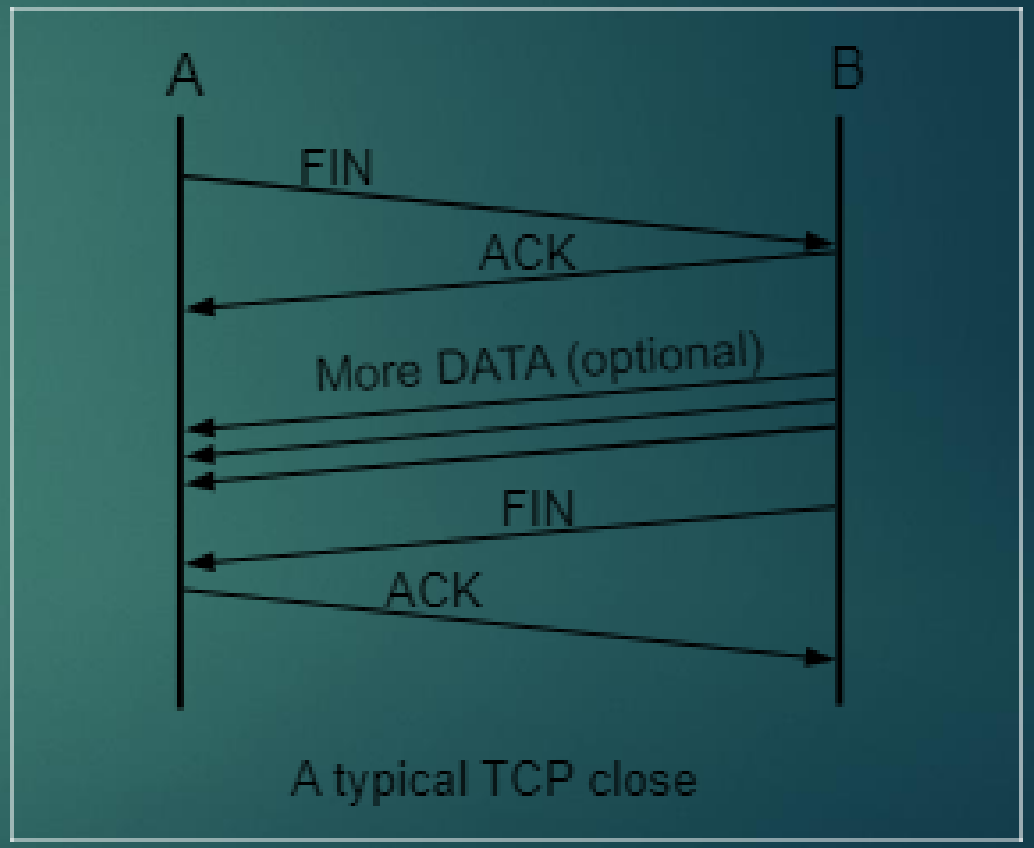

TCP three-way handshake

# TCP 3-way Handshake Exploits

- Syn Flooding = consumes resources on server
- Syn scan (aka port scanning)
  - 3 states can be assessed by results
    1. Ack is sent back indicating the port is open.
    2. RST is sent back indicating the port is closed.
    3. Nothing is sent back indicating the port is filtered (firewalled)

# Closing a connection

- 4-Way handshake
- FIN-ACK (A to B) & FIN-ACK (B to A)

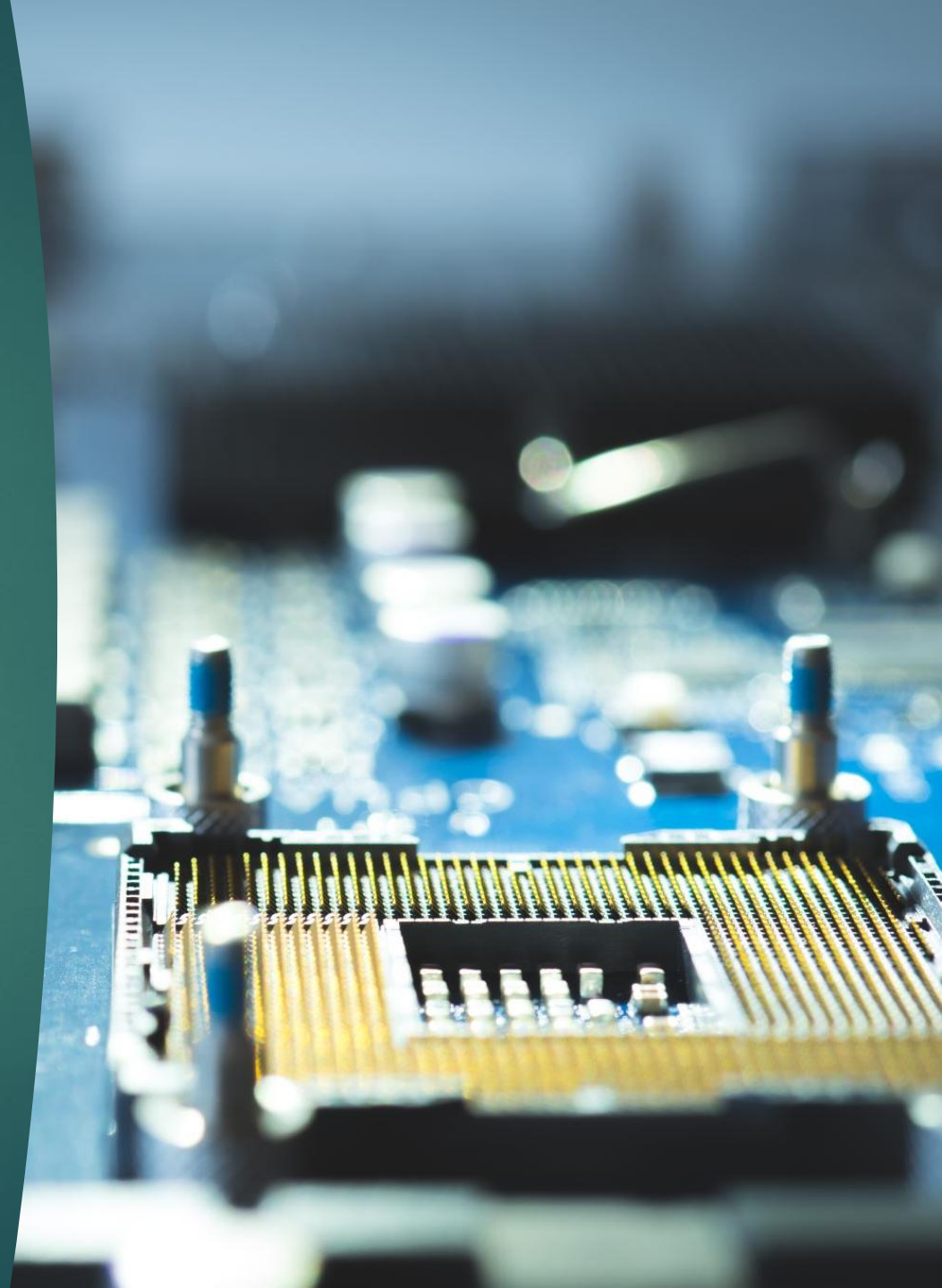



Peter L Dordal, An Introduction to Computer Networks

# TCP & Hardware Assistance

- **TCP Checksum Offloading (TCO)** involves using the NIC to handle checksum calculations which frees the processor from having to do those calculations within the network stack software. This improves efficiency and improves parallelism. It also works for UDP.
- **TCP Segmentation Offloading (TSO)** moves large chunks of data to the NIC in a buffer which then handles breaking the data up into smaller chunks. This approach can be used for improving performance of both sending and receiving.

Peter L Dordal, An Introduction to Computer Networks



# TCP coding examples (server)

```
1.  import socket
2.  print("Launching TCP server")
3.  TCP_IP = "127.0.0.1"
4.  TCP_PORT = 5006
5.  BUFFER_SIZE = 1024
6.  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7.  s.bind((TCP_IP, TCP_PORT))
8.  s.listen(1)
9.  conn, addr = s.accept()
10. print ("Connection address:", addr)
11. a=""
12. while a !="~":
13.     data = conn.recv(BUFFER_SIZE)
14.     a = str(data)
15.     a = a[2:-1]
16.     print ("received data from client:", a)
17.     conn.send(bytearray("got " + a,encoding="ascii")) # tell client what we got
18. conn.close()
19. print("Server stopped")
```

# TCP coding examples (client)

```
1.  import socket
2.  print("Launching TCP client")
3.  TCP_IP = "127.0.0.1"
4.  TCP_PORT = 5006
5.  BUFFER_SIZE = 1024
6.  MESSAGE = ""
7.  try:
8.      s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)   # Internet, TCP
9.      s.connect((TCP_IP, TCP_PORT))
10.     while MESSAGE != "~":
11.         MESSAGE = input("Message? ")
12.         s.send(bytearray((MESSAGE),encoding="ascii"))
13.         data = s.recv(BUFFER_SIZE)
14.         a = str(data)
15.         a = a[2:-1]
16.         print ("received data from server:", a)
17.     s.close()
18. except:
19.     print("socket error - make sure server is running")
20. print("Client stopped")
```

Peter L Dordal, An Introduction to Computer Networks