Briefly on Computer Networks

Troels Henriksen

Networks

Purpose of this lecture

- What is the physical structure of a computer network?
- What are the layered abstractions that form a computer network?
- How do applications communicate across a network?

Only a high level overview — this is an extremely rich area.

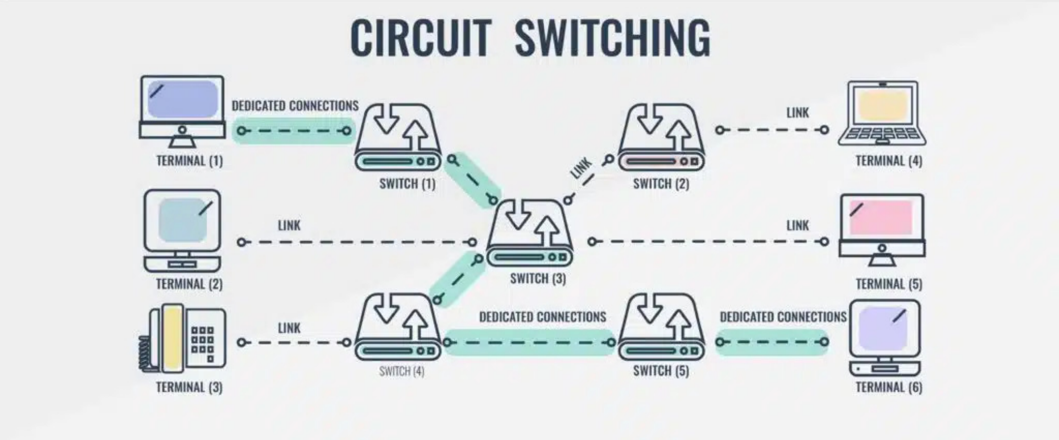
Circuit switching vs packet switching

A **network** consists of **nodes** (called **hosts**) exchanging data.

- With **circuit switching**, two communicating hosts have a dedicated connection.
- With **packet switching**, communication is split up into smaller *packets*, that are *routed* independently through network, with links shared by multiple hosts.

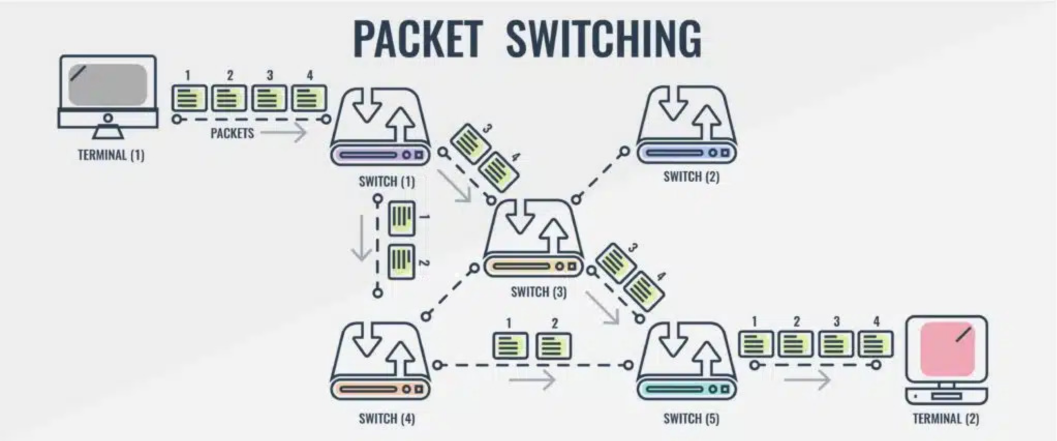
Packet

A finitely sized block of data, with metadata such as sender and destination.



- Traditionally used in phone networks.
- Used for certain high-reliability applications.
- Circuits can be fixed or flexible.

Illustration from `https://www.comparitech.com/net-admin/circuit-switching-vs-packet-switching/`



- Used for the internet and all large networks.

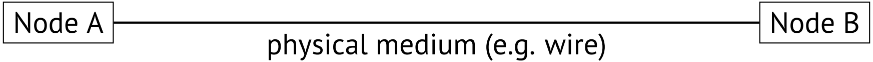
Illustration from `https://www.comparitech.com/net-admin/circuit-switching-vs-packet-switching/`

The network is a stack of abstractions

- **Physical layer:** wires, radios, etc.
- **Link layer:** Locally addressable communication.
- **Network layer:** Globally addressable communication.
- **Transport layer:** Multiplexing, reliability.
- **Application layer:** Application-specific protocols.

The physical layer

- Copper cables, fibre-optic cables, radio waves, laser beams, USB sticks taped to pigeons[1], etc.



Node A

physical medium (e.g. wire)

Node B

- Various ways of handling multiple hosts.
- Various levels of reliability.
- Various packet sizes allowed (or perhaps only single bits).

Software generally does not interact much with this layer.

[1] https://www.rfc-editor.org/rfc/rfc1149

Link layer — using ethernet as example

- Provides packet exchange between hosts in local network.

Network switch

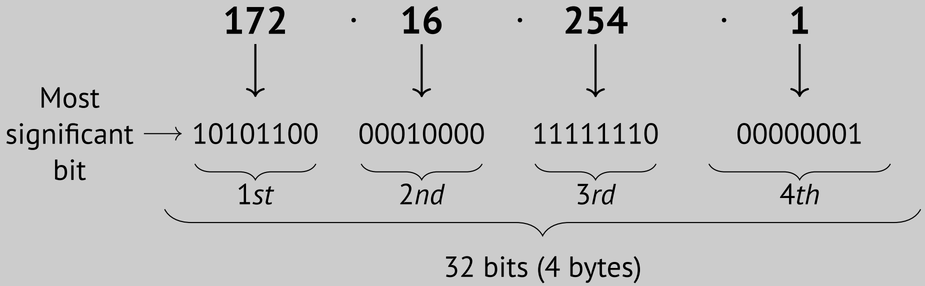
By Geek2003 - Own work, CC BY-SA 3.0, `https://commons.wikimedia.org/w/index.php?curid=18446352`

- Each host on the network identified by *MAC address*, a 48-bit number.
  - Globally unique to each physical network card, e.g: 10:ff:e0:3d:16:7e.
- Packets sent to *specific MAC*.
- Switch knows which MAC is available via each port, and transmits packets out through that port as appropriate.

Network layer — using IP as example

- Provides global routing of packets.
- Each host is identified with an *IP address* which is (almost) globally unique.
- Two major versions of IP: IPv4 (32-bit addresses) and IPv6 (128-bit addresses).
- *Routing tables* tell hosts how to move a packet through the network.

IPv4 address consist of four octets


172 · **16** · **254** · **1**

Most significant bit → 10101100 00010000 11111110 00000001

1*st* 2*nd* 3*rd* 4*th*

32 bits (4 bytes)

IP addresses have structure

- Two IP addresses beloning to the same *subnet* have the same *network prefix*.
- The number of bits allocated to the network prefix is written after a slash.
  - `198.51.100.0/24` has 24 bits allocated to network prefix, and 8 bits for addressing hosts within the network.
- **Not to the degree of memory addresses**, but numeric closeness of IP addresses do have some meaning.
- Routers forward packets between subnets.

IP addresses have structure

- Two IP addresses beloning to the same *subnet* have the same *network prefix*.
- The number of bits allocated to the network prefix is written after a slash.
  - `198.51.100.0/24` has 24 bits allocated to network prefix, and 8 bits for addressing hosts within the network.
- **Not to the degree of memory addresses**, but numeric closeness of IP addresses do have some meaning.
- Routers forward packets between subnets.

The Internet

Interconnected autonomous IP-based networks that exchange packets and routing tables via various protocols, such as the *Border Gateway Protocol*.

- **Nice:** Importantly, no host needs to know everything.
- **Not nice:** It is a complete mess and packets are often dropped, lost, or arrive in unpredictable order.

Transport layer

- IP allows any host to contact another, but...
  - How do *applications* contact each other?
  - What if we care that our packets arrive, and in the right order?
- **Transport-layer protocols** built on top of IP provide these facilities.

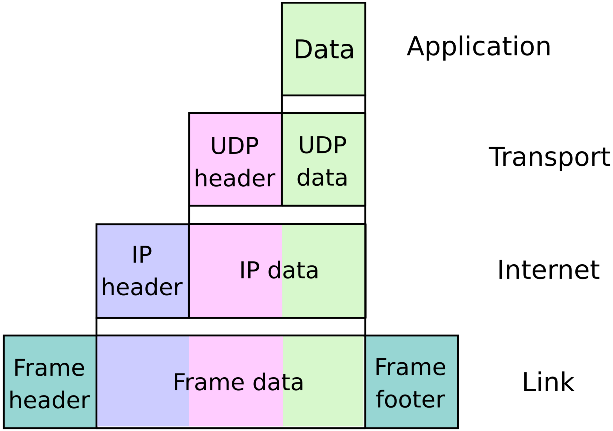
Transport layer protocol: UDP

User Datagram Protocol

Provides connectionless unreliable datagram transmission between hosts, with each datagram market with a target address and *port*.

- A process can register itself as the *recepient* for all UDP packets received on a given port.
- **UDP is unreliable:** packets are not guaranteed to arrive, or to arrive in same order they were sent, and may even be duplicated.
  - But “guaranteed” not to be corrupted *if* they arrive.

The stack in a packet



By en:User:Cburnett original work, colorization by en:User:Kbrose - Original artwork by en:User:Cburnett, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1546338

Listening for UDP packets in Python

```python
import socket

HOST = "0.0.0.0"
PORT = 1337

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((HOST, PORT))

print(f"Listening␣for␣UDP␣packets␣on␣{HOST}:{PORT}...")

while True:
    data, addr = sock.recvfrom(4096)
    print(f"Received␣{len(data)}␣bytes␣from␣{addr}:")
    print(data.decode("utf-8"))
```

Transport layer protocol: TCP

Transmission Control Protocol

Connection-based, stateful, stream-oriented, reliable communication.

- **Connection-based:** A program that wishes to communicate over TCP must create a *connection* over which data is then sent.
  - Still packet-based underneath, but invisible to application.
- A connection can be seen as an arbitrarily-sized file that you cannot rewind.
- The TCP implementation (in the kernel) takes care of resending dropped packets and assembling their data into the desired order, before handing it off to the receiving process.

Listening for TCP connections in Python

```
import socket

HOST = "0.0.0.0"
PORT = 1337

server_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_sock.bind((HOST, PORT))
server_sock.listen(1)

print(f"Listening␣for␣TCP␣connections␣on␣{HOST}:{PORT}...")

while True:
    conn, addr = server_sock.accept()
    print(f"Connection␣from␣{addr}")
    with conn:
        while True:
            data = conn.recv(4096)
            if len(data) == 0:
                break
            print(data.decode("utf-8"))
    print(f"Connection␣closed:␣{addr}")
```

Application layer

- Application protocols are like file formats, and just as diverse and hard to pin down.
- Main ones specified in the IETF-maintained RFCs, such as RFC 9110, which defines HTTP, used by the world wide web.
  - Often used as a “meta-protocol” by embedding other protocols in HTTP, because network firewall may ban other ports.

HTTP 1.1

TCP-based communication over port 80 (443 for encrypted connections), based on requests/responses rather than persistent connections.

Summary

- Computer networks are based on stacks of protocols.
- Not perfect abstractions, more like extensions—we often have to be aware of what is going on beneath.
- TCP provides a remarkably robust and simple interface on top of a chaotic substrate.
- The programming APIs bare based on *sockets*.