# Lecture 1 - Introduction to Communication Networks

Gidon Rosalki

2025-10-19

**Notice:** If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

## 1 Administration

Homework will make up 25% of the final grade, and are done in pairs. There are available 5 bonus points if they are typed. Additionally, 2 interviews will be carried out during the semester, on the homework. Failing to pass the interview results in a failure of the homework in question. There are 2 bonus points available for attendance.

We will mostly follow the book *Computer Networking: A Top Down Approach* (5th of 6th edition) by J. Kurose and K. Ross.

### 1.1 Key concepts

We will study the concepts in networking, such as governing paradigms, key ideas and principles, and fundamental networking tasks. Furthermore, we will also discuss **Domain Specific knowledge** such as how the internet works, specific architectures / protocols, the contents of an IP packet, and "How can a single typo bring down a third of the internet?"

Networking is particularly interested due to the number of unsolved challenges. These include questions such as security and privacy, performance, evolvability (ability to update the networking protocols), and so on.

In this course, we will start with with fundamental concepts, and terminology, and finish by building the internet from the ground up.

## 2 Experience

Let us consider a situation of streaming video. It has been established heuristically that a user will leave if a video does not begin playing within 2 seconds. This is a very short period of time, and does not allow us to download an entire video to the user's computer before beginning. To try and resolve this we have **Dynamic Adaptive Streaming over HTTP** or DASH. This requests chunks of the video at specific bitrates, allowing us to dynamically change the requested bitrate according to the bandwidth of the connection. Overall, we want to ensure a high **Quality of Experience (QoE)**: The network provides user with a level of performance needed for the desired function. For example, a phone call with 100ms of delay results in a very bad user experience, as you are unsure if the other person is pausing, or has finished speaking.

Depending where you are in the world, bandwidth limitation can be significant. For example, Utah to Berlin would take 2 days, 22 hours, and 46 minutes to transmit 100GB of data, where flying would only take 14 hours and 5 minutes.

The internet has become a massively important resource, between conversations, video streaming, information searching, and social media. Despite this, the modern internet is surprisingly fragile, with suboptimal performance, insecure protocols, and unpredictable problems. New challenges keep growing. We saw massive increase in demands on bandwidth in the last 2 decades with the advent of video streaming, and requirements for greater ability to handle slow connections despite requiring high bandwidth with the proliferation of mobile phones.

We consider the internet to "Only Just" work. We have many applications, and developing them is quick and easy. Between Zoom, Facebook, Google, YouTube, this space has rapid technology changes. However, the internet protocols upon which these sit are stagnant, and their methods for routing, congestion control, naming and so on have not changed in decades. However, the technologies upon which these sit, the computers, routers, and so on, have constant innovation. We **want** better internet protocols, and slow change is (finally!) on the horizon.

Consider transmitting information. To transmit information from Jerusalem to New York, the speed of light ensures that it will take at the very least 30.55ms to send a single bit. How much longer depends on the route, the propagation speed of the links, the transmission rate of the links (bits / sec), the number of hops (processing delay), and the congestion. In practice, it generally takes at least 70ms.

Let us consider, how many cycles does the PC execute before it can possibly get a reply? Since the round trip takes $\geq 140ms$, and a PC runs at (approximately) 4GHz, we get approximately $4.2 \times 10^8$ cycles, which is a *very* long time. So, communication feedback is always dated, and is fundamentally asynchronous. Even between machines that are directly connected, we can expect a latency of around $200\mu s$, or around 600,000 cycles, which is still a very long time.

# 3 Computer networks

The ultimate goal of a computer network is to transmit data between end users.

1. End user: These are generally hosts, terminals, stations, phones
2. Links: These connect between nodes, and can be wireless, optical fibre, copper, and so on
3. Nodes: Routers, switches, are the points within the network through which information flows

There are many requirements from a network:

- Size of transfers
- Bidirectionality (or not)
- Latency sensitive (or not)
- Tolerance of jitter (or not)
- Tolerance of packet drop (or not)
- Need for reliability (or not)
- Multicast (or not)

## 3.1 Computer networks vs Distributed systems

A **Computer Network** is the infrastructure and technologies needed to transmit data between hosts.
A **Distributed System** is the set of hosts using the computer network.

The network itself does **NOT** produce data (aside from control data to manage the network).

# 4 Telephone network and the internet

The telephone network uses circuit switching. Let us consider how this system works:

1. Establish: The source creates a circuit to the destination, and nodes along the path store connection information. Resources are reserved for the connection, and if the circuit is not available, then a "busy signal" is produced.
2. Transfer: The source sends data over the circuit. There is no destination address, since this is just a closed circuit. There is a continual stream of data.
3. Teardown: The source tears down the circuit when done, freeing resources, and breaking the circuit.

This kind of circuit switching involves physically connecting, and disconnecting wires to make circuits. This was originally done with a human operator, where you would ask them to connect you to a particular line, and they would physically move the wires.

In 1889, Almon Brown Strowger invented the *mechanical switching system*. This allows removing the human operators by having the line transmit information to the mechanical system, which would then automatically connect the lines.

## 4.1 Timing

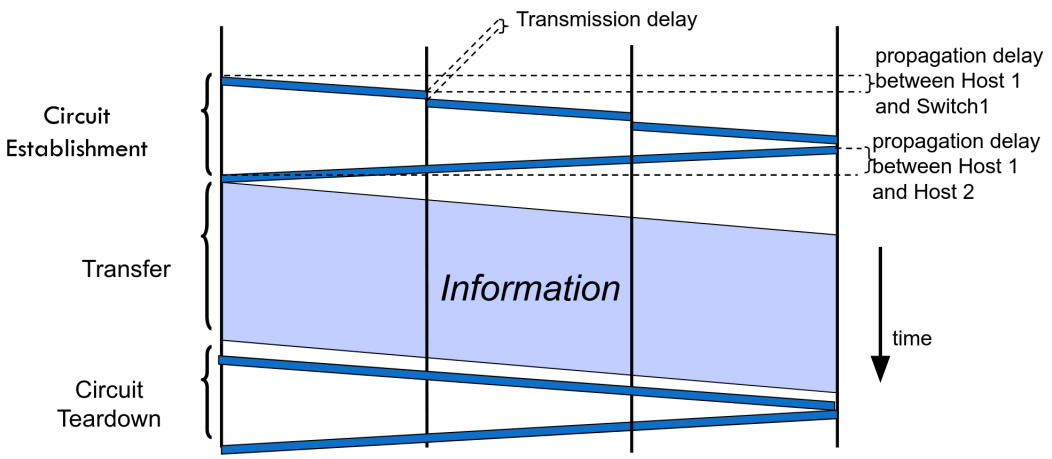


Figure 1: Timing

## 4.2 Multiplexing

All this talk of having a single circuit per link leaves us with the question of how do we share a link? We achieve this though **multiplexing**, and there are a couple of methods to achieve this. We may have each circuit allocated to certain time slots, switching between them too quickly for the humans at either end to notice. However, this means that all users have to be well synchronised. We will also need a buffer to ensure that we do not run out of data before the turn of a particular user comes up again.
We may also use frequency division, where every circuit is given different frequencies for transmission of data. This way, they do not interrupt each other, but since we can only use a section of the available data pipeline, the bitrate has been significantly limited.

In our following examples, we will stick to time division multiplexing, for simplicity.

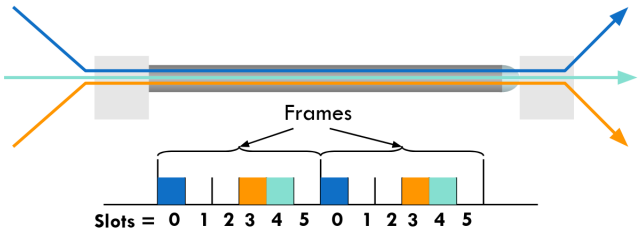


Figure 2: Timing

We divide the time into frames, and frames into slots. The relative slot position inside of a frame determines to which conversation the data belongs. For example, each slot 0 belongs to the blue conversation. However, this requires synchronisation between the sender, and the receiver. We also need to dynamically bind a slot to a conversation. If a conversation does not use its circuit, then capacity is lost.

This sort of system has several strength. It has predictable performance, with known delays, and no drops. However, it comes with a few weaknesses:

1. It is not resilient to failure: Any failure along the path will prevent transmission. The entire transmission would then have to be restarted. We call this sort of delivery model “all or nothing”

2. Wastes bandwidth: We have a fixed number of slots within each frame. Having fewer uses than slots results in wasted bandwidth, with unused slots. Consider a network application with the peak bandwidth P, and the average bandwidth A. In order for the application to work, it must reserve the peak bandwidth, but then the resulting level of utilisation (average throughput) will be the ratio $\frac{A}{P}$. Some applications have relatively small P/A ratios. Voice over IP might have a ratio of 3:1 or so. Data applications tend to be rather bursty, resulting in ratios of 100 or greater to be rather common. Thus, circuit switching is too inefficient for bursty applications. Generally 2× performance penalties are considered acceptable, but when this gets to orders of magnitude larger, this is too much.

3. Design is tied to the application: This design revolves around the requirements of voice communication, and is not a general feature of circuit switching.

4. Setup time: Every connection requires at least the time of a round trip to set up, which results in a very significant penalty in short transfers, where there is little information to be transmitted.