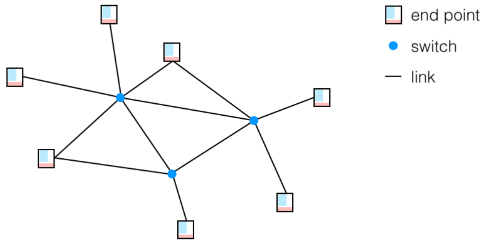
MIT 6.033 2020
Ten-minute Networking Extravaganza

> *This is not meant to be a comprehensive guide to networking. It contains the very basics that you need to know to understand how a network works; some terms have been intentionally defined imprecisely, and many details have been left out. We'll fill in a lot of those details, and make a lot of definitions more precise, as the course goes on.*

A computer network facilitates communication between two or more computers. You can imagine having, say, ten computers, all connected to one another with Ethernet cables. That's one example of a computer network. We would say that each of those machines had a direct connection to one another, or a **point-to-point link**. These point-to-point links don't have to be Ethernet cables; they could be 802.11 wireless links, satellite links, cellular links, etc. There are many different **physical** technologies that one can build a network out of.

It isn't feasible to have every computer in the world directly connected to every other computer. Instead, most networks involve **switches**. Switches are devices that sit in the middle of the network, and help move data from one machine to another.



Typically we'll refer to machines such as your computer as "endpoints", to differentiate them from switches. Usually, endpoints initiate the sending of data, and data is sent to them; switches help move data through the network. As you can see, data from multiple different endpoints might pass through a single switch.

There are multiple ways to handle this — multiple ways to send more than one stream of data through a switch (to **multiplex** the network). The most common way is to use **packet switching**. In a packet-switched network, each piece of data — a **packet** — contains some meta-information attached to it, called a **header**. This header contains things such as the source address: the address of the machine where the packet is headed. As the packet passes through different switches, they can look at the packet header and figure out where the packet needs to go.

> A contrasting idea is **circuit switching**, where the relevant state is kept on switches instead of attached to teach packet.

To actually get data from one place to another, a network first needs to make sure every endpoint has a unique **address**. Depending on the size of the network, the addressing scheme may have to be clever in certain ways; for small networks, it's fine to just image that a centralized service gives every endpoint a unique address.

In addition to addresses, a network needs a **routing protocol**. That is a protocol that helps it find *routes* through the network. Once a switch has run a routing protocol, when it gets a packet destined for Machine A, it knows which switch to send the packet to next.

With addressing and routing, we now have a network that works, at least to an extent: endpoints can send data to one another. Importantly, addressing and routing protocols don't need to worry about what physical technology is running "underneath": a routing protocol can work over an Ethernet network, a wireless network, etc. This is thanks to **layering**: the **physical layer** need not be concerned with what the **network layer** is doing, and vice versa.

Each switch is only capable of handling so many packets at once. Switches contain **queues**, which store packets that they've received but cannot send yet (because their outgoing link is busy sending other packets). If a packet arrives when the queue is full, that packet will be **dropped**; it disappears from the network. Full queues are one reason that a network might experience **loss**. Other reasons are if a switch crashes or if a link fails (e.g., an Ethernet cable gets cut).

For some applications, this type of loss is okay; others might desire **reliable transport**, where more effort is made to deliver data reliably. **Reliable transport protocols** provide reliable — but not perfect — transport. These protocols, and many others, are part of the **transport layer**, which sits above the network layer. That implies that transport layer protocols don't care what routing protocol the network is using, e.g.

This discussion also reveals yet another layer: the **application layer**. Ultimately, applications — like Skype, Spotify, or other applications that you might run — generate the network traffic. As revealed, different applications desire different things.

Besides reliability, there are other ways to think about how the network performs:

- How much data can it transport over some unit of time? There are many ways to think about this: what is the physical speed that the underlying links are capable of? How much data is the network *actually* sending (say, per second), which is based at least partly on how much traffic the endpoints are generating? Etc. There are lots of terms you'll see used to describe these concepts: the **speed** of a network, its **bandwidth**, its **throughput**, etc.

- How long does it take a packet to get from one end of the network to the other? This quantity is referred to as **latency**. The latency of a network is affected by its physical properties, as well as how much other data is in the network.

There are a number of other things we can ask of our networks:

- Does it provide **multicast**: the ability to send a piece of data to multiple endpoints at once, instead of just one? The implication there is that the underlying network replicates the data where appropriate; the sender sends a single copy, while somehow specifying the endpoints that should receive it. If a network provides the ability to multicast to all endpoints at once (sometimes, all in-range endpoints), we say that network provides **broadcast**.

- Does it allow for **prioritization**, where different types of data are treated differently? For instance, perhaps a switch prioritizes HTTP packets, sending those before all others.

- Does the network support any other **policies**? Usually, those policies are set by a human, not automated by the network. Prioritizing certain types of data is one example of a policy.

In this discussion, we've presented a four-layer model: physical, network, transport, and application. One of the most widely-used models actually uses seven layers: a "link layer" sits between the physical and network layers, and the "session" and "presentation" layers sit between the transport and application layers. There's no need to worry about the details of those layers at the moment.