In the previous lecture, we discussed the five-layer model of networking. In this lecture, we will discuss four topics. First, we’ll look at the original goals for the Internet when it was first designed. This is useful because it’ll help us understand why the Internet is the way it is. Second, we’ll look at two approaches to building networks: packet and circuit switching. We’ll explain why the Internet picked packet switching given its goals. Third, we’ll look at the five layers in a bit more detail and understand where (host or router) each layer is implemented in the Internet, again given the goals of the Internet. Fourth, we’ll conclude by discussing how we measure the Internet’s performance.

1 The Internet’s original goals

1.1 Why care about the Internet’s original goals?

The first quarter of today’s lecture might be a bit abstract, but it’s useful to try and vaguely understand it so that the rest of the course makes sense. Things will become more concrete as we proceed through the rest of the lecture and the rest of the course. Let’s begin with the definition of the term internet. The term “internet” (short for internetwork or internetworking) refers to any network of interconnected networks. The term “Internet” (with the capital I) refers to the global system of interconnected networks that we use in our day-to-day lives.

What kinds of networks does the Internet interconnect? Examples of such networks are all around us. For instance, the WiFi network shared by members of a household, a building-wide Ethernet network such as the one connecting desktops in this building, a high-capacity private network connecting together multiple geographically distributed sites of a company, or a satellite network used by ships to provide Internet access.

Understanding the requirements that originally drove the development of the Internet is useful because it tells us why the Internet is the way that it is. Throughout the course, it will be useful to keep in mind these original design requirements. Once I tell you what the original requirements were, it will be educational to ponder two questions as you proceed through the course: (1) are any of the original requirements for the Internet no longer as important as they once were?, and (2) are there any new requirements today that necessitate a different design from the original Internet?

1.2 The goal of low-effort interconnection and the end-to-end principle

The key design goal/requirement of the Internet was to somehow connect together a bunch of existing networks with the lowest effort possible. The thinking was that this would broaden the reach of the Internet from a set of local networks to one large global supernetwork, where any computer could reach any other computer. To provide this global network, the designers of the original Internet envisioned a set of gateways that would glue together existing networks at their peripheries.

\footnote{This distinction might soon disappear. The New York Times decided to lowercase the I in Internet recently, arguing that the Internet had become as ubiquitous as electricity, water, or the telephone. I’ll use Internet with the capital I because of muscle memory.}

\footnote{Gateways are also called routers. The term switch is used to refer to an interconnecting device within a local network—as opposed to between two networks. We’ll use the term gateway and router interchangeably.}
1.3 The goal of generality and the idea of layering

To ensure low-effort interconnection, the requirements for the gateways were kept to a bare minimum. This bare minimum requirement was forwarding data from one network to another. Anything else that could be implemented at the end hosts was implemented at the end hosts. In other words, functionality was implemented on the routers only if there was no other alternative. Even in such cases, functionality was implemented on the routers only if the router implementation was sufficient to cover all corner cases of that functionality.

Let me illustrate this with an example. Let’s look at the functionality of reliable data transfer: ensuring that data from a sender is delivered to a receiver reliably despite data losses in the network. This requires retransmitting any data that is dropped in the network. This can be done one of two ways.

The first is end-to-end reliability. Here, the sender can perform an end-to-end check by having the receiver acknowledge receipt of every piece of data and then retransmitting anything that was lost. The second is link-by-link reliability. Here, each router on the sender-to-receiver path could reliably forward data to the next router on the path, by having the next router acknowledge receipt of any forwarded data. This way, the sender can forget about the data once it has been acknowledged by the first router in the sender-to-receiver path.

Reliability was and continues to be implemented largely on end hosts in the Internet. Why? First, routers don’t need to be involved. You can implement the requisite checks for reliability (i.e., did this byte get successfully delivered to the other end) by having the receiver acknowledge every byte to the sender. Second, a router check by itself is not sufficient. This is because the router might lose power, the data might be dropped at the link to the receiver, or the data may be corrupted in the receiver’s memory before it is delivered to the receiver application. The only way to take care of all of these bizarre corner cases is for reliability to be implemented end-to-end using acknowledgements from the receiver.

This example of minimalism in routers is the essence of what later came to be called the end-to-end principle: if you can do something at the end hosts, you should do it there without burdening the routers.

1.3 The goal of generality and the idea of layering

The original designers of the Internet also did not know what applications would run on the Internet or what networks it would interconnect. As a result, they designed the Internet to be as general purpose as possible. So the Internet architecture ended up being general and good enough for most things, but not the best choice for anything.

The 5-layer stack we discussed in the previous lecture is one example of this generality. Each layer is as general as possible to accommodate a variety of requirements. Let’s see how each layer does this.

The application layer can accommodate anything from AI to video conferencing to web browsing to social networking so long as these applications send and receive data using a uniform interface to the transport layer. This uniform interface is the Sockets API, which we’ll be covering next lecture. None of the other layers encodes any application-specific information in it, e.g., the routing layer doesn’t specifically cater to or optimize for AI applications on the Internet.

One layer below, the transport layer, only worries about end-to-end delivery (e.g., if the application needs reliability, the transport provides it) between a sender and receiver application without worrying about how packets get from the sender to receiver. The routing layer, by contrast, does not worry about end-to-end delivery of data. It only worries about how packets get from one host/router to another host router using unreliable forwarding between one router and the next. It is also focused on global delivery of packets, i.e., from a router in California to a router in NYC. But, it does not worry about the local aspects of networking: getting data from your AP or switch to the laptop or desktop.

The local aspects of networking are taken care of by the link layer because there could be a variety of local networks (e.g., WiFi, Ethernet, Bluetooth, cellular, FIOS, etc.). It’s challenging for the routing layer to be

---

3 The end-to-end principle does allow for functionality to run on routers as a performance optimization, so long as it is not required for correctness. For instance, on some router links where the error rate could be high such as a cellular link, having the router automatically retransmit some dropped packets might be desirable because the retransmitted packets don’t need to traverse the whole network.

4 There is an April Fools’ joke about how the Internet can run over anything, including carrier pigeons: [https://tools.ietf.org/html/rfc1149](https://tools.ietf.org/html/rfc1149)
cognizant of all of them. Finally, the link layer does not worry about physically encoding these bits as voltages on a wire or an antenna. That’s the job of the physical layer.

1.4 Non-goals when the Internet was designed

Because low-effort interconnection and basic global connectivity were the overarching goals, many other goals were sidelined. In particular, while this seems a bit hard to believe, cost effectiveness was quite low on the priority list of goals, and good network performance and network security were not on the list at all. If you want to read more about this, David Clark’s paper [2] has a good discussion of these goals.

Now if you were running your own network, you may have different goals. You may want your network to give you the best data transfer rates possible (we’ll formalize this at the end of this lecture). Or you might want to guarantee that your request gets to the other end of the network as quickly as possible (again, we’ll formalize this at the end of the lecture). Or you might want your network’s performance to be predictable and not variable, even if the performance is poor.

It is important to keep in mind that these were not the Internet’s goals, and if they were, the protocols and the Internet’s overall design may have turned out differently. In fact, a good amount of networking research today is about how to develop specialized network designs that are different from the Internet, when the network’s requirements are different from the Internet’s original requirements (e.g., the network within a large company).

2 Packet vs. circuit switching

Now that we have described the requirements that drove the design of the Internet, let’s see how the networks within the Internet are implemented. Broadly, there are two approaches to building a network: packet switching and circuit switching. We’ll describe both below and then explain why packet switching is a better choice for the Internet given its original goals.

2.1 Circuit switching

Historically, telephone networks operated around the idea of circuit switching. In a telephone network, before a conversation started, there would be a call setup phase that would establish an electrical connection between the two parties all the way from the caller’s handset to the callee’s handset. This connection would make its way through multiple telephone exchanges. This was initially done manually through telephone operators, who would plug in various cables into sockets and turn on different switches to connect different telephone exchanges together at the electrical level. This process, called switching, was later done digitally.

Once the call was setup, the two parties could communicate until the call was explicitly torn down, usually when one of the parties put their phone down. While the call was active, the conversation could proceed at whatever raw link capacity was supported by the underlying electrical wires that made up the electrical connection between the caller and the callee.

This model was a good fit for telephony, which demanded a constant bit-rate communication channel of a few kbit/s, which was achievable by electrical wires of the time. It also provided isolation between different conversations, because each conversation had a separate electrical connection for itself and hence did not interfere with other conversations.

However, circuit switching was not particularly suited to anything other than voice calls. Most computer programs that communicate with other computer programs are much more bursty in their communication patterns: they send a short burst of information, followed by a relatively large period of silence. For instance, when using a web browser, the web browser communicates with the external world when the user navigates to a new link, but is dormant between user interactions.

In such cases, circuit switching isn’t efficient. Imagine setting up a dedicated electrical connection between your web browser and a server. First, this connection would be idle most of the time, except for when you
2.2 Packet switching

In response to these concerns, the 1960s saw the development of packet switching. The idea behind packet switching was to divide a piece of data into small units called packets that could be independently routed over the network between the source and destination. To allow packets to be routed independently, each packet carried a header that specified where the packet was destined.

Packet switching fixes the three problems with circuit switching outlined before. First, packets from different connections could be transmitted one after the other on the same link. This improved network utilization because it was unlikely that two bursty applications wanted to transmit data at the exact same instant. It was more likely that the busy period of one application coincided with the idle period of another, thus allowing the link to be more fully utilized (Figure 1). Second, because the packet header identified the destination address, each packet could be transmitted without having to set up a connection first. Third, if a link (or router) failed, packets could simply be detoured along a different network path, so long as the network had some spare paths available. This was unlike circuit switching, which either needed links/routers that seldom failed, or needed connections to be setup again on every failure.

But packet switching has some performance penalties. First, it takes slightly longer for a message to get from a sender to a receiver in packet switching because each router needs to determine the next router for a packet by inspecting the packet’s header. In contrast, this choice of next router is electrically hardwired during connection setup in circuit switching. Second, performance is no longer predictable. If multiple packets from different senders end up at the same link at the same instant, then the packets need to be buffered at the link, leading to a build up of queues at the link (Figure 2). When a queue builds up, a packet’s delivery is delayed depending on the number of packets ahead of it in the queue.

---

6We use the term routed to refer to the process by which data finds its path of routers from the sender to the receiver, similar to how cars are routed on a road network.

7One reason why this can happen is because the bursts from two different applications happened to coincide. This could occasionally happen, because the bursts from different applications being mostly out of sync is something that is expected, but not guaranteed.

8We are assuming a first-in first-out policy for servicing packets from a queue.
2.3 Why the Internet uses packet switching

Ultimately, for the Internet, the pros of packet switching outweighed its cons given what the Internet cared about. The Internet was not designed for performance or performance predictability; it was a best-effort network, which is essentially code for “all bets are off.” On the other hand, the ability to survive router failures through rerouting was important. The circuit switching alternative of making the routers reliable enough to rarely fail (something that the telephone companies did very successfully), would have increased the requirement on routers, which in turn would have raised the barrier to entry into the Internet.

It is important to reiterate that when the Internet was developed the goal was to get bare minimum connectivity between hosts on the Internet and to broaden the reach of the Internet by making it as easy as possible to join the Internet. Everything else was secondary. It is likely that focusing on a different set of goals may have led to a different design. But it’s also likely that the Internet may have never taken off if the barrier to entry was higher than what it was. Sometimes, it just makes sense to build something, make it as easy to use as possible, and then iterate from there as required.

3 Where are the five layers implemented?

Each layer is characterized by a set of protocols that run at that layer. A protocol is a restrictive language that allows different entities to talk to each other without ambiguity. Let’s start at the top and see where each layer is implemented in the Internet (Figure 3). First, the application layer. The predominant protocol that runs at the application layer is HTTP. The protocol allows you (the client/user) to specify which URLs you want and allows the server to respond with the web page at that particular URL.

3.1 The application layer

Where is the application layer implemented? The only place that knows which URL you want is your web browser. So that’s the logical place to implement it: on your web browser within your end host. If you had a video conferencing application, such as Skype, it might use a different application layer protocol, just because its needs are different, but it would still be implemented within the video-conferencing application on the end host.
3.2 The transport layer

Let’s move on to the transport layer. As I told you last class, the transport layer provides reliable, in-order delivery. This is implemented by a protocol called TCP (or the Transport Control Protocol). The transport layer also provides an unreliable service called UDP (the User Datagram Protocol), where you can send messages (called datagrams) instead of bytes, to another end host, but there is no guarantee on whether a message would be delivered or not.

Why might you want two services? Typically, reliable, in-order delivery carries a delay penalty. If you drop a particular byte, you need to wait for that byte before delivering the bytes following that byte to the application. For a live streaming application, that delay might be more problematic than an occasional dropped frame. In other words, I am happy not seeing the game for a split second because of a lost byte/frame, because I’ll barely notice it. But, when I do see the game, I want it to be the latest version of it.

OK, so where are TCP and UDP implemented? They are implemented in your operating system’s kernel. Why? Because they are useful to many different applications such as different browsers, different live-streaming programs, and so on. Why not implement them on the router? UDP is easy to answer: because it has no requirements, the router doesn’t need to handle it specially, so it is trivially “implemented” on routers so long as they forward packets. But for TCP, if the router started providing reliability, it would be a case of too much effort on the router for little reward. The end host has to implement reliability checks anyway, so the router’s implementation is not particularly useful.

3.3 The routing layer

The next layer is the routing layer. The predominant protocol here is IP (or the Internet Protocol), which was first developed in 1974. Though it has evolved over the years, many of its essential features have remained the same, and it is an impressive example of designing for generality, when you don’t know your use cases. The IP protocol is the glue that connects together the whole Internet and is the only component of the five-layer stack that needs to be understood by every device that’s on the Internet. Everything else is device specific: the transport layer only runs on end hosts, the application protocol is different for each application, and the Internet has a variety of physical and link layers (e.g., WiFi and Ethernet).

The IP protocol specifies the source and destination address (or the IP addresses) of a particular piece of data. The end hosts need to participate in the IP layer because they insert the source and destination addresses. Unlike the top two layers, the routers and gateways also need to implement and understand the IP protocol because they need to forward data from one network to another. How do they perform this forwarding? By using the IP destination address and looking it up in a table that tells them which link to send the packet out on.
3.4 The link and physical layers

The next is the link layer, which is the local part of the Internet. It takes care of getting data from your desktop to its Ethernet switch, or from your laptop to its WiFi access point (AP). Whether its WiFi or Ethernet, both ends need to participate in sending and receiving data between the end host and the switch or AP. In more detail, this layer is implemented in the hardware and firmware of your Network Interface Card (NIC) and in the hardware and firmware of your switch/AP.

Finally, we are left with the cable/antenna, which is the physical layer. Again, because both ends of cable/antenna need to cooperate to transfer information (e.g., to synchronize on when bits on the air/cable begin), the physical layer is implemented both in your NIC and your switch/AP.

4 How do you measure the Internet’s performance?

Once you get over the initial wonderment of being able to transmit information between two computers, you start worrying about whether the network can do more than send “hi” or “hello” between two computers. In other words, you move beyond whether the network can do something to how efficiently it can do it: network performance. We’ll use two primary metrics to measure a packet-switched network’s performance. Both these metrics are proxies for what an application truly cares about: good user experience.

4.1 Throughput

The first metric is throughput. This metric measures the amount of data transmitted or received per unit time. Throughput is specific to a particular layer. For instance, the raw physical link may be capable of transmitting 10 Gbit/s. But, after accounting for the space taken up by the MAC layer’s headers (for MAC source and destination addresses, etc.), the MAC-layer throughput, which is defined as the number of MAC layer payload bytes transmitted or received per second, will be lower than 10 Gbit/s. How much lower depends on the size of the packets. For instance, if each packet had only 1 byte of MAC payload, and carried the standard 18-byte MAC header, the MAC-layer throughput will be only \( \frac{1}{1 + 18} \) of the raw physical link capacity because of the overheads of the packet headers. However, if each packet carried the largest payload possible (1500 bytes), the MAC-layer throughput will be \( \frac{1500}{1500 + 18} \) of the raw physical link capacity, which is much higher. We’ll always report throughput at a particular layer.

Throughput for a reliable in-order transport layer like TCP might be lower still because the bytes have to be received in the correct order for them to be useful because the application running on top expects in-order delivery. So, even if 1000 bytes are received per second, but the second and third bytes are reordered, the throughput is only 1 byte per second.

4.2 Latency

The second metric of interest is latency, which reflects the time between when a packet is sent at a sender and received at the receiver. Again, depending on where in the 5 layers the packet send and receive times are measured, the packet latency might be different.

4.3 Do applications care about throughput or latency?

Applications may care about throughput or latency or both. An application that cares about throughput, but not latency, is a file download application. If you’re downloading a large file, you care that you eventually receive all bytes of the file in the shortest overall time possible. It is not important if a particular packet containing a particular set of bytes got to you earlier or later so long as the overall time is minimized. On the other hand, if you place a voice call over the Internet, the throughput requirements are rather modest: any network supporting

---

9Your switch or AP needs to know which end host to send its data to. So both the WiFi and Ethernet packet formats carry a header called the MAC address to identify the end host or switch/AP. Unlike the IP address that is globally meaningful over the entire Internet, the MAC address is a local concept that is local to a WiFi or Ethernet network.
a few tens of kbit/s of throughput will do. But, it is important that what you say gets to the other side as quickly as possible. If not, the interactivity of the conversation is destroyed. An application that cares about both is interactive video conferencing. Ideally, you want to make sure that the video (or audio) frame you see/hear on your screen is not too behind the actual video (or audio) frame of the person you are conferencing with (latency). But, you also want to support high-quality video, which needs more bytes per second (throughput).

4.4 Performance differences between packet and circuit switching

In a packet-switched network, latency and throughput are both variable quantities that are affected by the number of applications that are sharing the network. For instance, if two applications send a burst of packets into a link at the same time, the link will be forced to buffer some of these packets, leading to an increase in latency. Similarly, if an application is sending traffic at 9 Mbit/s into a 10 Mbit/s link, it leaves only 1 Mbit/s of remaining capacity for the other applications.

Network performance is one area where there is a big difference between packet and circuit switching. In circuit switching, throughput and latency are guaranteed quantities that are known when the connection is setup. In packet switching, these quantities are variable depending on how many other applications are sharing the network, when they transmit, and which network paths they take. On the positive side, packet switching provides better utilization. Essentially, with packet switching, you are giving up performance determinism (or worst-case performance) for better average-case performance.

References
