Lecture 2: Internet overview

Anirudh Sivaraman

2018/10/16

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 required because it provides the underlying context for the rest of the course. 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.[1]

First, what kinds of networks does the Internet interconnect? Examples of such networks are all around us. I have provided a few examples below; the names used to informally refers to these networks are provided in parentheses.

1. The WiFi network shared by members of a household (a residential network)
2. A building-wide Ethernet network such as the one connecting desktops in a university or a small company (an enterprise network)
3. A much faster and larger building-wide Ethernet network within Web companies such as Google, Amazon, Facebook, and Microsoft (a datacenter)
4. A high-capacity private network connecting together multiple such geographically distributed datacenters (a private wide-area network or private WAN)
5. A satellite network used by ships to provide Internet access

Understanding the requirements that drove the development of the Internet in the mid 70s is useful because it tells us why the Internet is the way that it currently is. Throughout the course, it will be useful to keep these original design requirements in mind. 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] This distinction might soon disappear. The New York Times decided to lowercase the I in Internet recently [1], 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.

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

The key design goal of the Internet was to easily connect together several existing networks. The thinking was that this would broaden the reach of networking from a set of existing local networks to one large global supernetwork (the Internet), where any computer could reach any other computer. To provide this global network, the designers of the original Internet envisioned a set of *gateways*[2] that would glue together existing networks at their peripheries.

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. For any other functionality, implementing it at the end hosts was the preferred option. This guideline came to be called the *end-to-end principle* in retrospect. Put differently, 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, whatever be the reason (e.g., a router failure, a router misconfiguration, a misbehaving receiver, or an overloaded receiver). Reliability can be achieved in 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 any data that was lost. The second is link-by-link or hop-by-hop 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; essentially, the first router is obliged now to reliably deliver data to the receiver.[3]

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 receiver once it has been acknowledged to the last router, 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 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.[4]

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.[5]

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.

[2] Gatways are also called routers today. 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 terms gateway, router, and switch interchangeably. The specific type of device we are referring to should be clear from the context.

[3] We assume there is at least one path in the network from the sender to the receiver. It is impossible to achieve reliability otherwise.

[4] 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 (e.g., a 3G cellular link), having the router automatically retransmit some dropped data might be desirable because it would avoid having retransmitted data repeatedly traverse the whole nework.

[5] There is an April Fools' joke about how the Internet can run over anything, including carrier pigeons: `https://tools.ietf.org/html/rfc1149`.

The application layer can accommodate anything from machine-learning-based applications (e.g., web search, Siri) to video conferencing (e.g., Skype, Hangouts) to web browsing (e.g., Chrome, Firefox) to social networking (e.g., Facebook)—as 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 Internet's routing layer doesn't specifically cater to or optimize for video conferencing applications.

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 a receiver application without worrying about how the network gets data from the sender to the receiver. The routing layer, also called the network layer, by contrast, does not worry about end-to-end delivery of data. It only worries about how data gets from one host/router to another host/router using unreliable forwarding beween one router and the next.

The routing layer is also focused on *global* delivery of data, 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 WiFi access point (AP) or switch to your 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 cognizant of all of them (e.g., some are wired and others are wireless, leading to different methods of local delivery). 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.

1.3.1 How does layering relate to generality?

Each layer permits a diversity of protocols, applications, communication technologies, cable types, or algorithms within that layer—as long as the layer respects interfaces to adjacent layers. In other words, by not imposing too many restrictions within a layer, the Internet promotes a diversity of implementations and serves as a general-purpose platform for unanticipated use cases. At the risk of repeating myself, let's look at a few examples from each layer.

1. Applications only need to send and receive data using the sockets interface (we'll discuss this next lecture) to respect their interface to the transport layer. Beyond this, they are free to use their own application logic to decide when to send and receive data, how much data to send or receive, and how this data is interpreted at the other end (e.g., HTML, audio, video, virtual reality, etc.).

2. The transport layer needs to produce data at end hosts with a destination address to respect its interface with the routing layer. Beyond this, the transport protocol can support different abstractions. Reliable and in-order delivery is one option. Unreliable but potentially prompter delivery is another. This corresponds to two different protocols (TCP and UDP), which we'll touch upon briefly at the end of this lecture and explore in more detail in the next.

3. The routing layer is free to use whatever method it deems appropriate to get data from end of the Internet to the other (i.e., it provides global reachability). Further, the routing within an Internet Service Provider (ISP) is completely up to the ISP so long as it can find a way to connect to another ISP for global reachability. We'll discuss both routing within and across ISPs later in this course.

4. The link layer supports many different technologies (Ethernet, WiFi, 4G, Bluetooth, etc.). It simply takes data handed to it by the routing layer and figures out how to complete the last and local portion of data delivery (colloquially called last mile connectivity). This last portion could be from an access point to your laptop or from a base station to a phone, for instance.

5. Finally, for any given link layer, there can be many different implementations of the physical layer that use electromagnetic waves of different frequencies (900 MHz, 2.4 GHz, 3.6 GHz, 4.9 GHz, 5 GHz, 5.9 GHz, and 60 GHz for WiFi) or different cable types (Category 5, Category 6, and Category 7 cables for Ethernet).

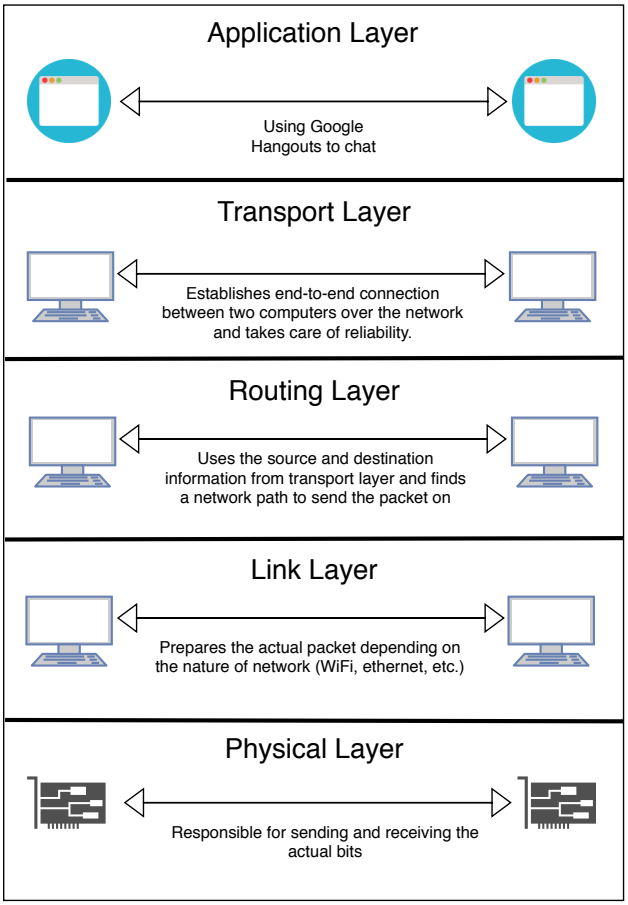

Figure 1: 5 Layer Network Model

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 list of priorities, 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 [3] has a good discussion of these goals.

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 data 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 Internet's overall design may have turned out differently. In fact, an active area of networking research today is about developing 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 design of a datacenter network).

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's 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. 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 raw link capacity was on the order of a few kbit/s.

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.[6]

However, circuit switching was not particularly suited to anything other than phone calls. Unlike phone calls, computer programs that communicate with other communicate 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 sporadically click a link that navigates you to a new web page on the server. Second, for every new web server you want to connect to, you would have to wait to setup a new connection. Third, if any of the links on the electrical connection between your browser and the server failed, the connection would be rendered unusable.

[6] In some cases, a single physical wire was rotated across different conversations over time, but each conversation was still guaranteed a fixed fraction of the wire's capacity, by limiting the number of conversations that used the same wire.

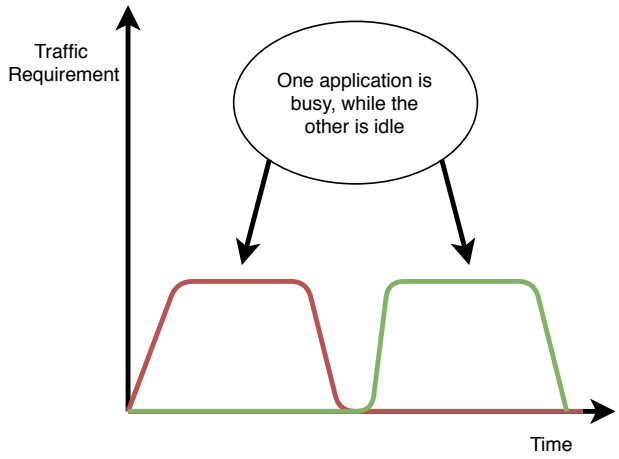

Figure 2: Packet switching benefits from the fact that application traffic bursts are typically uncorrelated.

One way around this is to build links (and telephone exchanges) with such a high degree of reliability that they seldom fail (the Bell telephone company prided itself on such reliability), but this increases the cost of the network.

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[7] over the network between the source and destination. To allow packets to be routed independently, each packet carried a *header* that specified the end host to which 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 and prevented an idle link 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 2). Second, because the packet header identified the destination address, each packet could be transmitted without having to setup 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, in return for these benefits, packet switching has some performance penalties relative to circuit switching. First, it takes slightly longer for a piece of data 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,[8] then the packets need to be buffered at the link, leading to a build up of *queues* at the link (Figure 3). When a queue build ups, a packet's delivery is delayed depending on the number of packets ahead of it in the queue.[9]

[7] We 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.

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

[9] We are assuming the common first-in first-out policy for servicing packets from a queue.

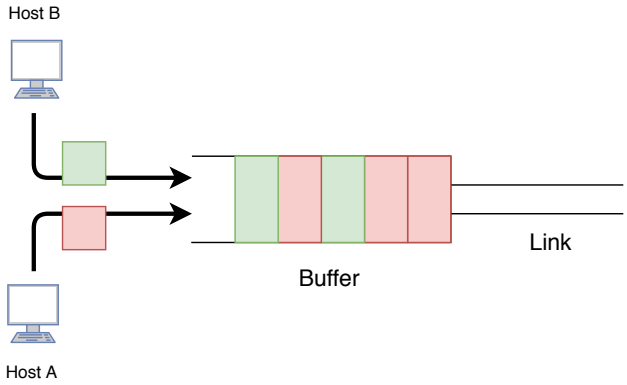

Figure 3: When application bursts coincide, queues build up at network links.

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 succesfully), would have increased the requirement on routers, which in turn would have raised the barrier to entry into the Internet for existing networks.

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. Richard Gabriel's essay "The Rise of Worse is Better" [2] provides a good description of this philosophy.

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 (or a set of rules) 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 4). 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 (e.g., http://www.google.com/index.html) you want and allows the server to respond with the web page at that particular URL.[10]

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.

[10] A URL stands for a Uniform Resource Locator, a mechanism to specify resources such as web pages and files on the Internet.

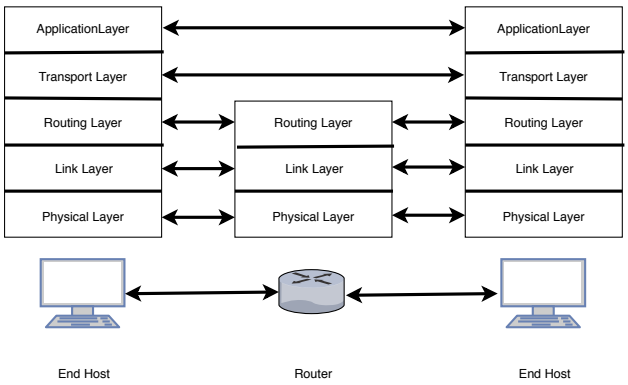

Figure 4: Where each layer is implemented in the Internet.

3.2 The transport layer

Let's move on to the transport layer. As I told you last class, the transport layer primarily provides reliable and in-order delivery. This is implemented by a protocol called TCP (the Transmission 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 and whether message order is preserved.

Why might you want two services? Reliable and 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 chat clients, and different video conferencing applications.

Just as a thought exercise, why not implement them on the router? Let's look at UDP and TCP separately. There is nothing really to implement on a router as far as UDP is concerned. Because UDP provides no guarantees, it has no special requirements, the router doesn't need to handle UDP specially. But for TCP, the router could implement reliability. However, if the router started providing reliability in order to implement TCP, as discussed before, it would be a case of too much effort on the router's part for little reward. The end host has to implement reliability checks anyway, so the router's implementation is not particulary 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 routing 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.[11] 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 synchonize 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 link layer's headers (for MAC source and destination addresses, etc.), the *link-layer throughput*, which is defined as the number of link 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 link-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 link-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. We will also occasionally talk about the two-way latency, also called the round-trip time (RTT), i.e., the time taken to send a single packet from the sender to the receiver plus the time taken for the receiver to send an acknowledgement back to the sender.

When we say latency, we will typically mention the two events between which we are interested in measuring latency. Instead of the term latency, you may also occasionally see the term delay used in the networking literature. Latency or delay consists of several parts, listed below:

1. **Transmission delay.** This is the delay required to transmit a packet of a finite size on a link that can only carry a finite number of bits per second (also called link capacity). For instance, it takes a millisecond

[11] Your 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 (for medium access control) 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.

between the transmission of the first and last bits of a 10000-bit packet on a 10 Mbit/s link. This delay is computed by dividing the packet size by the link's capacity.

2. **Propagation delay.** This is the minimum amount of time required to get information from one point to another. This is computed by dividing the distance between two points by the speed of light within the appropriate medium between the two points (either a wire or the atmosphere). This is also occasionally called the speed-of-light delay.

3. **Application delay.** This is the time between when an end-host's application decides it has data to send and when the end-host's network can actually transmit the data. This is made up of several factors, most notably the time required to transfer between the application running in user space and the end-host's transport, routing, and link layers running in the operating system's kernel.

4. **Queueing delay.** This is the time spent by packets in queues inside the network—a consequence of the unpredictable nature of packet switching as we discussed earlier.

5. **Retransmission delay.** This is the time required to transmit lost data/packets for a protocol that requires reliable delivery of data (e.g., TCP).

This list is not exhaustive by any means. The important point here is that latency or delay can arise due to a number of factors; cutting down each of these sources of latency is an important goal for a good fraction of networking research today.

4.3 Natural limits on throughput and latency

The goal of many networking systems is to increase throughput or decrease latency or some compromise between the two when it is impossible to do both. A reasonable question to ask is whether there are any limits on how low you can make the latency or how high you can make the throughput of a system. In other words, while latency and throughput are measured properties of a system that has been engineered by humans, are there any natural limits on the latency and throughput an engineered system can achieve?

Let's consider throughput and latency in turn. The upper bound on throughput is called capacity. For instance, an Ethernet link might have a capacity of 10 or 100 Gbit/s, implying it can not support a throughput beyond 10 or 100 Gbit/s. However, it is still as yet unclear whether we can build Ethernet links with even higher capacity (e.g., currently there are lab prototypes of 400 Gbit/s links). The capacity of a network is even harder to characterize. Formally characterizing the capacity of single links (whether wired or wireless), groups of links, and entire networks is the province of Information Theory, a topic we won't go into in this class.

The obvious lower bound on latency is the propagation delay between two points plus the transmission delay for a particular packet size. Similarly, the lower bound on the RTT, which we will call RTT_{min} is twice the propagation delay plus the transmission delay of the packet plus the transmission delay of the ACKs. However, we are still quite far from these goals; see `https://people.inf.ethz.ch/asingla/papers/hotnets14.pdf` for a discussion of the vision of a speed-of-light Internet.

To summarize, latency and throughput are measured properties of an engineered system, while propagation delay and capacity represent the best possible scenario for what can be achieved given limits placed by nature.

4.4 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 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 throughput and latency 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).

4.5 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 (and hence throughput) 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.

5 Acknowledgements

Thanks to Salil Kapur for the figures in this lecture.

References

[1] Bulletin! The Internet Is About to Get Smaller. `https://www.nytimes.com/2016/05/25/business/media/internet-to-be-lowercase-in-new-york-times-and-associated-press.html`.

[2] Rise of Worse is Better. `https://www.dreamsongs.com/RiseOfWorseIsBetter.html`.

[3] David D. Clark. The Design Philosophy of the DARPA Internet Protocols. *SIGCOMM Comput. Commun. Rev.*, 1995.