CS-202 Exercises on Wireshark and Network Performance (L12 - L13)

Mini-lab: Wireshark

The goal of this mini-lab is to get you familiar with a network inspection tool called Wireshark. If you are performing the lab on your own Windows/Mac computer, you can download Wireshark from this link. For Ubuntu, you can install it by running the following command in the terminal: *sudo apt-get install wireshark*

Exercise 1: Layers and headers

The Internet architecture operates in layers. As a result, a packet that traverses the Internet looks, in a way, like an onion: On the "outside," it is "wrapped up" in a link-layer header (which can be understood only by the link layer of computers and packet switches). If we "peel away" the link-layer header, we will find a network-layer header (which can be understood only by the network layer of computers and packet switches). If we also peel away the network-layer header, we will find a transport-layer header (which can be understood only by the transport layer of the end-point computers). And if we peel that away, too, we will find the application-layer header and data, which is the actual message that this packet is carrying.

So, if we look inside an Internet packet, we will find a lot more information than the application-layer message that the packet is carrying: we will find meta-data, in the form of headers, which are needed by the various Internet layers in order to get the message from its source to its destination.

We will now use **Wireshark** to look inside Internet packets. To get started, do the following:

- Start your browser and clear its cache. For Firefox, click on the ≡ symbol on the upper-right, go to Settings → Privacy & Security → Cookies & Site Data → Clear data.
- Start the Wireshark tool, e.g., by typing *wireshark* in the command line. You should see a list of your computer's network interfaces (see Fig 1). Identify the one whose packets you will capture. If you are working through an INF3 computer or connected through vdi, you want to capture packets from your ethernet network interface. If you are working on a wirelessly connected computer, capture packets from your WiFi interface.
- Start a capture by double-clicking on the target network interface. You should see data rolling inside the top part of your Wireshark window. These are the packets that are departing from and arriving at your network interface. They most likely make no sense, and that's normal (by the end of the course, they will).
- Use your web browser to visit http://www.mit.edu.
- Stop capturing packets when the web page is fully loaded, by clicking on the square red button at the left of the top menu.

- Right underneath the top menu, you can specify a filter that you want to apply to the packets that you see.

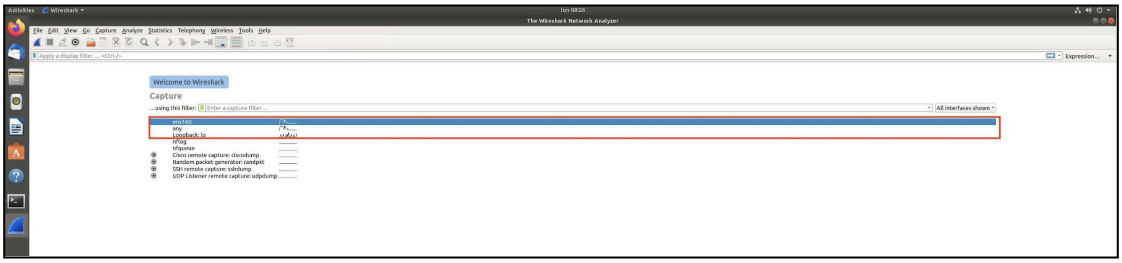


Figure 1: Wireshark startup page

Answer the following questions:

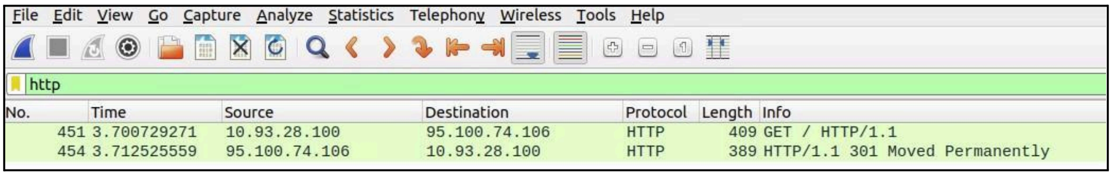
- What messages were exchanged at the **application layer**, i.e., between your web browser and the MIT web server?

  Type http in the filter line, you should see all the packets carrying HTTP messages. HTTP (Hypertext Transfer Protocol) is the communication protocol used between web browsers and web servers. Now check the "Info" column to find the information that these messages are carrying. Is MIT server using HTTP or another protocol to send website content to your browser (Hint: look at the HTTP response & its header fields)?

As we can see in the "Info" column of Fig 1a, two HTTP messages were exchanged:

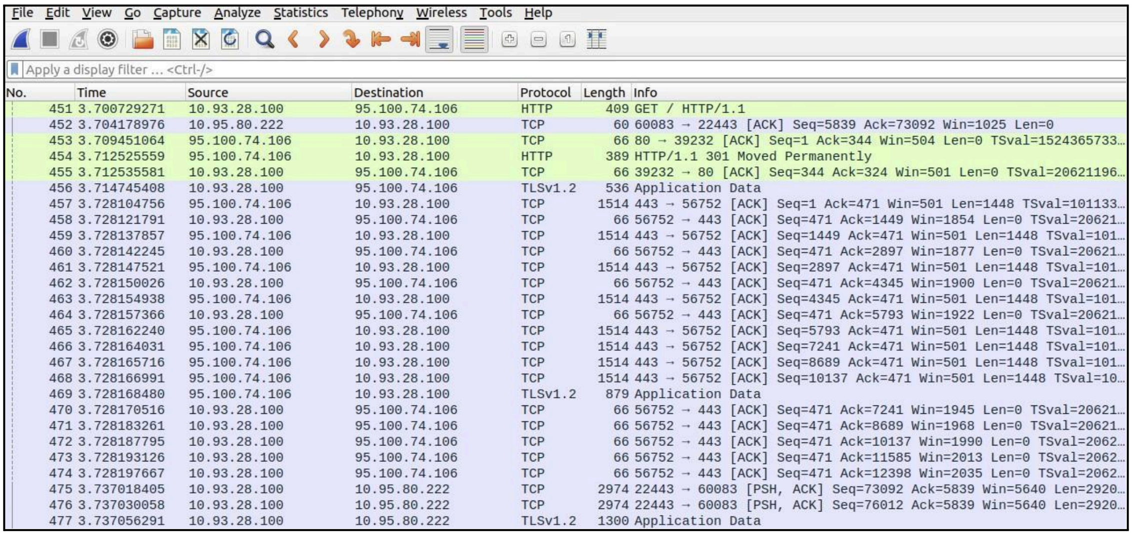
- A GET HTTP/1.1 request, sent by the web browser, to get content from the web server;
- An HTTP/1.1 response, sent by the web server redirecting to the HTTPS website. You can see the redirection address under the "Hypertext Transfer Protocol" header, field: Location, value: https://www.mit.edu/.

The MIT server no longer serves the unsecured HTTP version of the website which we tried to access. If you remove the "http" filter, you can identify the subsequent HTTPS messages by looking for "TLSv1.2" in the "Protocol" column (filter: "tls"). You can see them in Fig 1b.



No.	Time	Source	Destination	Protocol	Length	Info
451	3.700729271	10.93.28.100	95.100.74.106	HTTP	409	GET / HTTP/1.1
454	3.712525559	95.100.74.106	10.93.28.100	HTTP	389	HTTP/1.1 301 Moved Permanently

Figure 1a: Messages containing the HTTP protocol



No.	Time	Source	Destination	Protocol	Length	Info
451	3.700729271	10.93.28.100	95.100.74.106	HTTP	409	GET / HTTP/1.1
452	3.704178976	10.95.80.222	10.93.28.100	TCP	60	60083 → 22443 [ACK] Seq=5839 Ack=73092 Win=1025 Len=0
453	3.709451064	95.100.74.106	10.93.28.100	TCP	66	80 → 39232 [ACK] Seq=1 Ack=344 Win=504 Len=0 TSval=1524365733…
454	3.712525559	95.100.74.106	10.93.28.100	HTTP	389	HTTP/1.1 301 Moved Permanently
455	3.712535581	10.93.28.100	95.100.74.106	TCP	66	39232 → 80 [ACK] Seq=344 Ack=324 Win=501 Len=0 TSval=20621196…
456	3.714745408	10.93.28.100	95.100.74.106	TLSv1.2	536	Application Data
457	3.728104756	95.100.74.106	10.93.28.100	TCP	1514	443 → 56752 [ACK] Seq=1 Ack=471 Win=501 Len=1448 TSval=101133…
458	3.728121791	10.93.28.100	95.100.74.106	TCP	66	56752 → 443 [ACK] Seq=471 Ack=1449 Win=1854 Len=0 TSval=20621…
459	3.728137857	95.100.74.106	10.93.28.100	TCP	1514	443 → 56752 [ACK] Seq=1449 Ack=471 Win=501 Len=1448 TSval=101…
460	3.728142245	10.93.28.100	95.100.74.106	TCP	66	56752 → 443 [ACK] Seq=471 Ack=2897 Win=1877 Len=0 TSval=20621…
461	3.728147521	95.100.74.106	10.93.28.100	TCP	1514	443 → 56752 [ACK] Seq=2897 Ack=471 Win=501 Len=1448 TSval=101…
462	3.728150026	10.93.28.100	95.100.74.106	TCP	66	56752 → 443 [ACK] Seq=471 Ack=4345 Win=1900 Len=0 TSval=20621…
463	3.728154938	95.100.74.106	10.93.28.100	TCP	1514	443 → 56752 [ACK] Seq=4345 Ack=471 Win=501 Len=1448 TSval=101…
464	3.728157366	10.93.28.100	95.100.74.106	TCP	66	56752 → 443 [ACK] Seq=471 Ack=5793 Win=1922 Len=0 TSval=20621…
465	3.728162240	95.100.74.106	10.93.28.100	TCP	1514	443 → 56752 [ACK] Seq=5793 Ack=471 Win=501 Len=1448 TSval=101…
466	3.728164031	95.100.74.106	10.93.28.100	TCP	1514	443 → 56752 [ACK] Seq=7241 Ack=471 Win=501 Len=1448 TSval=101…
467	3.728165716	95.100.74.106	10.93.28.100	TCP	1514	443 → 56752 [ACK] Seq=8689 Ack=471 Win=501 Len=1448 TSval=101…
468	3.728166991	95.100.74.106	10.93.28.100	TCP	1514	443 → 56752 [ACK] Seq=10137 Ack=471 Win=501 Len=1448 TSval=10…
469	3.728168480	95.100.74.106	10.93.28.100	TLSv1.2	879	Application Data
470	3.728170516	10.93.28.100	95.100.74.106	TCP	66	56752 → 443 [ACK] Seq=471 Ack=7241 Win=1945 Len=0 TSval=20621…
471	3.728183261	10.93.28.100	95.100.74.106	TCP	66	56752 → 443 [ACK] Seq=471 Ack=8689 Win=1968 Len=0 TSval=20621…
472	3.728187795	10.93.28.100	95.100.74.106	TCP	66	56752 → 443 [ACK] Seq=471 Ack=10137 Win=1990 Len=0 TSval=2062…
473	3.728193126	10.93.28.100	95.100.74.106	TCP	66	56752 → 443 [ACK] Seq=471 Ack=11585 Win=2013 Len=0 TSval=2062…
474	3.728197667	10.93.28.100	95.100.74.106	TCP	66	56752 → 443 [ACK] Seq=471 Ack=12398 Win=2035 Len=0 TSval=2062…
475	3.737018405	10.93.28.100	10.95.80.222	TCP	2974	22443 → 60083 [PSH, ACK] Seq=73092 Ack=5839 Win=5640 Len=2920…
476	3.737030058	10.93.28.100	10.95.80.222	TCP	2974	22443 → 60083 [PSH, ACK] Seq=76012 Ack=5839 Win=5640 Len=2920…
477	3.737056291	10.93.28.100	10.95.80.222	TLSv1.2	1300	Application Data

Figure 1b: Messages containing the HTTP and HTTPS protocols

- Which technology/communication protocol was used at the **transport layer**? There are two of them, TCP (Transmission Control Protocol) and UDP (User Datagram Protocol), and you need to figure out which one was used. To answer, click on one of the packets in the top section of your Wireshark window, then check the detailed information about this packet that appears in the middle section of your window. You should see information about each layer. Near the bottom, you should see a line that refers to the application layer (it says "Hypertext Transfer Protocol"). What does the line on top of that say?

TCP.

- What messages were exchanged at the **transport layer**, i.e., between the transport layer on your computer and the transport layer on the computer running the MIT web server?

  This is a little bit trickier to answer. First of all, you need to replace http in the filter line with the correct transport-layer technology/communication protocol, which you figured out in the previous question. But if you do just that, then you will see ALL the messages exchanged by your computer using that protocol, whereas you only want the ones exchanged with the computer running the MIT web server. So, you need to add something more to the filter. Poke around a bit in Wireshark documentation on how to specify filters, and you should figure it out.

A key point here is that the application-layer messages and the transport-layer messages were not carried in separate Internet packets. Rather, the same packets carried BOTH transport-layer and application-layer information, but the transport- layer information was stored inside the transport-layer header of each packet, whereas the application-layer information was stored inside the application-layer header and data.

The filter we need to apply is tcp and ip.addr==95.100.74.106, because it displays all the TCP messages sent or received by the computer running the MIT web server (which has IP address 95.100.74.106). You may need to change this IP address to a different one, if you happened to access the MIT web site through a different computer.

As we can see in Fig 1c, the messages exchanged at the transport layer are:

- SYN (to initiate a TCP connection),
- SYN ACK,
- data packets

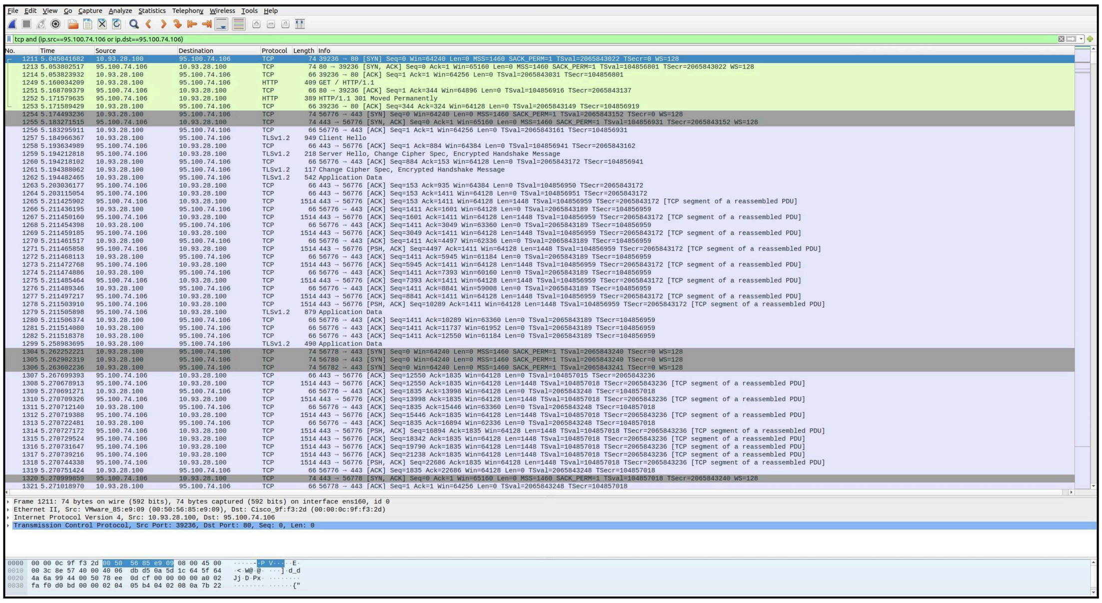


Figure 1c: TCP messages exchanged with www.mit.edu

Exercise 2: Encapsulation

Now we will examine the concept of **encapsulation**, meaning that each message encapsulates a message that belongs to a higher layer. E.g., a network-layer message consists of a network-layer header plus a transport-layer message, which consists of a transport-layer header plus an application-layer message, which consists of an application-layer header plus data.

Display the messages exchanged at the application layer (the HTTP messages) and click on one of them. Can you spot the different layers? Fig 2 shows where each header starts. Notice that each header has different fields from the other headers. Each field is there to serve a specific functionality related to that layer. In this course, we will go through each layer, from bottom to top, and explain its functionalities. Towards the end, you will understand how the different layers interact and what most of these fields mean.

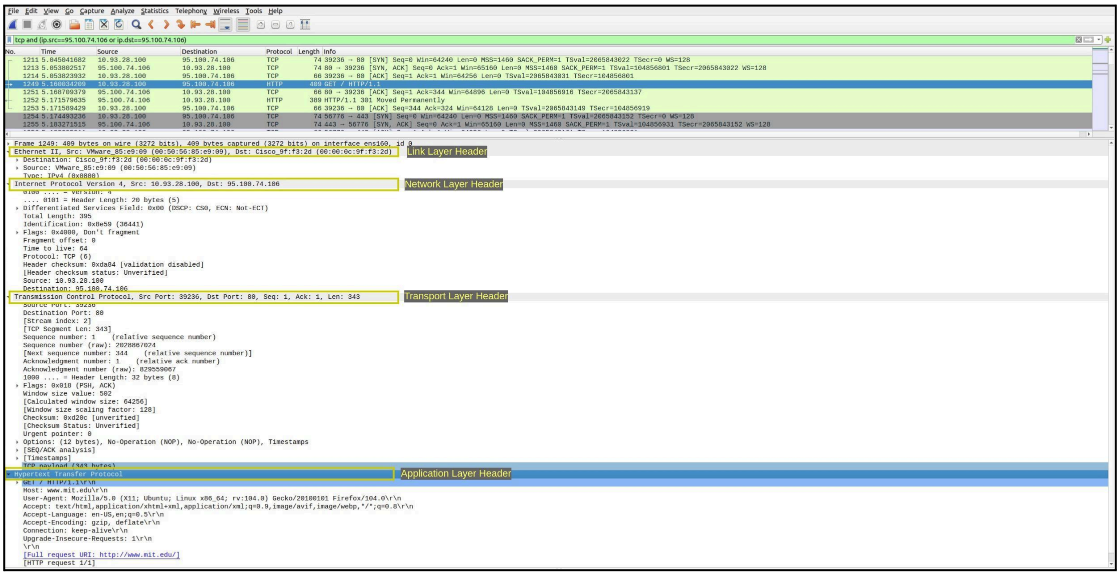


Figure 2: The different layers of a message

But now let us see if you can figure out some:

- How many bytes does the HTTP message contain? To answer, check the packet details in the middle section of your Wireshark window. Look at the transport- layer information and, in particular, the Len field, which specifies the size of the application-layer message that is encapsulated inside the transport-layer message.

- How many bytes do the transport-layer and network-layer headers add to the HTTP message?

- How many bytes does the link layer add?

See Fig 2a.

This HTTP message contains 343 bytes, as shown in the Len field of the Transmission Control Protocol.

The TCP header adds 32 bytes, and the IP header 20 bytes (look for "Header Length:").

The link layer adds 14 bytes (409 - (343 + 20 + 32) = 14).

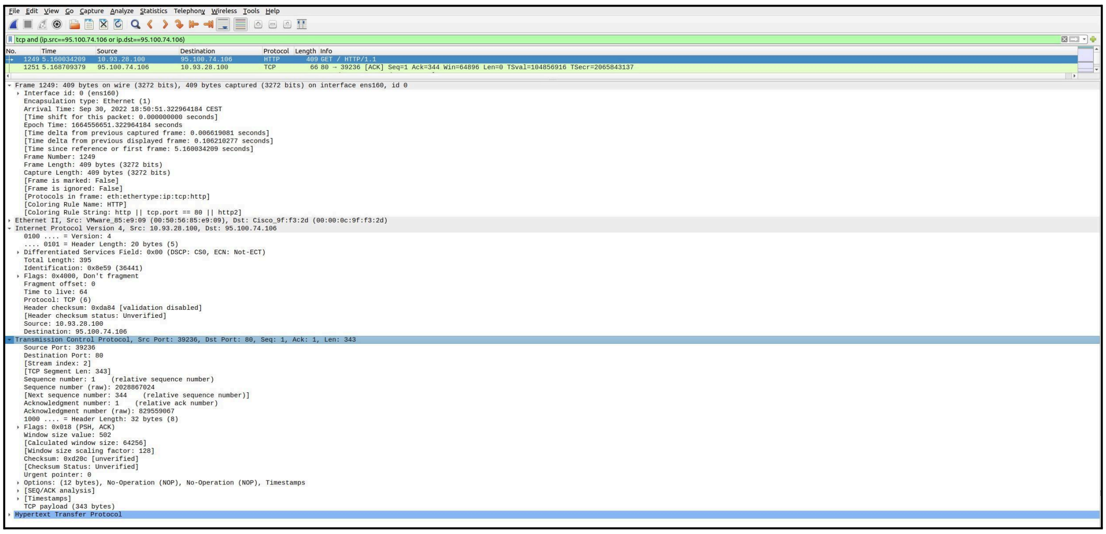


Figure 2a: HTTP message

Paper and pencil exercises: Network Performance

The goal of these exercises is to understand the meaning of throughput and delay, as well as the nature of different delay components.

Transmission rate and throughput

The **transmission rate** of a link is the **rate at which we can push bits into the link**,The **throughput** between two end-systems is **the rate at which destination receives data**, and computed as the total amount of data transferred divided by the total transfer time.

We measure both in "bits per second" (bps), "kilobits per second" (Kbps), "megabits per second" (Mbps), and so on.

The different components of delay

A packet experiences different kinds of delay:

- The **transmission delay** experienced by a **packet on a link** is the amount of time it takes to push the bits of the packet onto the link, and it is equal to the packet length divided by the link transmission rate.
- The **propagation delay of a link** is the amount of time it takes for one bit to go from one end of the link to the other, and it is equal to the link length divided by the link propagation speed.
- The **queuing delay** experienced by a packet at a switch is the **amount of time the packet sits inside a queue at the switch**, waiting for other packets to be processed and transmitted. Queuing delay depends on the other traffic that traverses the same links and shares the same queues as the packet.
- The **processing delay** experienced by a packet at a switch is the amount of **time it takes for the switch to process the packet** (after it removes it from the queue and before starting to transmit it). Processing delay depends on the switch's processing capabilities and is typically independent of packet size, at least in most of the scenarios we will discuss in this class.

Exercise 3: Delay estimation over a single link

Two end-systems, *A* and *B*, are connected by a single link of transmission rate *R*, length *l*, and propagation speed *c*. $R = 100$ Mbps, $l = 200$ km, $c = 2 \cdot 10^8$ m/s.

- What is the propagation delay of the link?

The propagation delay of the link is $d_{prop} = l/c = 2 \cdot 10^5\text{m}/(2 \cdot 10^8\text{m/s}) = 10^{-3}\text{s}$ $= 1\text{ms}$

A sends two back-to-back packets to *B*, the first one of size $L_1 = 1$ kb, the second one of size $L_2 = 10$ kb.

- What is the transmission delay experienced by each packet?

The transmission delay experienced by the first and second packet, respectively, is $d_{trans,1} = L_1/R = 10^3\text{b}/(10^8\text{b/s}) = 10^{-5}\text{s} = 10\text{us}$. $d_{trans,2} = L_2/R = 10^4\text{b}/(10^8\text{b/s}) = 10^{-4}\text{s} = 100\text{us}$.

- What is the total transfer time, i.e. the time that elapses from the moment *A* starts transmitting the first bit of the first packet until the moment *B* receives the last bit of the second packet?

From Figure 3a we can see that the total transfer time is $d_{prop} + d_{trans,1} + d_{trans,2} = 1.11$ms

- What is the packet inter-arrival time at B, i.e., the amount of time that elapses from the moment the last bit of the first packet arrives until the moment the last bit of the second packet arrives?

The packet inter-arrival time at B is equal to the transmission delay experienced by the second packet, i.e. $d_{trans,2} = 100$us.

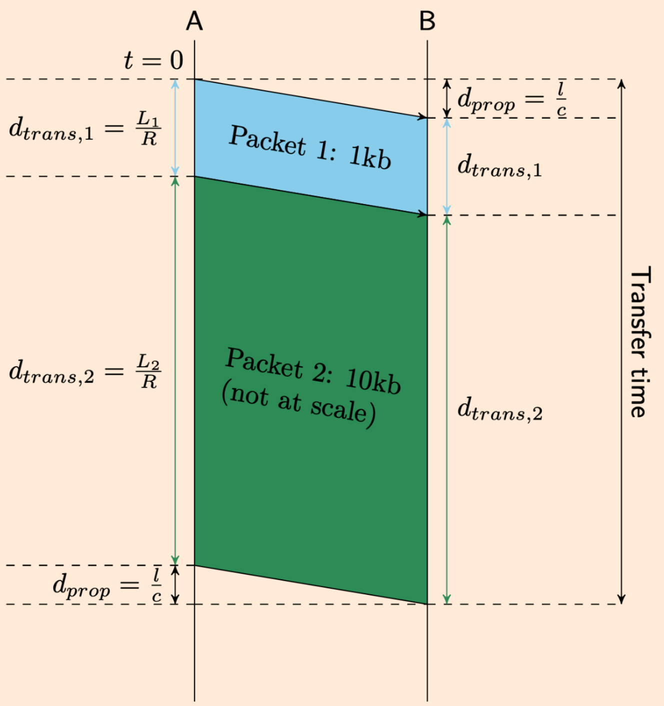


Figure 3a: Timing diagram for transmitting two packets in sequence over a single link.

Exercise 4: Store-and-forward packet switching

Let's examine how transfer time changes as we start introducing network devices between source and destination. Suppose end-systems A and B are connected by not one, but N links and N-1 packet switches. All the links have the same properties as the link in the previous problem. These packet switches typically perform *store-and-forward packet switching*: when a switch receives a bit, it must store that bit in a buffer until the last bit of the corresponding packet has arrived; only then can the switch process the packet and start transmitting it over the next link.

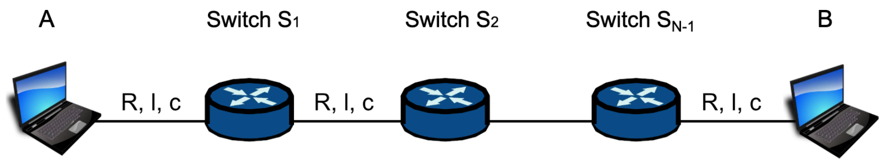


Figure 3: Two end-systems connected through multiple links and packet switches.

Assume that the processing delay at each switch is 0. Still, the fact that every switch must buffer every bit until it has received the corresponding packet means that there will be some extra delay. (Picture a car rally where every time the first car arrives at an intermediate stop, it must wait for all the cars behind it to arrive before it can start again. Now replace the cars with bits of a packet.)

This is a good moment to introduce what we call **timing diagrams**: pictures that represent the various delays that a packet experiences as it travels through a network path. We have started such a diagram below (Figure 4):

- There is a continuous vertical line for each end-system and switch; these lines represent time axes.
- A dashed horizontal line, that crosses the time axes, represents a point in time. For example, on the timing diagram below, the first dashed horizontal line represents $t = 0$, the second one represents $t = d_{prop}$, and so on.
- A continuous line that connects two time axes represents a bit as it travels between the corresponding devices; the beginning of this line represents the moment when the bit was transmitted, while the end of the line represents the moment when the bit arrived at the corresponding device. For example, on the timing diagram, the first continuous line between the *A* and Switch1 time axes represents the first bit of the first packet as it travels from *A* to Switch1; the second continuous line between the same time axes represents the last bit of the first packet.

Using these diagrams, we can easily identify the various delay components that a packet experiences on each link and at each switch. We have marked the first packet sent from *A* to *B*, when $N = 2$ links. Notice what is happening at the switch: because it is a store-and-forward switch, it cannot transmit the first bit of the first packet as soon as it receives it, it must wait for the last bit of the first packet to arrive; only then it can process and transmit the first packet.

You can draw a new diagram, with 4 time axes, to solve the problem for $N = 3$. You cannot draw a diagram for an arbitrary number of links N , but 2 and 3 links are usually enough to give you the right intuition.

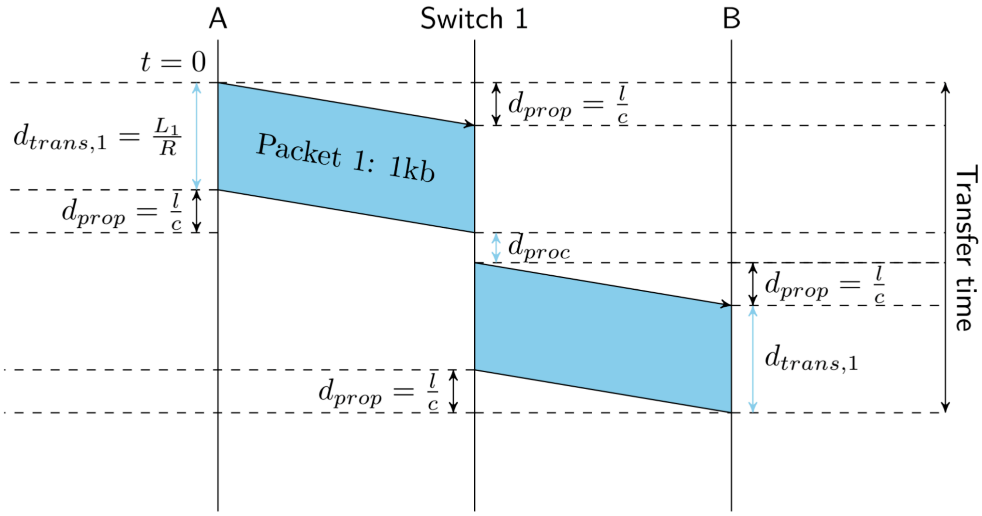


Figure 4: Timing diagram for a transmission over two links connected by a packet switch.

A sends two back-to-back packets to B, each of size $L = 1$ kb.

- What is the total transfer time when N is equal to 2, then 3 links?
- What is the total transfer time as a function of an arbitrary number of links N?

For this and the above question, the answer is practically the same as below, the only difference being that here $d_{proc} = 0$us.

- How do your answers change when the processing delay at each switch is $d_{proc} = 1\mu$s per packet? Assume that the switch must finish processing and transmitting a packet before it starts processing the next packet.

The timing diagram for $N = 2$ links is shown in Figure 4a. To solve for the transfer time, we first identify the bottleneck (i.e., the slowest part of the network). In this example, it is Switch 1. Then, we compute:

- the delay before (i.e., before Switch 1 sends the first packet): $d_{prop} + d_{trans,1}$
- The delay of the bottleneck (i.e., for Switch 1 to send everything): $d_{proc} + d_{trans,1} + d_{proc} + d_{trans,2}$
- And the delay after (i.e., after Switch 1 finishes sending the last packet): d_{prop}

Finally, you sum all the terms together and you get a transfer time equal to $2 \cdot (d_{prop} + d_{trans,1} + d_{proc}) + d_{trans,2} = 2.032$ms.

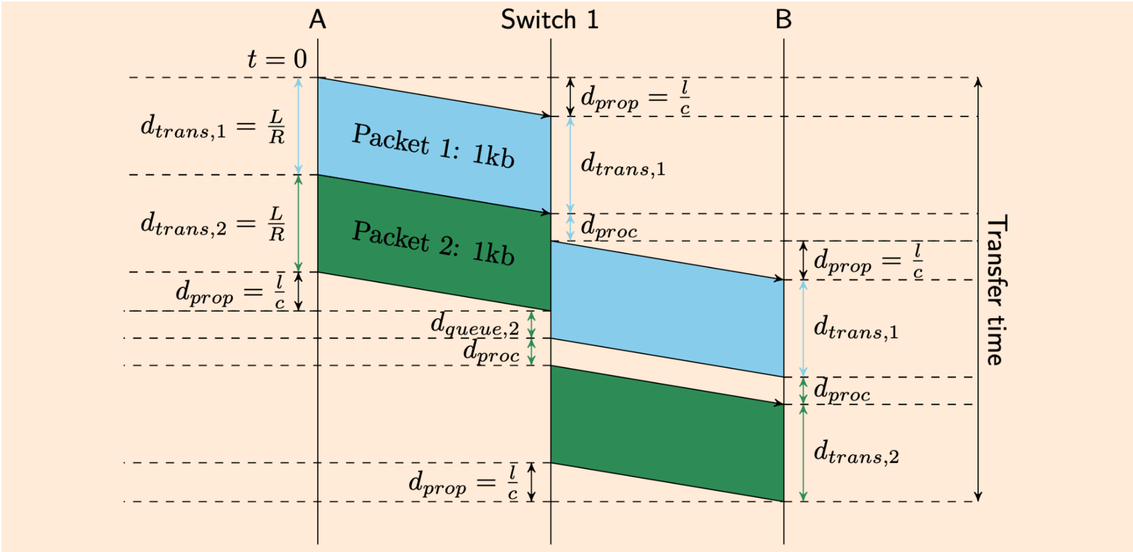


Figure 4a: Timing diagram for a transmission over two links connected by a store-and- forward switch.

The timing diagram for $N = 3$ links is shown in Figure 4b. The transfer time is $3 \cdot (d_{prop} + d_{trans,1} + d_{proc}) + d_{trans,2} = 3.043$ms.

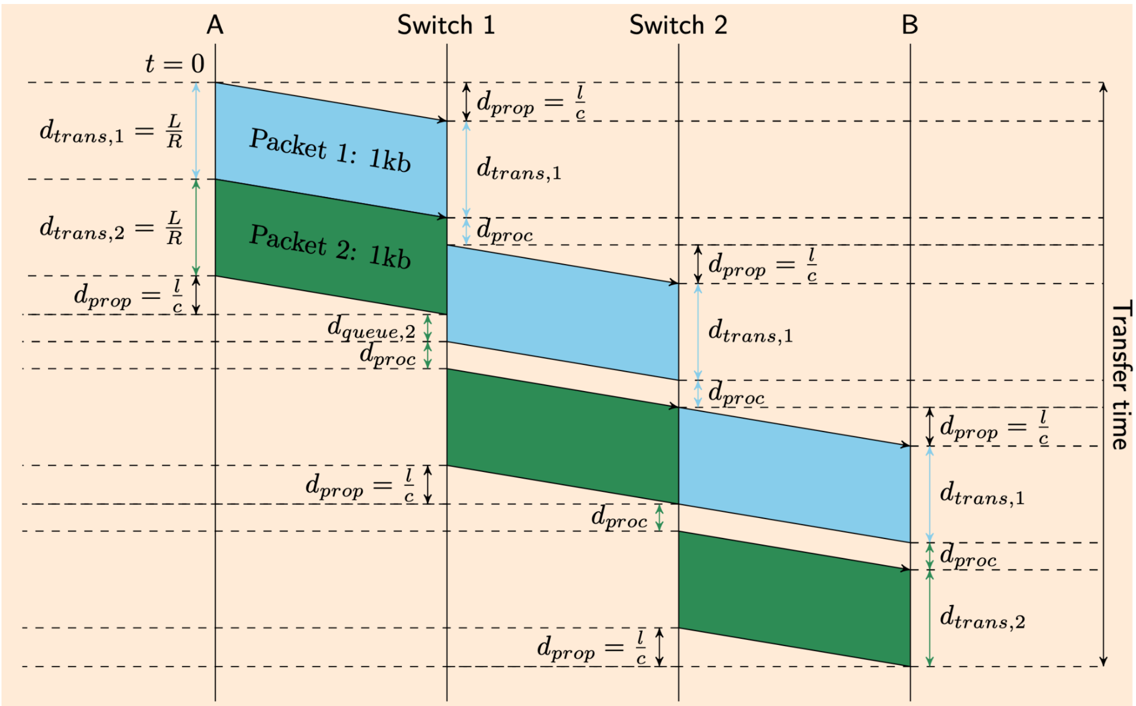


Figure 4b: Timing diagram for a transmission over three links connected by store-and- forward switches.

We can see from the previous two cases that as we add one more link, the transfer time increases by $(d_{prop} + d_{trans,1} + d_{proc})$. Thus for $N \geq 2$ links, the transfer time is $N \cdot (d_{prop} + d_{trans,1} + d_{proc}) + d_{trans,2}$

- Does the second packet experience any queuing delay at the first store-and-forward switch? To answer, focus on the first switch and compute the difference between: (a) when the switch transmits the last bit of the first packet and (b) when the switch receives the last bit of the second packet. If this is greater than 0, it means that the second packet must wait for the switch to finish transmitting the first packet, i.e., it experiences queuing delay.

Yes, it does. As shown in Figure 4a and Figure 4b, at Switch1, the second packet experiences queuing delay $d_{queue,2} = (d_{prop} + d_{trans,1} + d_{proc} + d_{trans,1}) - (d_{prop} + d_{trans,1} + d_{trans,2}) = d_{proc} = 1$us.

- Does the second packet experience any queuing delay at any other switch?

No, because as soon as the last bit of the second packet arrives at any of the subsequent switches, the first packet has already been fully transmitted over the next link.

- Consider that A and B are connected through $N - 1$ store-and-forward switches, each introducing processing delay 0. Except now, A sends P back-to-back packets to B, all with the same length L. Assume zero propagation delays. What is the total transfer time?

- When two end-systems communicate over the Internet, they often exchange $P > 2$ packets. Intuitively, that should increase the total transfer time, but by how much? A factor of P ? Does it depend on N ?

Since the timing diagram becomes a bit more complicated when consider- ing multiple links, we may think in the following way:

At time $N \cdot L/R$ the (last bit of the) first packet reaches end-system B, the second packet is ready to be transmitted over the last link, the third packet is ready to be transmitted over the second-to-last link, etc. At time $N \cdot (L/R) + L/R$, the second packet reaches B, the third packet is ready to be transmitted over the last link, etc. Continuing with this logic, we see that all packets will have reached B by time

$$d_{tranfer} = N \cdot L/R + (P-1) \cdot L/R = (N+P-1) \cdot L/R$$

Exercise 5: Queuing and bottlenecks

You may have noticed the artificial uniformity in the previous problems: all packets were of the same length, all links were of the same transmission rate, etc. Reality is not like this, of course. Packets have different lengths, and links have different transmission rates, and both of these things lead to queuing. Even if *A* and *B* are alone in the Internet—there is no other traffic to interfere with theirs—when *A* sends back-to-back packets to *B*, a packet may have to wait at a store-and-forward switch for the previous packet to be transmitted, either because the previous packet is longer, or because the next link is slower.

End-systems *A* and *B* are connected through two links with one store-and-forward packet switch in the middle, introducing processing delay 0. Both links have propagation delay d_{prop}. The first link has transmission rate R_1, while the second one has transmission rate R_2.

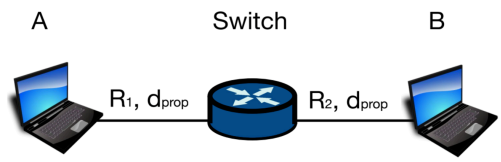


Figure 5: Two end-systems connected through two links.

Assume that $R_1 = R_2$. *A* sends two back-to-back packets to *B*, the first one of size L_1, the second one of size $L_2 \neq L_1$.

- In which scenario does the second packet experience queuing delay at the switch? How much?

In case the first packet is longer than the second packet, i.e., $L_1 > L_2$.

As shown in Figure 5a, the queuing delay experienced by the second packet is given by $d_{queue,2} = d_{trans,1} - d_{trans,2}$.

- What is the total transfer time (in this scenario)?

As shown in Figure 5a, the transfer time is $2 \cdot (d_{prop} + d_{trans,1}) + d_{trans,2}$

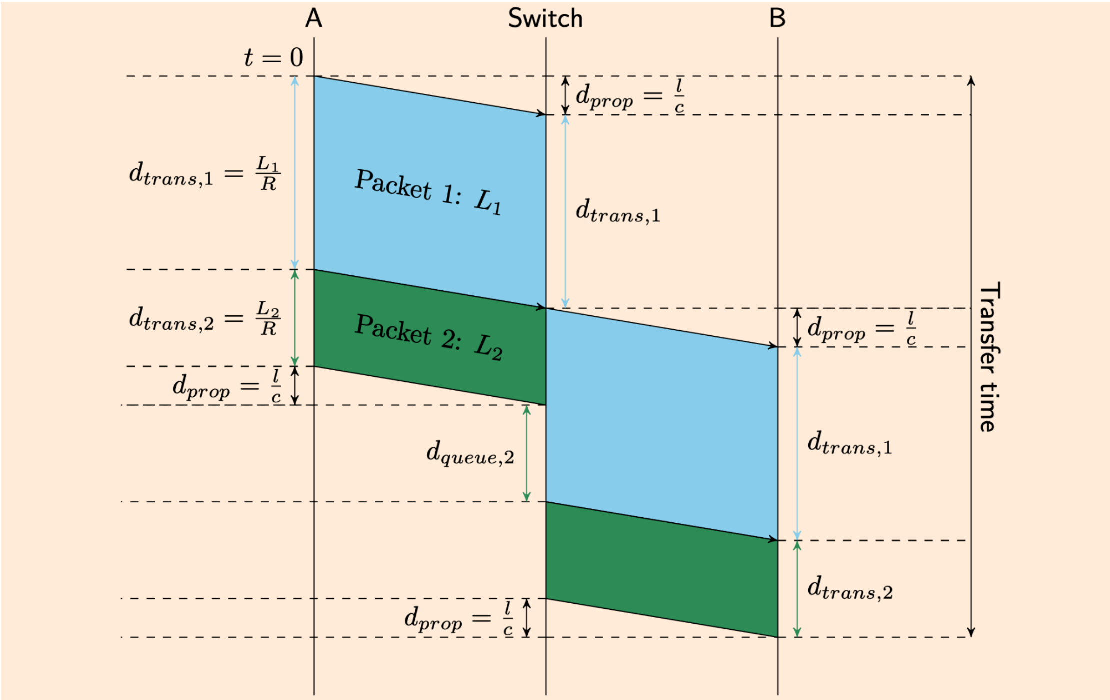


Figure 5a: Timing diagram for a transmission over two links connected by a store-and-forward switch.

Suppose A and B are connected though N links and $N-1$ store-and-forward packet switches, each introducing processing delay 0, all links have propagation delay d_{prop} and transmission rate R, and A sends P back-to-back packets to B, of different lengths, L_1, L_2, ..., L_P . Assume infinite packet-switch buffers (no packet drops).

- What is the total transfer time?

In general, when the packets have different lengths, the order at which the packets are sent may affect the total transfer time.

However, this is not the case in this exercise where all links have the same processing and propagation delays, the same transmission rate, and packet-switch buffers are infinite (no packet drops). Thus, the total transfer time is the same as when sending the *longer* packet, say L_1, first and then the rest of the packets. The total transfer time is given by

$$d_{tranfer} = N \cdot \left(\frac{L_1}{R} + d_{prop} \right) + \sum_{i=2}^{P} \frac{L_i}{R}, \quad \text{where} \quad L_1 \geq L_i, \quad \forall i \in [2, 3, \ldots, P]$$

Now let's go back to the simpler scenario where A and B are connected through two links with one store-and-forward packet switch in the middle, introducing processing delay 0. Assume that $R_2 \neq R_1$. A sends two back-to-back packets to B, each of size L. Is it possible that the second packet experiences queuing delay at the switch? Let's consider all the possibilities:

- Suppose $R_1 < R_2$, i.e., the first link has a lower transmission rate than the second one, in other words, the first link is the bottleneck. (Picture a narrow local street, followed by a wide highway. Would you expect cars to queue up at the highway entrance?)
  - How much queuing delay does the second packet experience at the switch?

As shown in Figure 5b, the second packet experiences 0 queuing delay at the switch.

  - What is the packet inter-arrival time at B?

As shown in Figure 5b, the packet inter-arrival time at B is given by

$$d_{interarr} = d_{tranfer} - d_{tranfer_1} = d_{trans,2} + d_{trans',2} - d_{trans',1}$$

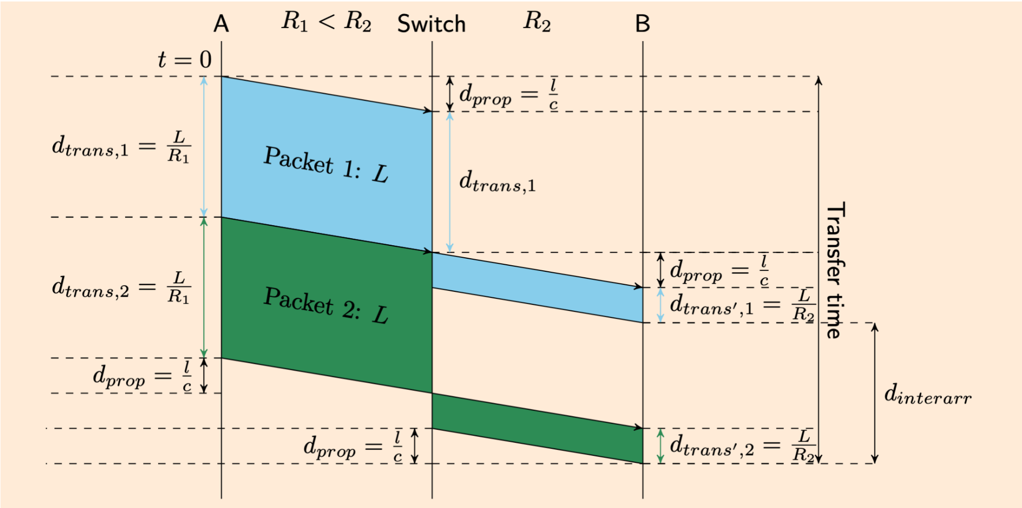


Figure 5b: Timing diagram for a transmission over two links connected by a store-and-forward switch.

- Now suppose $R_1 > R_2$, i.e., the second link is the bottleneck. (Picture a wide highway, followed by a narrow local street. Would you expect cars to queue up at the highway exit?)
  - How much queuing delay does the second packet experience at the switch?

As shown in Figure 5c, the queuing delay of the second packet at the switch is given by $d_{queue,2} = d_{trans',1} - d_{trans,2}$

  - Suppose that A sends the second packet T seconds after sending the first one. How large must T be to ensure no queuing at the switch?

$T \geq d_{queue,2}$ to ensure no queuing delay at the switch.

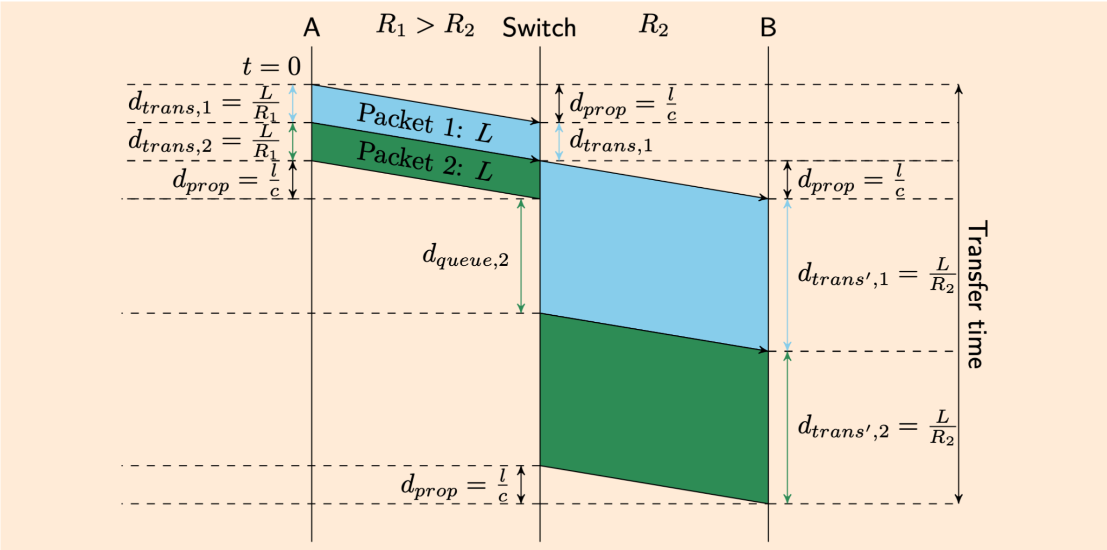


Figure 5c: Timing diagram for a transmission over two links connected by a store-and-forward switch.

Exercise 6: Parallel paths and throughput

Usually, multiple network paths exist between end-systems. Intuitively, adding multiple paths between two end-systems should improve network performance, but which metric exactly? Clearly, adding paths (of the same type) cannot reduce the propagation delay between two end-systems, nor the transmission delay experienced by each single packet. What it *can* change is the overall rate at which A can send data to B: the throughput.

End-systems A and B are connected over M parallel network paths; in this context, "parallel" means that they do not share any links between them. Each path $k \in [1, 2, \ldots, M]$ consists of N links with transmission rates $R_1{}^k, R_2{}^k, \ldots, Rn^k$.

Assume that A wants to send an infinite amount of data to B.

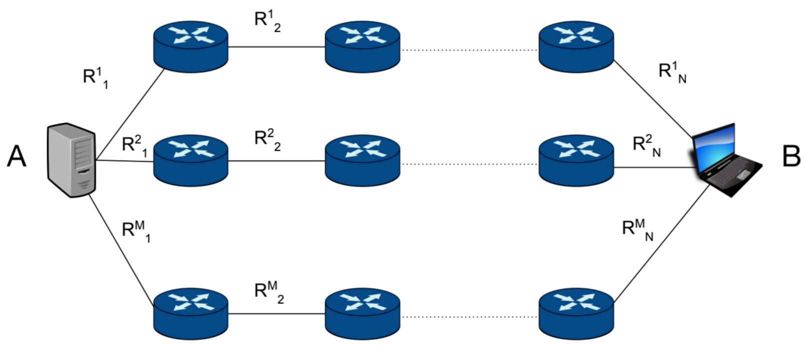


Figure 6: A network topology with multiple parallel paths.

- What is the maximum possible throughput from A to B, when they can use only one path at a time?

When allowed to use only one path at a time, the maximum possible throughput is the maximum throughput across paths, where for each path k, the throughput is equal to the rate of the slowest link on that path. So overall: $\max_{k=1}^{M} \min(R_1^k, R_2^k, \ldots, R_N^k)$.

- How does your answer change when A and B can use all M paths simultaneously?

When allowed to use all M paths simultaneously, the maximum possible throughput is the sum of the throughput over each path. So overall: $\sum_{k=1}^{M} \min(R_1^k, R_2^k, \ldots, R_N^k)$.