MIT 6.02 Lecture Notes
Fall 2011 (Last update: November 27, 2011)
Comments, questions or bug reports?
Please contact hari at mit.edu

CHAPTER 17
Communication Networks: Sharing and Switches

So far in this course we have studied techniques to engineer a *point-to-point* communication link to send messages between two directly connected devices. Two important considerations were at the heart of everything we studied:

1. *Improving link communication reliability:* Inter-symbol interference (ISI) and noise conspired to introduce errors in transmission. We developed techniques to select a suitable sampling rate and method using *eye diagrams* and then reduced the bit-error rate using *channel coding* (linear block codes and convolutional codes, in particular).

2. *Sharing a link:* We developed methods to share a common communication medium amongst multiple nodes. We studied two time sharing techniques, time division multiple access (TDMA) and contention MAC protocols, as well as frequency division multiplexing by modulating different transmissions on carriers of different frequencies.

These techniques give us a communication link between two devices that, in general, has a certain error rate and a corresponding message loss rate. Message losses occur when the error correction mechanism is unable to correct all the errors that occur due to noise or interference from other concurrent transmissions in a contention MAC protocol.

We now turn to the study of *multi-hop communication networks*—systems that connect three or more devices together.[1] The key idea that we will use to engineer communication networks is *composition*: we will build small networks by composing links together, and build larger networks by composing smaller networks together.

The fundamental challenges in the design of a communication network are the same as those that face the designer of a communication link: *sharing* and *reliability*. The big difference is that the sharing problem has different challenges because the system is now *distributed*, spread across a geographic span that is much larger than even the biggest shared medium we can practically build. Moreover, as we will see, many more things can go

[1] By device, we mean things like computer, phones, embedded sensors, and the like—pretty much anything with some computation and communication capability that can be part of a network.

wrong in a network in addition to just bit errors on the point-to-point links, making communication more unreliable than a single link's unreliability.[2]. The next few chapters will discuss these two broad challenges and the key principles that allow us to overcome them.

In addition to sharing and reliability, an important and difficult problem that many communication networks (such as the Internet) face is *scalability*: how to engineer a very large, global system. We won't say very much about scalability in this course, leaving this important topic for later classes.

This chapter focuses on the sharing problem and discusses the following concepts:

1. Switches and how they enable *multiplexing* of different communications on individual links and over the network. Two forms of switching: circuit switching and packet switching.

2. Understanding the role of queues to absorb bursts of traffic in packet-switched networks.

3. Understanding the factors that contribute to delays in networks: three largely fixed delays (propagation, processing, and transmission delays), and one significant variable source of delays (queueing delays).

4. Little's law, relating the average delay to the average rate of arrivals and the average queue size.

■ 17.1 Sharing with Switches

The collection of techniques used to design a communication link, including modulation and error-correcting channel coding, is usually implemented in a module called the *physical layer* (or "PHY" for short). The sending PHY takes a stream of bits and arranges to send it across the link to the receiver; the receiving PHY provides its best estimate of the stream of bits sent from the other end. On the face of it, once we know how to develop a communication link, connecting a collection of N devices together is ostensibly quite straightforward: one could simply connect each pair of devices with a wire and use the physical layer running over the wire to communicate between the two devices. This picture for a small 5-node network is shown in Figure 17-1.

This simple strawman using dedicated pairwise links has two severe problems. First, it is extremely expensive. The reason is that the number of distinct communication links that one needs to build scales quadratically with N—there are $\binom{N}{2} = \frac{N(N-1)}{2}$ bi-directional links in this design (a *bi-directional* link is one that can transmit data in both directions, as opposed to a *uni-directional* link). The cost of operating such a network would be prohibitively expensive, and each additional node added to the network would incur a cost proportional to the size of the network! Second, some of these links would have to span an enormous distance; imagine how the devices in Cambridge, MA, would be connected to those in Cambridge, UK, or (to go further) to those in India or China. Such "long-haul" links are difficult to engineer, so one can't assume that they will be available in abundance.

[2] As one wag put it: "Networking, just one letter away from not working."

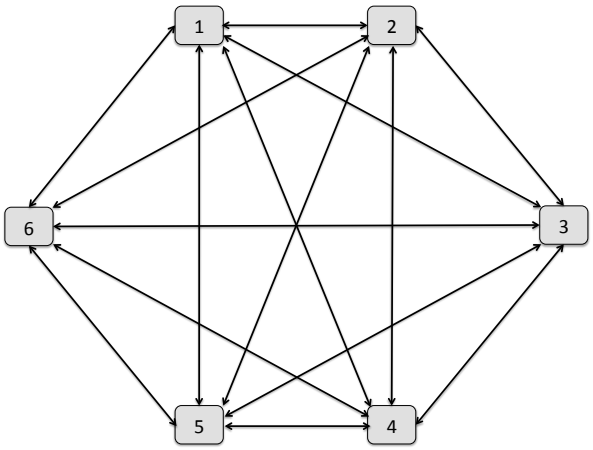


Figure 17-1: A communication network with a link between every pair of devices has a quadratic number of links. Such topologies are generally too expensive, and are especially untenable when the devices are far from each other.

Clearly we need a better design, one that can "do for a dime what any fool can do for a dollar".[3] The key to a practical design of a communication network is a special computing device called a *switch*. A switch has multiple "interfaces" (or ports) on it; a link (wire or radio) can be connected to each interface. The switch allows multiple different communications between different pairs of devices to run over each individual link—that is, it arranges for the network's links to be *shared* by different communications. In addition to the links, the switches themselves have some resources (memory and computation) that will be shared by all the communicating devices.

Figure 17-2 shows the general idea. A switch receives bits that are encapsulated in *data frames* arriving over its links, processes them (in a way that we will make precise later), and forwards them (again, in a way that we will make precise later) over one or more other links. In the most common kind of network, these frames are called *packets*, as explained below.

We will use the term *end points* to refer to the communicating devices, and call the switches and links over which they communicate the *network infrastructure*. The resulting structure is termed the *network topology*, and consists of *nodes* (the switches and end points) and links. A simple network topology is shown in Figure 17-2. We will model the network topology as a *graph*, consisting of a set of nodes and a set of links (edges) connecting various nodes together, to solve various problems.

Figure 17-3 show a few switches of relatively current vintage (ca. 2006).

■ 17.1.1 Three Problems That Switches Solve

The fundamental functions performed by switches are to multiplex and demultiplex data frames belonging to different device-to-device information transfer sessions, and to determine the link(s) along which to forward any given data frame. This task is essential be-

[3] That's what an engineer does, according to an old saying.

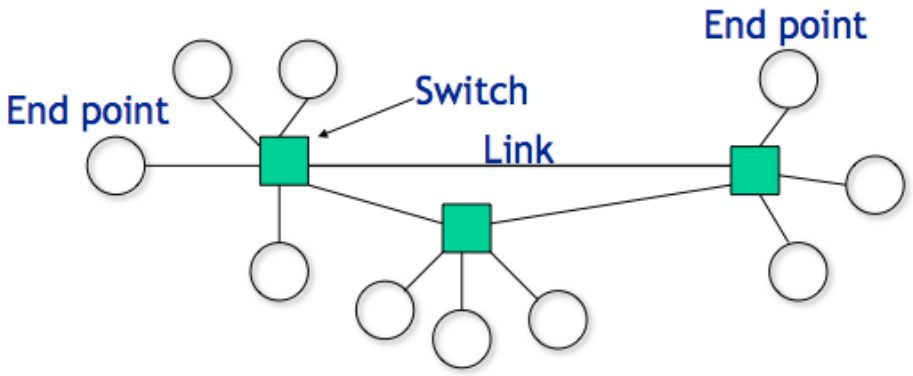


Figure 17-2: A simple network topology showing communicating end points, links, and switches.

cause a given physical link will usually be shared by several concurrent sessions between different devices. We break these functions into three problems:

1. **Forwarding:** When a data frame arrives at a switch, the switch needs to process it, determine the correct outgoing link, and decide when to send the frame on that link.

2. **Routing:** Each switch somehow needs to determine the topology of the network, so that it can correctly construct the data structures required for proper forwarding. The process by which the switches in a network collaboratively compute the network topology, adapting to various kinds of failures, is called routing. It does not happen on each data frame, but occurs in the "background". The next two chapters will discuss forward and routing in more detail.

3. **Resource allocation:** Switches allocate their resources—access to the link and local memory—to the different communications that are in progress.

Over time, two radically different methods have been developed for solving these problems. These techniques differ in the way the switches forward data and allocate resources (there are also some differences in routing, but they are less significant). The first method, used by networks like the telephone network, is called *circuit switching*. The second method, used by networks like the Internet, is called *packet switching*.

There are two crucial differences between the two methods, one philosophical and the other mechanistic. The mechanistic difference is the easier one to understand, so we'll talk about it first. In a circuit-switched network, the frames do not (need to) carry any special information that tells the switches how to forward information, while in packet-switched networks, they do. The philosophical difference is more substantive: a circuit-switched network provides the abstraction of a *dedicated link* of some bit rate to the communicating entities, whereas a packet switched network does not.[4] Of course, this dedicated link traverses multiple physical links and at least one switch, so the end points and switches must do some additional work to provide the illusion of a dedicated link. A packet-switched network, in contrast, provides no such illusion; once again, the end points and switches

[4]One can try to layer such an abstraction atop a packet-switched network, but we're talking about the inherent abstraction provided by the network here.

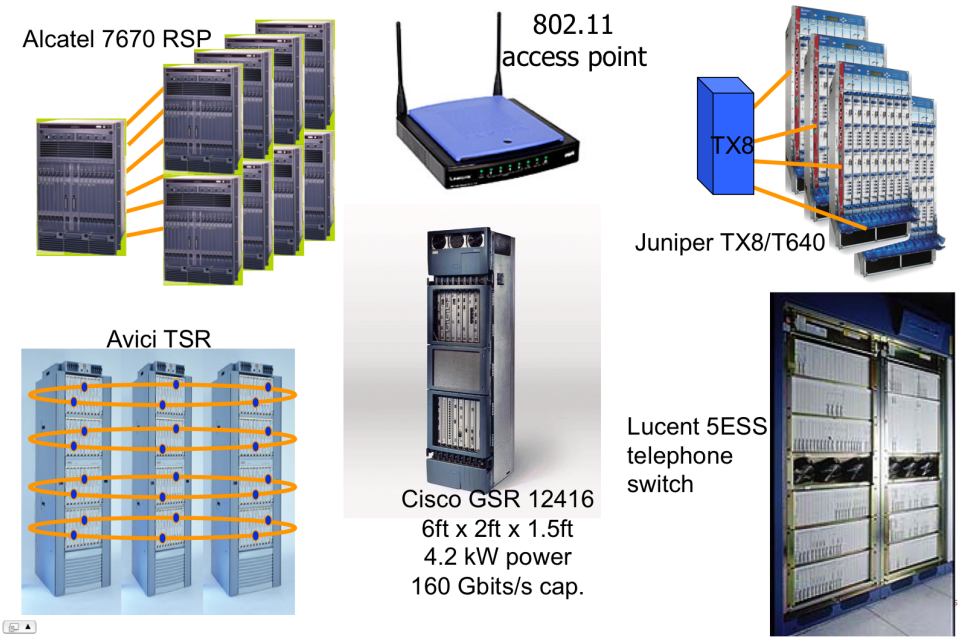


Figure 17-3: A few modern switches.

must do some work to provide reliable and efficient communication service to the applications running on the end points.

17.2 Circuit Switching

The transmission of information in circuit-switched networks usually occurs in three phases (see Figure 17-4):

1. The *setup phase*, in which some state is configured at each switch along a path from source to destination,
2. The *data transfer phase* when the communication of interest occurs, and
3. The *teardown phase* that cleans up the state in the switches after the data transfer ends.

Because the frames themselves contain no information about where they should go, the setup phase needs to take care of this task, and also configure (reserve) any resources needed for the communication so that the illusion of a dedicated link is provided. The teardown phase is needed to release any reserved resources.

17.2.1 Example: Time-Division Multiplexing (TDM)

A common (but not the only) way to implement circuit switching is using *time-division multiplexing (TDM)*, also known as *isochronous transmission*. Here, the physical capacity, or *bit rate*,[5] of a link connected to a switch, C (in bits/s), is conceptually divided into N

[5]This number is sometimes referred to as the "bandwidth" of the link. Technically, bandwidth is a quantity measured in Hertz and refers to the width of the frequency over which the transmission is being done. To

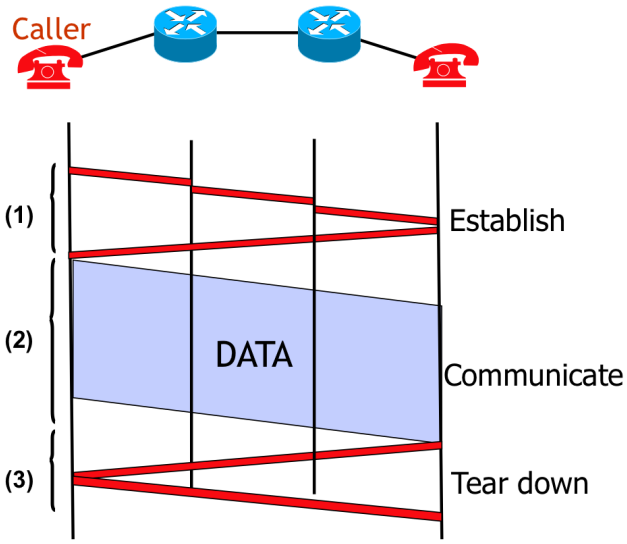


Figure 17-4: Circuit switching requires setup and teardown phases.

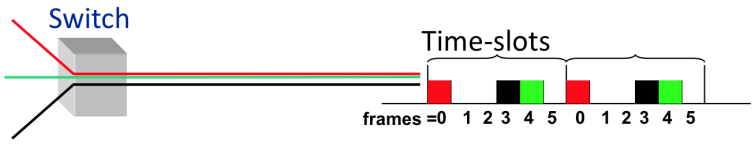


Figure 17-5: Circuit switching with Time Division Multiplexing (TDM). Each color is a different conversation and there are a maximum of $N = 6$ concurrent communications on the link in this picture. Each communication (color) is sent in a fixed time-slot, modulo N.

"virtual links", each virtual link being allocated C/N bits/s and associated with a data transfer session. Call this quantity R, the *rate* of each independent data transfer session. Now, if we constrain each frame to be of some fixed size, s bits, then the switch can perform time multiplexing by allocating the link's capacity in time-slots of length s/C units each, and by associating the ith time-slice to the ith transfer (modulo N), as shown in Figure 17-5. It is easy to see that this approach provides each session with the required rate of R bits/s, because each session gets to send s bits over a time period of Ns/C seconds, and the ratio of the two is equal to $C/N = R$ bits/s.

Each data frame is therefore forwarded by simply using the time slot in which it arrives

avoid confusion, we will use the term "bit rate" to refer to the number of bits per second that a link is currently operating at, but the reader should realize that the literature often uses "bandwidth" to refer to this term. The reader should also be warned that some people (curmudgeons?) become apoplectic when they hear someone using "bandwidth" for the bit rate of a link. A more reasonable position is to realize that when the context is clear, there's not much harm in using "bandwidth". The reader should also realize that in practice most wired links usually operate at a single bit rate (or perhaps pick one from a fixed set when the link is configured), but that wireless links using radio communication can operate at a range of bit rates, adaptively selecting the modulation and coding being used to cope with the time-varying channel conditions caused by interference and movement.

at the switch to decide which port it should be sent on. Thus, the state set up during the first phase has to associate one of these channels with the corresponding soon-to-follow data transfer by allocating the *i*th time-slice to the *i*th transfer. The end points transmitting data send frames only at the specific time-slots that they have been told to do so by the setup phase.

Other ways of doing circuit switching include *wavelength division multiplexing (WDM)*, *frequency division multiplexing (FDM)*, and *code division multiplexing (CDM)*; the latter two (as well as TDM) are used in some wireless networks, while WDM is used in some high-speed wired optical networks.

17.2.2 Pros and Cons

Circuit switching makes sense for a network where the workload is relatively uniform, with all information transfers using the same capacity, and where each transfer uses a *constant bit rate* (or near-constant bit rate). The most compelling example of such a workload is telephony, where each digitized voice call might operate at 64 kbits/s. Switching was first invented for the telephone network, well before devices were on the scene, so this design choice makes a great deal of sense. The classical telephone network as well as the cellular telephone network in most countries still operate in this way, though telephony over the Internet is becoming increasingly popular and some of the network infrastructure of the classical telephone networks is moving toward packet switching.

However, circuit-switching tends to waste link capacity if the workload has a *variable bit rate*, or if the frames arrive in bursts at a switch. Because a large number of computer applications induce burst data patterns, we should consider a different link sharing strategy for computer networks. Another drawback of circuit switching shows up when the $(N+1)^{\text{st}}$ communication arrives at a switch whose relevant link already has the maximum number (N) of communications going over it. This communication must be denied access (or admission) to the system, because there is no capacity left for it. For applications that require a certain minimum bit rate, this approach might make sense, but even in that case a "busy tone" is the result. However, there are many applications that don't have a minimum bit rate requirement (file delivery is a prominent example); for this reason as well, a different sharing strategy is worth considering.

Packet switching doesn't have these drawbacks.

17.3 Packet Switching

An attractive way to overcome the inefficiencies of circuit switching is to permit any sender to transmit data at any time, but yet allow the link to be shared. Packet switching is a way to accomplish this task, and uses a tantalizingly simple idea: add to each frame of data a little bit of information that tells the switch how to forward the frame. This information is usually added inside a *header* immediately before the payload of the frame, and the resulting frame is called a *packet*.[6] In the most common form of packet switching, the header of each packet contains the *address* of the destination, which uniquely identifies the destination of data. The switches use this information to process and forward each packet.

[6]Sometimes, the term *datagram* is used instead of (or in addition to) the term "packet".

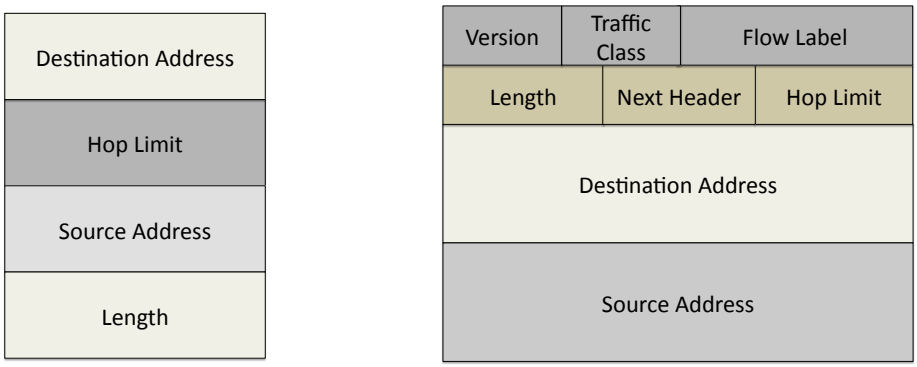


Figure 17-6: LEFT: A *simple and basic* example of a packet header for a packet-switched network. The destination address is used by switches in the forwarding process. The hop limit field will be explained in the chapter on network routing; it is used to discard packets that have been forwarded in the network for more than a certain number of hops, because it's likely that those packets are simply stuck in a loop. Following the header is the payload (or data) associated with the packet, which we haven't shown in this picture. RIGHT: For comparison, the format of the IPv6 ("IP version 6") packet header is shown. Four of the eight fields are similar to our simple header format. The additional fields are the version number, which specifies the version of IP, such as "6" or "4" (the current version that version 6 seeks to replace) and fields that specify, or hint at, how switches must prioritize or provide other traffic management features for the packet.

Packets usually also include the sender's address to help the receiver send messages back to the sender. A simple example of a packet header is shown in Figure 17-6. In addition to the destination and source addresses, this header shows a checksum that can be used for error detection at the receiver.

The figure also shows the packet header used by IPv6 (the Internet Protocol version 6), which is increasingly used on the Internet today. The Internet is the most prominent and successful example of a packet-switched network.

The job of the switch is to use the destination address as a key and perform a lookup on a data structure called a *routing table*. This lookup returns an outgoing link to forward the packet on its way toward the intended destination. There are many ways to implement the lookup opertion on a routing table, but for our purposes we can consider the routing table to be a dictionary mapping each destination to one of the links on the switch.

While forwarding is a relatively simple[7] lookup in a data structure, the trickier question that we will spend time on is determining how the entries in the routing table are obtained. The plan is to use a background process called a *routing protocol*, which is typically implemented in a distributed manner by the switches. There are two common classes of routing protocols, which we will study in later chapters. For now, it is enough to understand that if the routing protocol works as expected, each switch obtains a *route* to every destination. Each switch participates in the routing protocol, dynamically constructing and updating its routing table in response to information received from its neighbors, and providing information to each neighbor to help them construct their own routing tables.

[7] At low speeds. At high speeds, forwarding is a challenging problem.

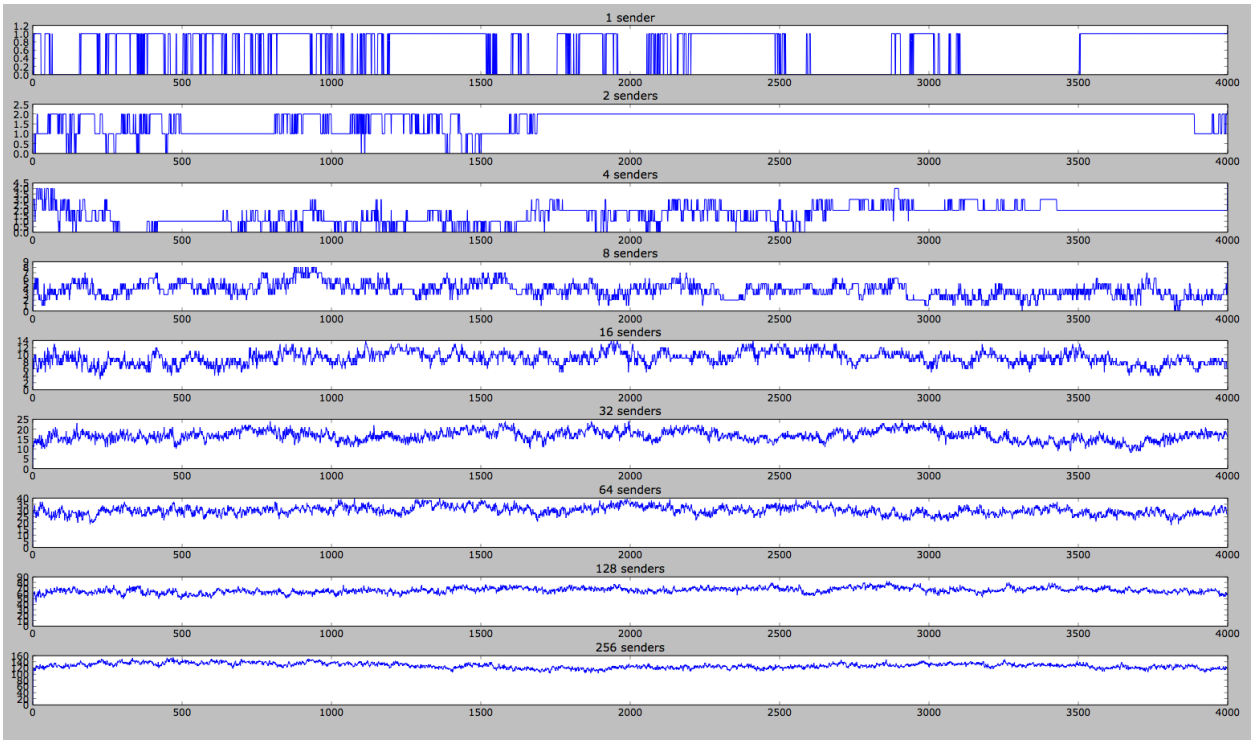


Figure 17-7: Packet switching works because of statistical multiplexing. This picture shows a simulation of N senders, each connected at a fixed bit rate of 1 megabit/s to a switch, sharing a single outgoing link. The y-axis shows the aggregate bit rate (in megabits/s) as a function of time (in milliseconds). In this simulation, each sender is in either the "on" (sending) state or the "off" (idle) state; the durations of each state are drawn from a Pareto distribution (which has a "heavy tail").

Switches in packet-switched networks that implement the functions described in this section are also known as *routers*, and we will use the terms "switch" and "router" interchangeably when talking about packet-switched networks.

■ 17.3.1 Why Packet Switching Works: Statistical Multiplexing

Packet switching does not provide the illusion of a dedicated link to any pair of communicating end points, but it has a few things going for it:

1. It doesn't waste the capacity of any link because each switch can send any packet available to it that needs to use that link.

2. It does not require any setup or teardown phases and so can be used even for small transfers without any overhead.

3. It can provide variable data rates to different communications essentially on an "as needed" basis.

At the same time, because there is no reservation of resources, packets could arrive faster than can be sent over a link, and the switch must be able to handle such situations. Switches deal with transient bursts of traffic that arrive faster than a link's bit rate using

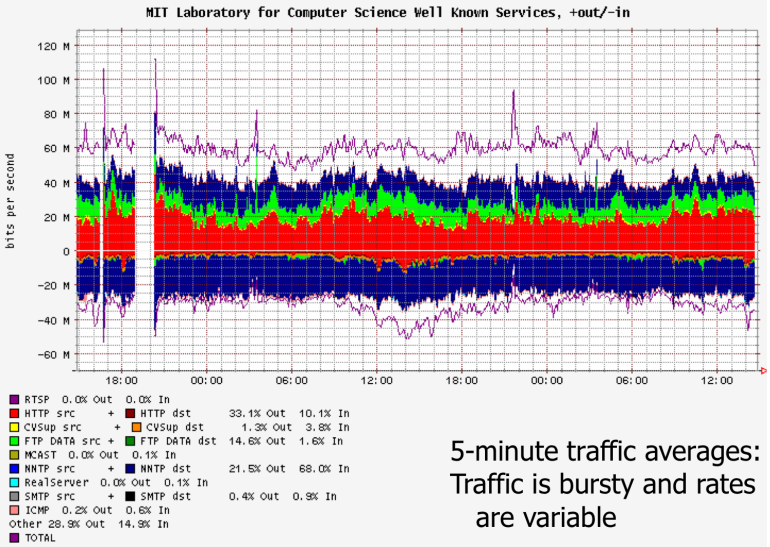


Figure 17-8: Network traffic variability.

queues. We will spend some time understanding what a queue does and how it absorbs bursts, but for now, let's assume that a switch has large queues and understand why packet switching actually works.

Packet switching supports end points sending data at variable rates. If a large number of end points conspired to send data in a synchronized way to exercise a link at the same time, then one would end up having to provision a link to handle the peak synchronized rate for packet switching to provide reasonable service to all the concurrent communications.

Fortunately, at least in a network with benign, or even greedy (but non-malicious) sending nodes, it is highly unlikely that all the senders will be perfectly synchronized. Even when senders send long bursts of traffic, as long as they alternate between "on" and "off" states and move between these states at random (the probability distributions for these could be complicated and involve "heavy tails" and high variances), the aggregate traffic of multiple senders tends to smooth out a bit.[8]

An example is shown in Figure 17-7. The x-axis is time in milliseconds and the y-axis shows the bit rate of the set of senders. Each sender has a link with a fixed bit rate connecting it to the switch. The picture shows how the aggregate bit rate over this short time-scale (4 seconds), though variable, becomes smoother as more senders share the link. This kind of multiplexing relies on the randomness inherent in the concurrent communications, and is called *statistical multiplexing*.

Real-world traffic has bigger bursts than shown in this picture and the data rate usually varies by a large amount depending on time of day. Figure 17-8 shows the bit rates observed at an MIT lab for different network applications. Each point on the y-axis is a 5-minute average, so it doesn't show the variations over smaller time-scales as in the previous figure. However, it shows how much variation there is with time-of-day.

[8]It's worth noting that many large-scale *distributed denial-of-service attacks* try to take out web sites by saturating its link with a huge number of synchronized requests or garbage packets, each of which individually takes up only a tiny fraction of the link.

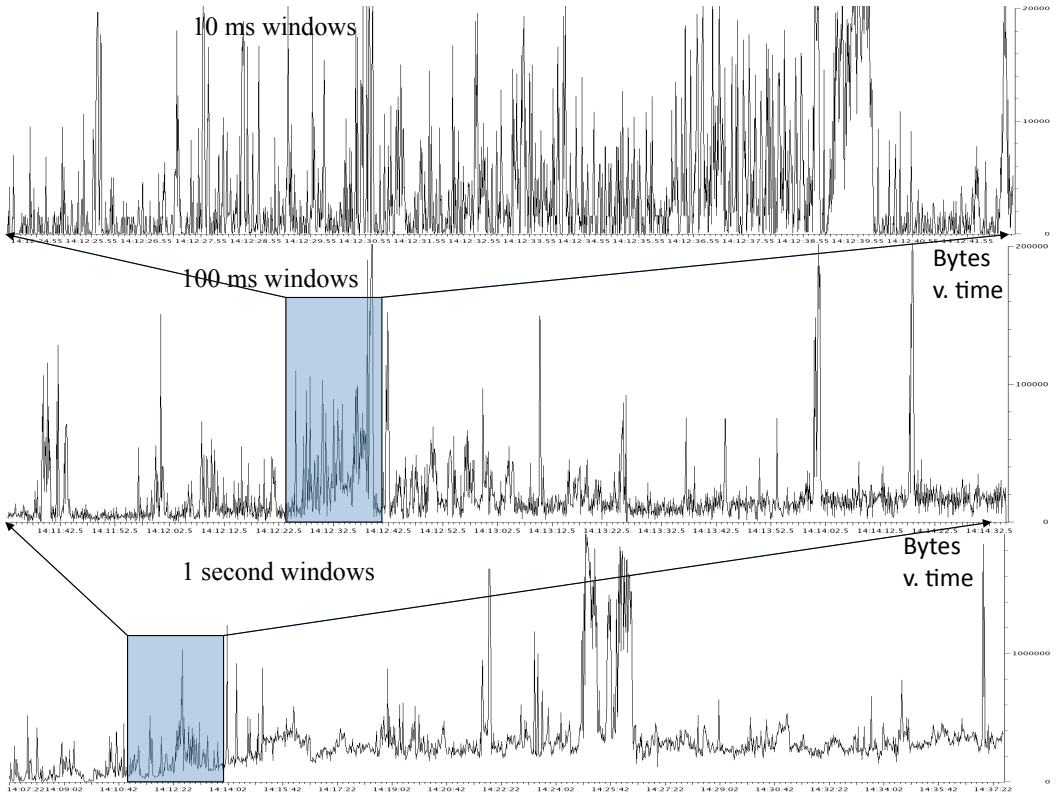


Figure 17-9: Traffic bursts at different time-scales, showing some smoothing. Bursts still persist, though.

So far, we have discussed how the aggregation of multiple sources sending data tends to smooth out traffic a bit, enabling the network designer to avoid provisioning a link for the sum of the peak offered loads of the sources. In addition, for the packet switching idea to really work, one needs to appreciate the time-scales over which bursts of traffic occur in real life.

What better example to use than traffic generated over the duration of a 6.02 lecture on the 802.11 wireless LAN in 34-101 to illustrate the point?! We captured all the traffic that traversed this shared wireless network on a few days during lecture in Fall 2010. On a typical day, we measured about 1 Gigabyte of traffic traversing the wireless network via the access point our monitoring laptop was connected to, with numerous applications in the mix. Most of the observed traffic was from Bittorrent, Web browsing, email, with the occasional IM sessions thrown in the mix. Domain name system (DNS) lookups, which are used by most Internet applications, also generate a sizable number of packets (but not bytes).

Figure 17-9 shows the aggregate amount of data, in bytes, as a function of time, over different time durations. The top picture shows the data over 10 millisecond windows—here, each y-axis point is the total number of bytes observed over the wireless network corresponding to a non-overlapping 10-millisecond time window. We show the data here

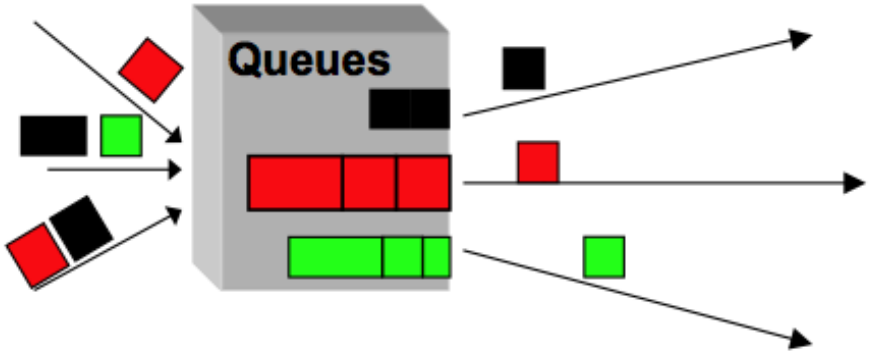


Figure 17-10: Packet switching uses queues to buffer bursts of packets that have arrived at a rate faster than the bit rate of the link.

for a randomly chosen time period that lasts 17 seconds. The most noteworthy aspect of this picture is the bursts that are evident: the maximum (not shown) is as high as 50,000 bytes over this duration, but also note how successive time windows could change between close to 20,000 bytes and 0 bytes. From time to time, larger bursts occur where the network is essentially continuously in use (for example, starting at 14:12:38.55).

The middle picture shows what happens when we look at windows of that are 100 milliseconds long. Clearly, bursts persist, but one can see from the picture that the variance has reduced. When we move to longer windows of 1 second each, we see the same effect persisting, though again it's worth noting that the bursts don't actually disappear.

These data sets exemplify the traffic dynamics that a network designer has to plan for while designing a network. One could pick a data rate that is higher than the peak expected over a short time-scale, but that would be several times larger than picking a smaller value and using a queue to absorb the bursts and send out packets over a link of a smaller rate. In practice, this problem is complicated because network sources are not "open loop", but actually react to how the network responds to previously sent traffic. Understanding how this feedback system works is beyond the scope of 6.02; here, we will look at how queues work.

■ 17.3.2 Absorbing bursts with queues

Queues are a crucial component in any packet-switched network. The queues in a switch absorb bursts of data (see Figure 17-10): when packets arrives for an outgoing link faster than the speed of that link, the queue for that link stores the arriving packets. If a packet arrives and the queue is full, then that packet is simply dropped (if the packet is really important, then the original sender can always infer that the packet was lost because it never got an acknowledgment for it from the receiver, and might decide to re-send it).

One might be tempted to provision large amounts of memory for packet queues because packet losses sound like a bad thing. In fact, queues are like seasoning in a meal—they need to be "just right" in quantity (size). Too small, and too many packets may be lost, but too large, and packets may be excessively delayed, causing it to take *longer* for the senders to know that packets are only getting stuck in a queue and not being delivered.

So how big must queues be? The answer is not that easy: one way to think of it is to ask what we might want the maximum packet delay to be, and use that to size the queue. A more nuanced answer is to analyze the dynamics of how senders react to packet losses and use that to size the queue. Answering this question is beyond the scope of this course, but is an important issue in network design. (The short answer is that we typically want a few tens to ≈ 100 milliseconds of a queue size—that is, we want the queueing delay of a packet to not exceed this quantity, so the buffer size in bytes should be this quantity multiplied by the rate of the link concerned.)

Thus, queues can prevent packet losses, but they cause packets to get delayed. These delays are therefore a "necessary evil". Moreover, queueing delays are *variable*—different packets experience different delays, in general. As a result, analyzing the performance of a network is not a straightforward task. We will discuss performance measures next.

■ 17.4 Network Performance Metrics

Suppose you are asked to evaluate whether a network is working well or not. To do your job, it's clear you need to define some metrics that you can measure. As a user, if you're trying to deliver or download some data, a natural measure to use is the time it takes to finish delivering the data. If the data has a size of S bytes, and it takes T seconds to deliver the data, the *throughput* of the data transfer is $\frac{S}{T}$ bytes/second. The greater the throughput, the happier you will be with the network.

The throughput of a data transfer is clearly upper-bounded by the rate of the slowest link on the path between sender and receiver (assuming the network uses only one path to deliver data). When we discuss reliable data delivery, we will develop protocols that attempt to optimize the throughput of a large data transfer. Our ability to optimize throughput depends more fundamentally on two factors: the first factor is the *per-packet delay*, sometimes called the per-packet *latency* and the second factor is the *packet loss rate*.

The packet loss rate is easier to understand: it is simply equal to the number of packets dropped by the network along the path from sender to receiver divided by the total number of packets transmitted by the sender. So, if the sender sent S_t packets and the receiver got S_r packets, then the packet loss rate is equal to $1 - \frac{S_r}{S_t} = \frac{S_t - S_r}{S_t}$. One can equivalently think of this quantity in terms of the sending and receiving rates too: for simplicity, suppose there is one queue that drops packets between a sender and receiver. If the arrival rate of packets into the queue from the sender is A packets per second and the departure rate from the queue is D packets per second, then the packet loss rate is equal to $1 - \frac{D}{A}$.

The delay experienced by packets is actually the sum of four distinct sources: *propagation*, *transmission*, *processing*, and *queueing*, as explained below:

1. **Propagation delay.** This source of delay is due to the fundamental limit on the time it takes to send any signal over the medium. For a wire, it's the speed of light over that material (for typical fiber links, it's about two-thirds the speed of light in vacuum). For radio communication, it's the speed of light in vacuum (air), about 3×10^8 meters/second.

   The best way to think about the propagation delay for a link is that it is equal to the *time for the first bit of any transmission to reach the intended destination*. For a path

comprising multiple links, just add up the individual propagation delays to get the propagation delay of the path.

2. **Processing delay.** Whenever a packet (or data frame) enters a switch, it needs to be processed before it is sent over the outgoing link. In a packet-switched network, this processing involves, at the very least, looking up the header of the packet in a table to determine the outgoing link. It may also involve modifications to the header of the packet. The total time taken for all such operations is called the processing delay of the switch.

3. **Transmission delay.** The transmission delay of a link is the time it takes for a packet of size S bits to traverse the link. If the bit rate of the link is R bits/second, then the transmission delay is S/R seconds.

   We should note that the processing delay adds to the other sources of delay in a network with *store-and-forward* switches, the most common kind of network switch today. In such a switch, each data frame (packet) is stored before any processing (such as a lookup) is done and the packet then sent. In contrast, some extremely low latency switch designs are *cut-through*: as a packet arrives, the destination field in the header is used for a table lookup, and the packet is sent on the outgoing link without any storage step. In this design, the switch *pipelines* the transmission of a packet on one link with the reception on another, and the processing at one switch is pipelined with the reception on a link, so the end-to-end per-packet delay is smaller than the sum of the individual sources of delay.

   Unless mentioned explicitly, we will deal only with store-and-forward switches in this course.

4. **Queueing delay.** Queues are a fundamental data structure used in packet-switched networks to absorb bursts of data arriving for an outgoing link at speeds that are (transiently) faster than the link's bit rate. The time spent by a packet *waiting* in the queue is its queueing delay.

   Unlike the other components mentioned above, the queueing delay is usually variable. In many networks, it might also be the dominant source of delay, accounting for about 50% (or more) of the delay experienced by packets when the network is congested. In some networks, such as those with satellite links, the propagation delay could be the dominant source of delay.

■ 17.4.1 Little's Law

A common method used by engineers to analyze network performance, particularly delay and throughput (the rate at which packets are delivered), is *queueing theory*. In this course, we will use an important, widely applicable result from queueing theory, called *Little's law* (or Little's theorem).[9] It's used widely in the performance evaluation of systems ranging from communication networks to factory floors to manufacturing systems.

[9]This "queueing formula" was first proved in a general setting by John D.C. Little, who is now an Institute Professor at MIT (he also received his PhD from MIT in 1955). In addition to the result that bears his name, he is a pioneer in marketing science.

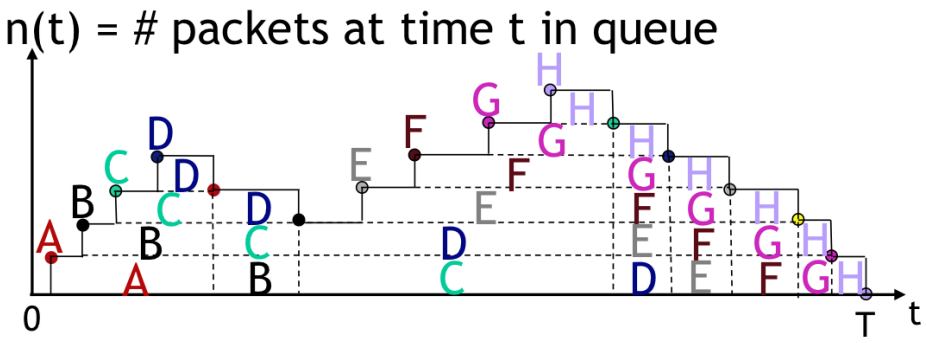


Figure 17-11: Packet arrivals into a queue, illustrating Little's law.

For any stable (i.e., where the queues aren't growing without bound) queueing system, Little's law relates the average arrival rate of items (e.g., packets), λ, the average delay experienced by an item in the queue, D, and the average number of items in the queue, N. The formula is simple and intuitive:

$$N = \lambda \times D \tag{17.1}$$

Example. Suppose packets arrive at an average rate of 1000 packets per second into a switch, and the rate of the outgoing link is larger than this number. (If the outgoing rate is smaller, then the queue will grow unbounded.) It doesn't matter how inter-packet arrivals are distributed; packets could arrive in weird bursts according to complicated distributions. Now, suppose there are 50 packets in the queue on average. That is, if we sample the queue size at random points in time and take the average, the number is 50 packets.

Then, from Little's law, we can conclude that **the average queueing delay experienced by a packet is 50/1000 seconds = 50 milliseconds**.

Little's law is quite remarkable because it is independent of how items (packets) arrive or are serviced by the queue. Packets could arrive according to any distribution. They can be serviced in any order, not just first-in-first-out (FIFO). They can be of any size. In fact, about the only practical requirement is that the queueing system be stable. It's a useful result that can be used profitably in back-of-the-envelope calculations to assess the performance of real systems.

Why does this result hold? Proving the result in its full generality is beyond the scope of this course, but we can show it quite easily with a few simplifying assumptions using an essentially pictorial argument. The argument is instructive and sheds some light into the dynamics of packets in a queue.

Figure 17-11 shows $n(t)$, the number of packets in a queue, as a function of time t. Each time a packet enters the queue, $n(t)$ increases by 1. Each time the packet leaves, $n(t)$ decreases by 1. The result is the step-wise curve like the one shown in the picture.

For simplicity, we will assume that the queue size is 0 at time 0 and that there is some time $T >> 0$ at which the queue empties to 0. We will also assumes that the queue services jobs in FIFO order (note that the formula holds whether these assumptions are true or not).

Let P be the total number of packets forwarded by the switch in time T (obviously, in

our special case when the queue fully empties, this number is the same as the number that entered the system).

Now, we need to define N, λ, and D. One can think of N as the *time average* of the number of packets in the queue; i.e.,

$$N = \sum_{t=0}^{T} n(t)/T.$$

The rate λ is simply equal to P/T, for the system processed P packets in time T.

D, the average delay, can be calculated with a little trick. Imagine taking the total area under the $n(t)$ curve and assigning it to packets as shown in Figure 17-11. That is, packets A, B, C, ... each are assigned the different rectangles shown. The height of each rectangle is 1 (i.e., one packet) and the length is the time until some packet leaves the system. Each packet's rectangle(s) last until the packet itself leaves the system.

Now, it should be clear that the time spent by any given packet is just the sum of the areas of the rectangles labeled by that packet.

Therefore, the average delay experienced by a packet, D, is simply the area under the $n(t)$ curve divided by the number of packets. That's because the total area under the curve, which is $\sum n(t)$, is the *total delay* experienced by all the packets.

Hence,

$$D = \sum_{t=0}^{T} n(t)/P.$$

From the above expressions, Little's law follows: $N = \lambda \times D$.

Little's law is useful in the analysis of networked systems because, depending on the context, one usually knows some two of the three quantities in Eq. (17.1), and is interested in the third. It is a statement about averages, and is remarkable in how little it assumes about the way in which packets arrive and are processed.

■ Acknowledgments

Many thanks to Sari Canelake, Lavanya Sharan, and Kerry Xing for their careful reading and helpful comments.

■ Problems and Questions

1. Under what conditions would circuit switching be a better network design than packet switching?

2. Which of these statements are correct?

   (a) Switches in a circuit-switched network process connection establishment and tear-down messages, whereas switches in a packet-switched network do not.

   (b) Under some circumstances, a circuit-switched network may prevent some senders from starting new conversations.

(c) Once a connection is correctly established, a switch in a circuit-switched network can forward data correctly without requiring data frames to include a destination address.

(d) Unlike in packet switching, switches in circuit-switched networks *do not* need any information about the network topology to function correctly.

3. Consider a switch that uses time division multiplexing (rather than statistical multiplexing) to share a link between four concurrent connections (A, B, C, and D) whose packets arrive in bursts. The link's data rate is 1 packet per time slot. Assume that the switch runs for a very long time.

   (a) The average packet arrival rates of the four connections (A through D), in packets per time slot, are 0.2, 0.2, 0.1, and 0.1 respectively. The average delays observed at the switch (in time slots) are 10, 10, 5, and 5. What are the average queue lengths of the four queues (A through D) at the switch?

   (b) Connection A's packet arrival rate now changes to 0.4 packets per time slot. All the other connections have the same arrival rates and the switch runs unchanged. What are the average queue lengths of the four queues (A through D) now?

4. Annette Werker has developed a new switch. In this switch, 10% of the packets are processed on the "slow path", which incurs an average delay of 1 millisecond. All the other packets are processed on the "fast path", incurring an average delay of 0.1 milliseconds. Annette observes the switch over a period of time and finds that the average number of packets in it is 19. What is the average rate, in packets per second, at which the switch processes packets?

5. Alyssa P. Hacker has set up eight-node shared medium network running the Carrier Sense Multiple Access (CSMA) MAC protocol. The maximum data rate of the network is 10 Megabits/s. Including retries, each node sends traffic according to some unknown random process at an average rate of 1 Megabit/s per node. Alyssa measures the network's utilization and finds that it is 0.75. No packets get dropped in the network except due to collisions, and each node's average queue size is 5 packets. Each packet is 10000 bits long.

   (a) What fraction of packets sent by the nodes (including retries) experience a collision?

   (b) What is the average queueing delay, in milliseconds, experienced by a packet before it is sent over the medium?

6. Over many months, you and your friends have painstakingly collected 1,000 Gigabytes (aka 1 Terabyte) worth of movies on computers in your dorm (we won't ask where the movies came from). To avoid losing it, you'd like to back the data up on to a computer belonging to one of your friends in New York.

   You have two options:

A. Send the data over the Internet to the computer in New York. The data rate for transmitting information across the Internet from your dorm to New York is 1 Megabyte per second.

B. Copy the data over to a set of disks, which you can do at 100 Megabytes per second (thank you, firewire!). Then rely on the US Postal Service to send the disks by mail, which takes 7 days.

Which of these two options (A or B) is faster? And by how much?

Note on units:

1 kilobyte = 10^3 bytes
1 megabyte = 1000 kilobytes = 10^6 bytes
1 gigabyte = 1000 megabytes = 10^9 bytes
1 terabyte = 1000 gigbytes = 10^{12} bytes

7. Little's law can be applied to a variety of problems in other fields. Here are some simple examples for you to work out.

   (a) F freshmen enter MIT every year on average. Some leave after their SB degrees (four years), the rest leave after their MEng (five years). No one drops out (yes, really). The total number of SB and MEng students at MIT is N.

   What fraction of students do an MEng?

   (b) A hardware vendor manufactures $300 million worth of equipment per year. On average, the company has $45 million in accounts receivable. How much time elapses between invoicing and payment?

   (c) While reading a newspaper, you come across a sentence claiming that *"less than 1% of the people in the world die every year"*. Using Little's law (and some common sense!), explain whether you would agree or disagree with this claim. Assume that the number of people in the world does not decrease during the year (this assumption holds).

   (d) (This problem is actually almost related to networks.) Your friendly 6.02 professor receives 200 non-spam emails every day on average. He estimates that of these, 50 need a reply. Over a period of time, he finds that the average number of unanswered emails in his inbox that still need a reply is 100.

      i. On average, how much time does it take for the professor to send a reply to an email that needs a response?

      ii. On average, 6.02 constitutes 25% of his emails that require a reply. He responds to each 6.02 email in 60 minutes, on average. How much time on average does it take him to send a reply to any *non-6.02* email?

8. You send a stream of packets of size 1000 bytes each across a network path from Cambridge to Berkeley. You find that the one-way delay varies between 50 ms (in the absence of any queueing) and 125 ms (full queue), with an average of 75 ms. The transmission rate at the sender is 1 Mbit/s; the receiver gets packets at the same rate without any packet loss.

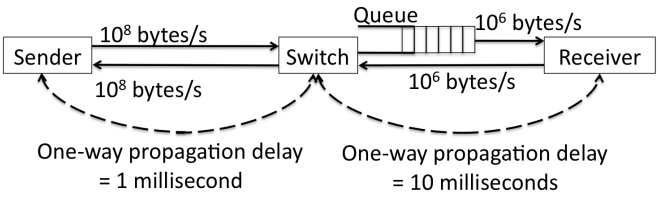
(a) What is the mean number of packets in the queue at the bottleneck link along the path (assume that any queueing happens at just one switch).

You now increase the transmission rate to 2 Mbits/s. You find that the receiver gets packets at a rate of 1.6 Mbits/s. The average queue length does not change appreciably from before.

(b) What is the packet loss rate at the switch?

(c) What is the average one-way delay now?

9. Consider the network topology shown below. Assume that the processing delay at all the nodes is negligible.



(a) The sender sends two 1000-byte data packets back-to-back with a negligible inter-packet delay. The queue has no other packets. What is the time delay between the arrival of the first bit of the second packet and the first bit of the first packet at the receiver?

(b) The receiver acknowledges each 1000-byte data packet to the sender, and each acknowledgment has a size $A = 100$ bytes. What is the minimum possible round trip time between the sender and receiver? The round trip time is defined as the duration between the transmission of a packet and the receipt of an acknowledgment for it.

10. The wireless network provider at a hotel wants to make sure that anyone trying to access the network is properly authorized and their credit card charged before being allowed. This billing system has the following property: if the average number of requests currently being processed is N, then the average delay for the request is $a + bN^2$ seconds, where a and b are constants. What is the maximum rate (in requests per second) at which the billing server can serve requests?