MIT 6.02 Lecture Notes
Spring 2009 (Last update: April 8, 2009)
Comments, questions or bug reports?
Please contact Hari Balakrishnan (hari@mit.edu) or 6.02-staff@mit.edu

LECTURE 17
Communication Networks

So far in this course we have studied techniques to engineer a *point-to-point* communication link. Two important considerations were at the heart of most 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* (block, Reed-Solomon, and convolutional codes).

2. *Sharing a link:* We wanted to share the same communication medium amongst k different receivers, each tuned to a different frequency. To achieve this goal, we used *digital modulation*, and learned that understanding the frequency response of an LTI system and designing filters are key building blocks for this task.

We now turn to the study of communication networks—systems that connect three or more computers (or phones)[1] together.

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 is considerably more challenging, and many more things can go wrong in networking[2] many computers together, making communication more unreliable than a single link's unreliability. The next few lectures will show you these challenges and you will understand the key ideas in how these challenges are overcome.

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 issue to future courses in EECS.

[1] The distinction between a phone and a computer is rapidly vanishing, so the difference is rather artificial.
[2] As one wag put it: "Networking, just one letter away from not working."

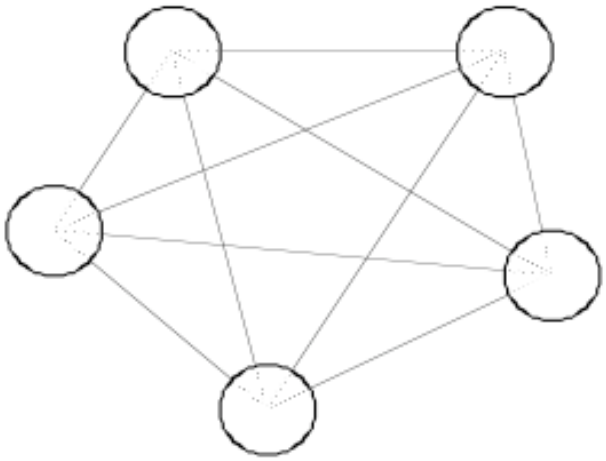

Figure 17-1: A communication network with a link between every pair of computers has a quadratic number of links. This approach is too expensive and is especially untenable when the computers are far from each other.

■ 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 computers together is ostensibly quite straightforward: one could simply connect each pair of computers with a wire and use the physical layer running over the wire to communicate between the two computers. 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, because there are about N^2 different bi-directional links in this design. 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 computers 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.

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 computers to run over each individual link—that is, it arranges for the network's links to be *shared* by different communications. In addition to

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

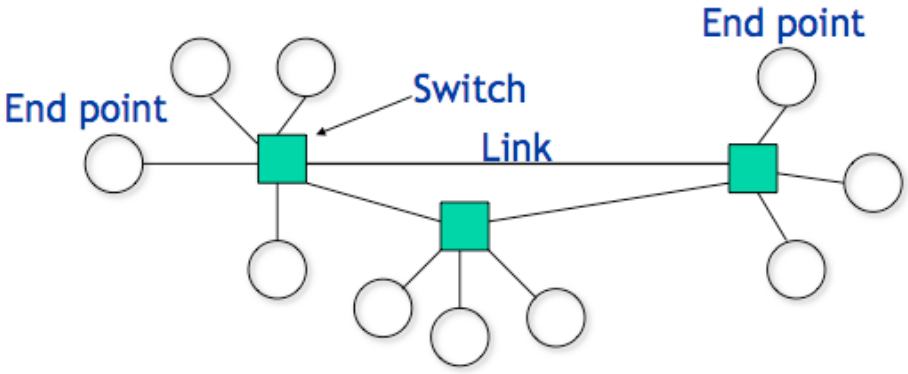

Figure 17-2: A simple network topology showing communicating end points, links, and switches.

the links, the switches themselves have some resources (memory and computation) that will be shared by all the communicating computers.

Figure 17-2 shows the general idea. A switch receives bits that are encapsulated in *data frames* (in some networks, these are called *packets*, as explained below) 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.

We will use the term *end points* to refer to the communicating computers, 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* 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 computer-to-computer information transfer sessions, and to determine the link(s) along which to forward any given data frame. This task is essential because a given physical link will usually be shared by several concurrent sessions between different computers. 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".

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

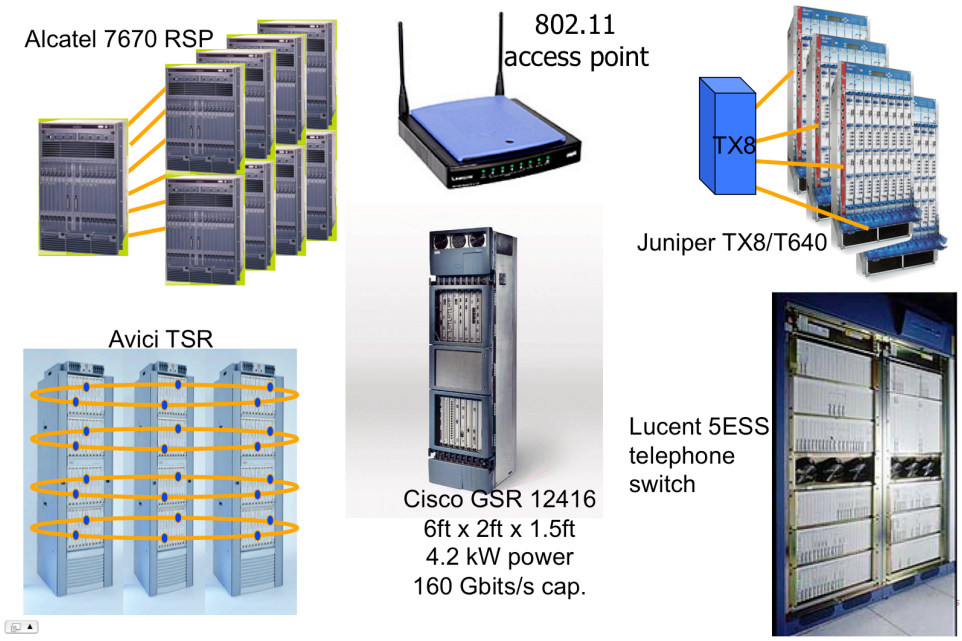

Figure 17-3: A few modern switches.

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 mechanistic. The mechanistic difference, which 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 same abstraction is 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 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):

[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.

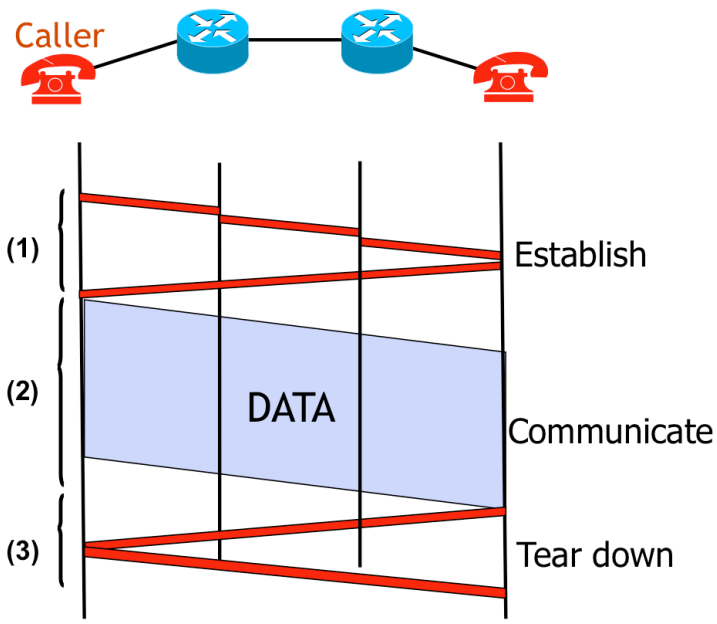

Figure 17-4: Circuit switching requires setup and teardown phases.

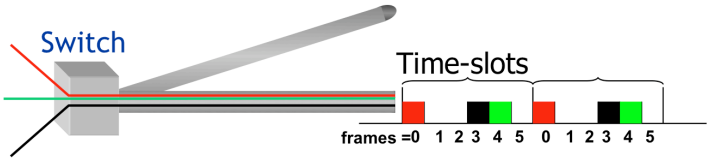

Figure 17-5: Circuit switching with Time Division Multiplexing (TDM). Each color is a different conversation and there are a maximum of $N = 6$ concurrent communications on the link in this picture. Each communication (color) is sent in a fixed time-slot, modulo N.

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 broken into some number N of virtual "channels" (or time slots), such that the ratio C/N bits/s is sufficient for each information transfer session (such as a telephone call between two parties). Call this ratio, R, the *rate* of each independent 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). 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 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 ith time-slice to the ith 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 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 computers 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

[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 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.

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 (email and file downloads are examples); for this reason as well, a different sharing strategy is worth considering.

Packet switching doesn't have these drawbacks.

17.3 Packet Switching

The best way to overcome the above inefficiencies is to allow for 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 it. This information is added in the form of a *packet header* 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 each packet. Packets usually also include the sender's address to help the receiver send messages back to the sender.

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* (or *forwarding table*; the distinction between the two is sometimes important and will become apparent in a later lecture in the course). This lookup returns an outgoing link to forward the packet on its way toward the intended destination.

While forwarding is a relatively simple 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 several types of routing protocols that one might consider, and we will study two common classes of protocols in later lectures. For now, it is enough to understand that the result of running a routing protocol is to obtain routes (which you can think of as paths for the time being) in the network to every destination—each switch dynamically constructs and updates its routing table using the routing protocol and uses this table to forward data packets.

Switches in packet-switched networks that implement the functions described in this section are often called *routers*. Packet forwarding and routing using the Internet Protocol (IP) in the Internet is an example of packet-switching.

17.3.1 Why Packet Switching Works: Statistical Multiplexing

Packet switching does not provide the illusion of a dedicated link, but it has a few things going for it: it doesn't waste the capacity of any link because each switch can send any packet available to it the needs to use that link, it does not require any setup or teardown phases and so can be used even for small transfers without any overhead, and it can provide variable data rates to different communications essentially on an "as needed" basis.

At the same time, notice that because there is no reservation of resources, packets could

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

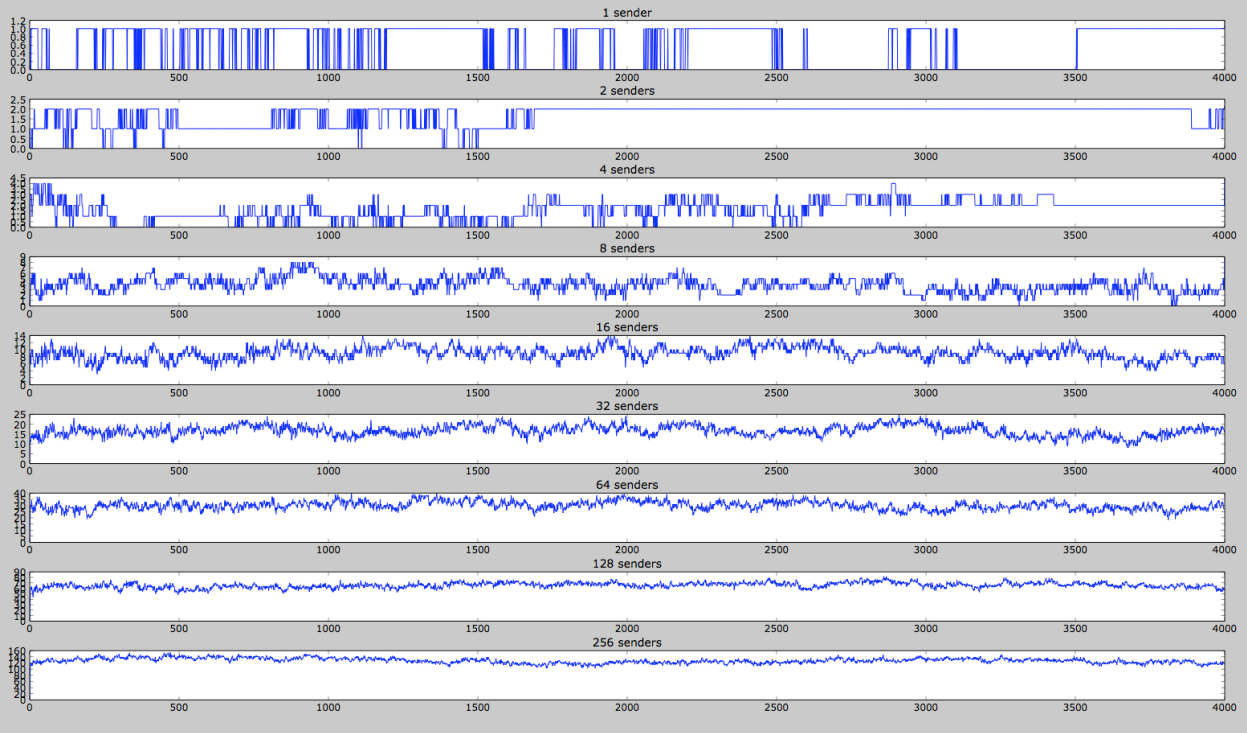

Figure 17-6: Packet switching works because of statistical multiplexing. This picture shows a simulation of N senders, each connected at a fixed bit rate of 1 megabit/s to a switch, sharing a single outgoing link. The y-axis shows the aggregate bit rate (in megabits/s) as a function of time (in milliseconds). 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").

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 *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 individual communicating pairs, 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.[7]

[7]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.

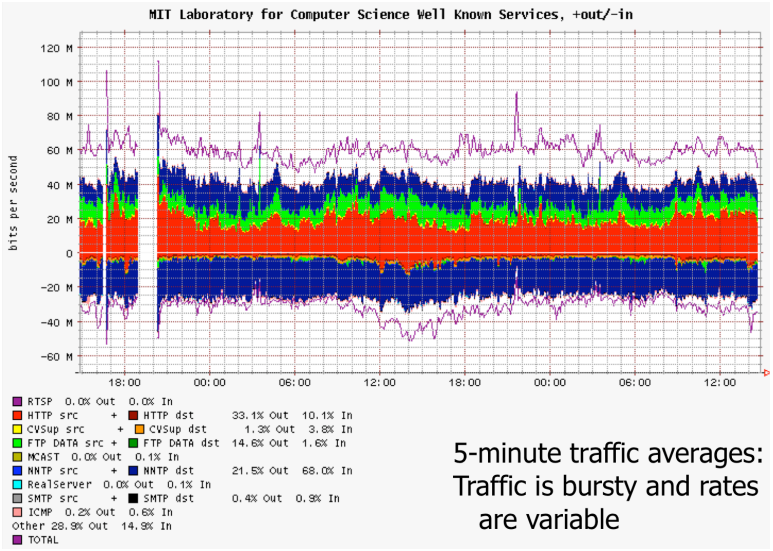

Figure 17-7: Network traffic variability.

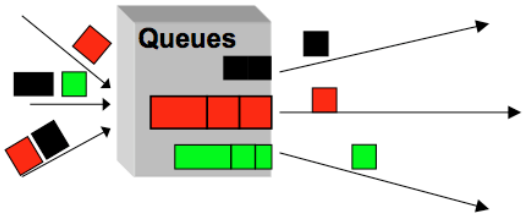

Figure 17-8: Packet switching uses queues to buffer bursts of packets that have arrived at a rate faster than the bit rate of the link.

An example is shown in Figure 17-6. The x-axis is time in milliseconds and the y-axis is 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, 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 17-7 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.

■ 17.3.2 Absorbing bursts with queues

Queues are a crucial component in any packet switch. They absorb bursts of data, so a key question is how they big they should be. The reason this question is important is that if you make them too big, all that happens is that delays grow (for queues by themselves don't make data transfers go any faster), but if you make them too small, the switch may drop

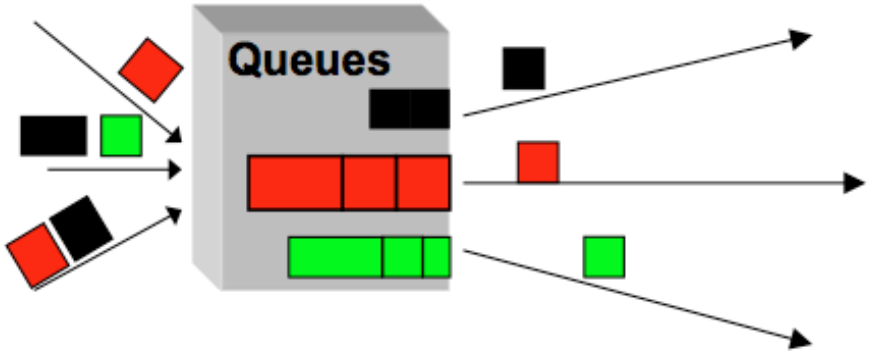

Figure 17-9: Packet switching uses queues to buffer bursts of packets that have arrived at a rate faster than the bit rate of the link.

too many packets. We won't be able to completely address this problem in this course, but it is a question that we will discuss further in a later lecture.

Queues add delay and these delays are variable. In fact, queues are only one of four different sources of delays in networks. The next section describes these sources and talks in more detail about queueing delays and how to analyze them. The key result that we will look at here is called Little's law.

■ 17.4 Understanding Network Delays

An important measure of performance of any communication network is the network delay, also called the network latency. There are four sources of delay:

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.

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 least, looking up the header of the packet in a table to determine the outgoing link. It may also involve operations like computing a packet checksum, modifying the packet's header, etc. 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.

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 delays in networks 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).[8] It's used widely in the performance evaluation of systems ranging from communication networks to factory floors to manufacturing systems.

For any stable (i.e., where the queues aren't growing without bound) queueing system, Litte'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. I.e., 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 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-envelop 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 17-10 shows $n(t)$, the number of packets in a queue, as a function of time t.

[8] 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.

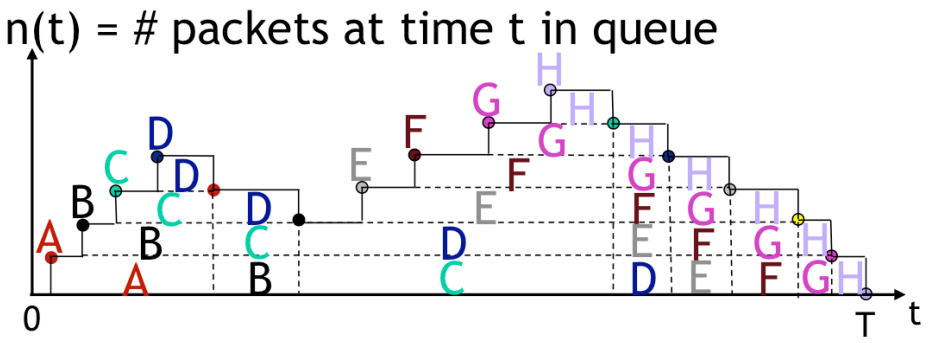

Figure 17-10: Packet arrivals into a queue, illustrating Little's law.

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 17-10. 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 total delay experienced by all packets.

Hence,

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

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

■ 17.4.2 Applying Little's Law

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

MEng ratio. 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?

Mortality statistics. While reading a newspaper, you come across a sentence asserting that *"less than 1% of the people in the US die every year"*. Using Little's law (and some common sense!), explain whether you would agree or disagree with this *assertion*. Assume for simplicity that the number of people in the US does not change.

A little queueing. 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?

A Little TDM. Alyssa P. Hacker is running some experiments on a TDM circuit-switched network switch. All packets are of the same size. The rate of the link connected to the switch is $\mu = 1000$ packets per second. The link can support 24 concurrent conversations between different pairs of nodes.

In a given experiment, she picks a random number of conversations, r, and has them communicate using the switch. She measures the expected per-frame delay at the receiver (D), and also monitors the number of frames in the switch with time (if a frame shows up earlier than its allocated time slot, the switch stores the frame until it is ready to be sent). The expected number of frames in the switch is N. No frames are dropped in her experiments.

Somewhat to her surprise, she finds that the product $D \cdot \mu$ is often not equal to N. Explain this "violation" of Little's law.