WASHINGTON UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CS423 Computer Communications: LANs
Fall 1995

This is the sixth of probably 7 lectures on the Data Link Layer. In the first two lectures we studied some common issues to all data links: first framing, and then error detection. In the next two lectures we studied error-recovery for point-to-point Data Links, starting with stop-and-wait, and then on to more pipelined sliding window schemes.

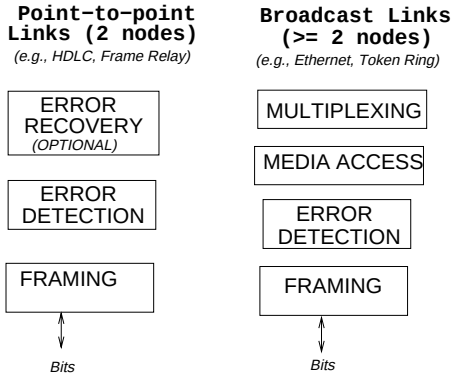

Figure 1: Data Link sublayers. We now move to the right and study multiaccess links.

If you recall the Data Link sublayer model (see Figure 1), we are going to make an important transition and study Data Links that connect multiple senders and multiple receivers. Since most of these links have very low error rates, error recovery is almost never done on these links. In all the links, we study the link is "sequential" — only one sender is allowed to send at a time or chaos ensues. Thus we have to solve another problem in place of error recovery — medium access. Each sender that has data to send tries to "grab" the link for transmission; the successful sender's transmission is then placed on the link and *all other stations* can receive the transmission.

Such links are typically given several names, each of which conveys some aspect of their natures. First, they are sometimes called *multiaccess links* because there are multiple nodes that may simultaneously access the link. Second, they are called *broadcast links* because every transmission can be heard by all other stations. This is extremely useful because it allows us to send a single message to all other stations or to certain other stations. Third, they are often called *Local Area Networks* or LANs because the geographical area serviced by a LAN is local and small – from 1-10 Kms, covering an office, a building, or at most a campus. The geographical area is typically limited because all stations *share* a common link, and the cost of medium access grows larger with the worst-case propagation delay on the link; the propagation delay in turn grows with the span of the multiaccess link. This makes LANs of very large span infeasible.

The two new features of LANs that we see immediately are medium access and *multicast* (multicast is a more refined form of broadcast where a frame is sent to a set of receivers, as opposed

to all receivers. Of course, in a LAN typically all receivers receive all transmissions. However, the ones that the message is not addressed to filter out the transmission in hardware; thus the software does not have to deal with every irrelevant message on the LAN. Examples of multicast use include "A message to all flight simulators containing new position of a plane" or "Request to any fileserver on link to send a font". The two uses of multicast, addressing *all* or *any* of a set will be examined later. However, note that multicast is almost an accident of the technology — because all receivers get all transmissions anyway, it comes for "free". However, we will see that in the wide-area network (the network that links together LANs), multicast is emphatically not free.

1 Why LANs

Why are LANs worth building and studying? Why can't we just connect all workstations in an office to a central router (or to many routers and connect the routers with point-to-point links)? The main reason is cost: router ports are expensive and each router has a large fixed cost for its power and packaging. Since most routers have only a limited number of ports, adding more routers increases the fixed cost of the router boxes and adds cost for the individual wires connecting the routers. On the other hand, the cost of a LAN attachment is cheap (Ethernet attachments can be had for under 50 dollars) because the control circuitry was reasonably simple and has been standardized into chip sets: as for the wire itself, its a cheap dumb wire, and everyone shares it. Thus, LANs connect up all computers in a building, office, or campus with say 200-1000 users. Saves wiring costs to share one wire, and cost is very important in these days of cheap PCs and PDAs.

Why make a distinction between LANs and wide-area networks? Because of the smaller area covered, LANs typically offer higher bandwidth and lower error rates for the same degree of connectivity and cost (when compared to wide area solutions). Why is more bandwidth and lower error rates for local group of users a good idea? Its worth it because a great deal of user traffic has high access locality — it is quite likely (e.g., file servers, messages to group members) that the bulk of your messages will be within your group, building, or campus. Thus it pays to have high-performance LANs.

2 Statistical Multiplexing and LANs

One way to solve the medium access control problem is to use the multiplexing solutions we used in the physical layer lectures. For instance, we could used time or frequency division (TDM, FDM) multiplexing to strictly divide the available bandwidth among all possible senders. Let us call *strict multiplexing* any scheme such as TDM or FDM where a user is given a fixed allocation *regardless* of whether the user has data to send or not.

Strict multiplexing is not a good idea because user traffic is typically very *bursty*. Bursty means that the traffic has a high peak/average ratio. Thus for many hours a workstation may only transmit some piddling control traffic; then in a single second it may send a Mbit of data as part of file transfer. The average rate per second is small, but the peak is much larger (by a factor of at

least 3600 in this example.). Thus the data traffic of a user looks like a burst of data, followed by almost idle periods, then another burst, and so on.

For bursty traffic, a strict division of the available bandwidth into smaller pieces for each user, means that each user is limited to a very small piece of the bandwidth. This is because there are a large number of potential users on a LAN with hundred stations. For example, in a 10 Mbps (Mega Bit per second) LAN with hundred users, each user would only get 0.1 Mbs using strict multiplexing. However, in a bursty model it is rare that more than a few users are busy at the same time! Thus it pays (in terms of both throughput and latency) if busy users can get access to the bandwidth of idle users. This is called *statistical multiplexing*. It is called statistical, because the multiplexing scheme attempts to take advantage of traffic statistics according to which most users are rarely busy at the same time.

You've encountered statistical multiplexing in many places. A private line isn't reserved for every two people who want to call each other; a part of the network bandwidth is not reserved for every pair of stations on the Internet that may communicate. Mathematically, strict multiplexing means that every source gets a bandwidth of B/N, where N is the number of possible sources; statistical multiplexing attempts to give each user, B/x, where x is the number of users that currently wish to use the system. Typically, N is large (100 -1000) while x is small (1-10). Hence the advantage of statistical multiplexing.

Consider the following concrete example. Suppose 100 users each transfer a 125 kbyte file every 2 hours. At 10 Mbps, a 125 kbyte file takes 0.1 seconds to transmit. If the user requests arrive randomly, a user has a high chance of having the request arrive when no other request arrives (i.e., when $x = 1$). (In an available space of 7200 seconds, only 10 seconds are used for transmission, which leaves a chance of about 3500/3600 of finding no concurrent request.) If no other user is transmitting a file at the time a request arrives, the file transfer time is 10 second. With strict multiplexing, it would take 10 seconds! Clearly, users would prefer statistical multiplexing because of the improved latency. Consider now a scenario where only half the users are using the system but they are all continuously busy — strict multiplexing would give only half the possible throughput that statistical multiplexing can achieve. Thus statistical multiplexing improves both latency and throughput, making both users and system managers happier.

It is important that statistical multiplexing must preserve fairness or protection. One way to improve throughput when two users are both busy is to give all the bandwidth to one user. Thus to be fair, statistical multiplexing must ensure that the bandwidth is allocated fairly among all the users that wish to use the link.

It is easy to implement *centralized* solutions to statistical multiplexing. Every user who wishes to transmit sends a request to a resource manager. The resource manager keeps each user's requests in a separate queue, and gives round-robin service to each busy user (i.e., it will not give another service opportunity to a user until it has given a service opportunity to all other busy users.)

If all users are at a single node, this solution is very efficient. However, in a LAN this kind of centralized solution is inefficient. First, the central resource manager is a bottleneck and failure prone. Second, how are requests made? If they are made using the common physical link, we have to recursively solve the problem of arbitrating between simultaneous requests and we are almost back to where we started from. On the other hand, if we use separate wires between each user and

the resource manager, we increase wiring costs. Third, in a LAN we have to pay three propagation delays for each request: one to send the request to the manager, a second to send the grant to the user, a second for the user to say that it has finished using the link. This increases latency in an idle system.

Thus all LAN solutions use *decentralized* solutions to the problem of medium access; however, the solutions attempt to provide statistical multiplexing. As we can see from the example above, the fact that the stations are separated and need to pay at least one propagation delay P to communicate, is going to limit the efficiency of medium access. We will see that in both schemes we study, Ethernet and Token Rings.

3 Ethernet

The basic idea in Ethernet is used everyday by people at meetings. Would-be speakers wait for a pause in the conversation (called *carrier sense* in Ethernet terminology). After the silent period, one or more speakers may try to break in. If there is more than one, they may detect that and stop (called *collision detection* in Ethernet terminology). In some real meetings, the stronger personalities keep speaking and sometimes win the floor over politer contenders. Ethernet stations are much better behaved. They wait a random period of time (called *backoff* in Ethernet) and then try again, repeating the process if necessary. The three main mechanisms in Ethernet thus are carrier sense, collision detection, and backoff. It is sometimes called CSMA/CD (Carrier Sense Multiple Access with Collision Detect.)

The figure at the end of this handout (called CSMA/CD operation) shows a nice example of a collision between two stations A and C. A starts sending at time t_1. Notice that A's signal begins to propagate (at roughly the speed of light) *in both directions* along the physical wire. Think of a wave rippling out in both directions along a long thin channel. At time t_1, A's wave has reached the left end of the wire and has reached station B and is still propagating to the right. A's wave has still not reached station C when C begins to transmit. C's signal begins to move to the left towards A and to the right. At time t_2, the signal from C and A meet (somewhere between C and B). The result is now a superimposed signal (shown cross-hatched) and represents a "collided" signal. The collided signal now begins to radiate outwards in both directions. At time t_3, the collided signal has reached the right end of the wire and has almost reached the left end.

The example shows that it takes time for both data signals and collisions to propagate along the Ethernet wire. For example, it is easy to see that C detects the collision first, then B, and A detects the collision after B and C. It is incorrect (but natural) to think of a collision as being an instantaneous event that everybody observes (as we do in meetings).

3.1 Historical Precursors of Ethernet

Ethernet was a fairly natural evolution of an earlier technology that was deployed at the University of Hawai. The university had lots of workstations at various islands in Hawai, and a central computer. The stations had a radio transmitter. They had two channels: a contention channel that all stations shared, to send to the computer; and a reverse broadcast channel that the computer

used to broadcast responses to all workstations (stations that did not want transmissions filtered them.)

The stations basically used carrier sense to grab the channel. Frames were small. However, collision detection is done in software. Unless the central computer sent back an ack, the station retransmitted after a random period. Thus two important differences between Ethernet and Aloha are: a) collision detection is done after frame transmission (as opposed to during frame transmission as in a Ethernet) and in software (hence its slower). b) The random backoff algorithm uses a fixed number of random slots (Ethernet as we will see uses a more dynamic backoff algorithm.)

One problem with Aloha is that a frame is vulnerable to frames that start before the frame is transmitted as well as frames that are sent after the frame is transmitted. Figure 2 shows this. Consider all frames of fixed size that take T time to transmit. A black frame sent at time t can collide with all frames sent between times $t-T$ and $t+T$. If a white frame is sent at $t-T$, the rear of the white frame will collide with the front of the black frame. Similarly, the rear of the black frame can collide with the front of a white frame sent at time $t-T$. Thus the vulnerable period of a frame is $2T$, where T is the frame transmission time.

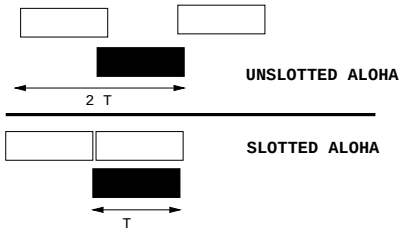

Figure 2: Aloha and Slotted Aloha

As a response to this, it was proposed to use slots of fixed length T and only allow transmissions at the start of a slot. In Figure 2 we see now that a black frame can only collide with frames that arrive T units before it is transmitted. Thus the vulnerable period is only T, a factor of two less than ordinary Aloha. Unfortunately, slotted Aloha requires fixed size slots (making them too small would be inefficient when you have large amounts of data; making them too big, means lots of idle time for small frames). It also requires synchronized clocks (easy when you have a central computer distributing clock ticks but not easy in a decentralized setup like an Ethernet). Thus Ethernet went on to use ordinary Aloha like mechanisms, with some refinements.

3.2 Why Collision Detection and Semi-reliability

Why does Ethernet do collision detection during transmission (unlike Aloha)? Firstly, as we will see later, it is possible to do collision detection on a wire in a way that is hard to do on a radio medium. Secondly, collision detection during transmission is much more efficient than collision detection after transmission, *especially for large frames.* Consider a large frame of 1500 bytes that is involved in a collision. Ethernet will detect the collision and *abort* the transmission after 64 bytes of the frame are sent. Aloha will send the entire 1500 bytes and detect the collision later when it does not get an ack. Thus Ethernet wastes only 64 bytes, while Aloha wastes 1500 bytes.

Since collisions are frequent, this wastage can be considerable. Aloha gets around this by using small (80 byte frames.) However, that is inefficient for large data transfers since it increases the number of frames transmitted and the communication overhead (headers, framing). Thus the Ethernet method of immediate collision detection allows the use of large frame sizes without a corresponding increase in wasted bandwidth. The immediate collision detection also allows a quick response to collisions unlike the Aloha method (which is based on timers and retransmissions as in Stop-and-Wait).

The Ethernet has another goal which is semi-reliability. A reliable Data Link must guarantee that all lost or corrupted frames are resent, including those lost due to ordinary random bit or burst errors. However, Ethernets have good random bit error rates, so that kind of error recovery is best left to higher layers. However, collisions can be quite common in the Ethernet. (As an example, perhaps one frame in a million frames may be corrupted due to bit errors on the Ethernet; however, one in every 10 frames may be involved in a collision on a busy Ethernet. Thus the Ethernet seeks to be semi-reliable: *the Ethernet does its best to retransmit (in hardware) all frames lost due to collisions.*

This principle leads to an important corollary. If any sender is potentially involved in a collision, it must know about it reliably so that it can retransmit. This corollary leads to certain mechanisms like jamming and the need for a minimum frame size that we study about next.

3.3 Ethernet Design Details

We proceed to examine details of the Ethernet scheme. We now study the mechanisms for collision detection, semi-reliability, and adaptive backoff.

3.3.1 Collision Detection

First, we examine collision detection. In early Ethernet implementations, this was done quite simply by using Manchester encoding with a net DC bias that is non-zero. Suppose a 1 is encoded by 4V followed by 0V, and a 0 is encode by 0V followed by 4V. Then the average DC voltage is 2V, regardless of the bits sent. Finally, whenever we have a collision, the net DC voltage rises (two 2V signals, for example, lead to a voltage of 4V). Recently, there is a committee called IEEE 802 that has standardized a variant of the original Ethernet that uses Manchester encoding biased at 0V. Thus the newer versions use a different scheme to detect collisions.

Carrier sense can be done either by sensing a DC voltage (in the biased case) or sensing the receipt of a Manchester signal (alternating voltage pulses). I'm pretty sure its done by the latter; when the clock recovery circuit senses a recovered clock, there is a frame being transmitted.

3.3.2 Semi-reliability Mechanisms: Min Frame Size and Jamming

Next, we describe how semi-reliability is ensured. To understand this, we first need to understand the concept of a *slot*. A slot is equal to $2T$, where T is the maximum propagation delay between two stations on the Ethernet. A slot is important, because it is the maximum time it takes for

a sender to detect a collision involving its own transmission. Suppose A is on the left end of an Ethernet wire and it sends a frame at time $t = 0$. At time $t = T$, the start of A's frame reaches the right end of the wire. Suppose just before A's frame reaches, a station D at the right end of the wire starts transmitting (carrier sense does not reveal any problems to D). Then the two signals collide but it takes another T units of time (see the example above) for the collided signal to travel back to A. The Ethernet Physical topology requires a maximum value of $2T = 51.2usec$.

So now consider a scenario where A sends a 1 byte frame and then stops transmitting. If A stops at this point, A may not receive this collision information about its frame. Thus waiting for 51.2 usec after transmission (as was suggested in class) does not help. This scenario is shown in Figure 3.

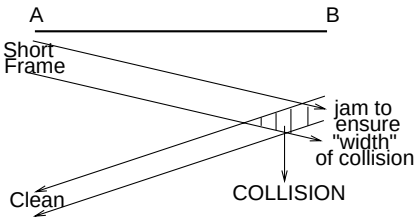

Figure 3: If A sends a short burst and does not send a min frame size, A may receive a wave from B that does not show a collision as the original signal from A has already propagated to the right. Thus B detects the collision but A does not.

Assume that A sends a short frame (time flows downwards) that moves to the right. Assume that just before transmitting B transmits. Assume that each wave moves outwards from its source in both directions independently of other waves. The net effect at any point is assumed to be the sum of any wave values currently at that point (linearity assumption). Thus B will detect the collision and stop transmitting (after a short time called the *jam* period, explained later). However, by the time B's signal propagates to A, A has stopped transmitting. Thus the net effect at point A is just the effect of B, and so A does not detect a collision. Thus A will not retransmit, thus violating our goal of semi-reliability.

However if A sends at least 64 bytes at 10 Mbps (the speed of the Ethernet), it will take A at least 51.2 usec to finish transmitting. In the meanwhile, A is guaranteed to detect the collision. Thus the Ethernet requires a minimum frame size of at least 64 bytes. If users have smaller amounts of data to be sent, they must add a special pad character to pad out the frame to 64 bytes. (This in turn necessitates a length field in the Ethernet header so that receivers know the length of the real data in order to remove the useless pad information.)

A second curious mechanism necessary for semi-reliable operation is called *jamming*. Suppose in the example above that B detects its collision with A's signal it stops its transmission (called *aborting*) after sending only a few bits. Then A may only get back a few collided bits and may not reliably be able to detect a collision. To prevent this, B sends a fixed number of bits to enforce the collision. The number of bits is smaller than the slot size (about 20 bits) and is required to ensure that A gets enough collided bits (i.e., the collided wave has a minimum "width" to detect a collision).

3.3.3 Adaptive Backoff Times

First, notice that the slot time is the vulnerable time for Ethernet. Once a frame has been transmitted for 51.2 usec without a collision, there should be no collisions for that frame. This is because enough time has elapsed for all other stations to sense the transmission of the frame and to send back collided bits (if there was a collision). After this period, the transmitting station has captured the wire.

What should we do after a collision. It makes sense to wait a multiple of a slot time. Suppose two stations collide. Then the ideal behavior is that one station transmits immediately, and the other waits a slot time to sense the medium. The second is guaranteed to sense the transmission of the first and defer its transmission. This is because the first will have detected a collision at most T time after the second, and the transmission of the first will arrive at the second at most T time after the first detects a collision.

One way to hope for this behavior is for each of the two colliding stations to flip a coin; if the coin comes heads, the station transmits immediately. If not, it waits one slot. (In all cases, it senses the medium before attempting the transmission and waits until any sense frame is finished before sending.) Thus with 50 percent probability, one station will wait one slot, and the other zero (i.e., after 1 slot the collisions are resolved as the second one now transmits after the first ones transmission is over, a total cost of 2 slots.) They will choose to wait the same number of slots also with 50 percent probability, which leads to a collision. However, then they iterate this process. This like having two people keep tossing coins until one wins. In practice, this takes only a few tries (at most two or three) before one wins. (The probability of failure falls off exponentially with the number of tries.)

That works for two stations. Suppose we had 16 station and each tossed a coin. Now we have a collision only if 1 station picks head and 15 other stations pick tails, or vice versa. If you work out the numbers, the probability of collision is very large. So simple coin-tossing does not work. However, if you toss a 16-sided coin (i.e., pick a random number between 0 and 15), then its more likely that there is unique person that picks the lowest number. Then the remaining 15 have to compete again.

So here's the problem with a static scheme. If you pick a random number between 0 and N, and N is much smaller than the number of simultaneous contenders, then we can keep having collisions. But if you pick N to be very large, there is another problem. If you pick N to be 15, then even if there are only two contenders, the average number of slots required to resolve the collision is at least 8. Thus a static scheme either does not work for a large number of colliders or is inefficient for a small number.

Thus it pays to use a dynamic scheme. Try to think of one for yourself. The Ethernet scheme starts with N small (equal to $2^0 = 1$) and roughly doubles after each unsuccessful collision. This is what is called *binary exponential backoff*, because the backoff time grows exponentially (by a factor of two) for every unsuccessful collision. You reset the backoff time after your frame (not others frames) is transmitted successfully.

3.4 Ethernet Code

The code for the Ethernet is shown at the end under the figure called Control Flow summary. It is a flow chart taken from the original Ethernet spec.

The code should reveal some subtle aspects: for example, on receipt there is a check for min frame size (why can this occur, since all frames are supposed to use a min frame size?) before checking the checksum (also called frame checksum or FCS).

3.5 Ethernet Frame Format

A specification must not only tell you how the protocol works (both normal operation as well as initialization) but also tell you frame formats. The Ethernet header is shown in Figure 4. Notice the preamble, a sequence of alternating 0's and 1's which help the clock recovery circuits get into synch, followed by a 11 to help the sender know when the frame starts. Every Ethernet station has a 6 byte unique address (unique worldwide), and the first 6 bytes identify the intended destination (multicast transmissions use special multicast addressing). Notice the length field and the pad field as well.

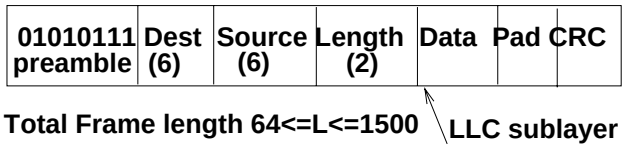

Figure 4: Ethernet Header

3.6 Implementation Limits

Ethernets come in two flavors: the older clumsier thick wire Ethernet and the new thin wire Ethernet that is easier to wire. There are a number of limits to installations (see Text, Section 3.4.1). For example, an overall distance limit of 2.5 kms. Actually a single wire can only go for 500 km before the signal strength gets unreliable. To go more distance, we need a device called a repeater that clocks in received bits and clocks them on the other side, with a "boosted" voltage level. Receivers add troubles of their own (for collision detection etc.) and we can have at most 4 repeaters.

The collision detection etc, is typically done at the Ethernet transceiver which is next to the cable; the frames waiting to be sent are kept at the adaptor which is a card that plugs into the computer.

3.7 Ethernet Pros and Cons

Ethernet has many purported disadvantages. Firstly, it is supposed to unpredictable because of random delays. So are all systems. The min frame size requirement is supposed to be wasteful.

However, the smallest control frames sent by higher layers are typically at least 40-50 bytes in length, and not all frames are are this small, so that's not as big a deal as one might think.

The three real disadvantages are that a) Ethernet has a limited distance, 2.5 km b) Its performance gets worse at high loads c) It does not work well on fiber but needs coax or twisted pair. The first disadvantage goes away with the use of bridges; the second is perhaps not as serious as it sounds because most LANs are typically underutilized. Also real measurements indicate that Ethernet does much better at high loads than the modeling people believed. However, the third reason led people to propose the use of token rings in the late 70's.

However, Ethernet is significantly cheaper and a very mature technology. This has lead to a new Ethernet renaissance with a proposal for a 100 Mbps Ethernet that has a smaller distance span (10 times less, which leaves the slot time expressed in bits unchanged.)

3.8 Cost of Statistical Multiplexing in Ethernets

Suppose we increase the distance of an Ethernet or increase the transmission rate. It is easy to see that the scheme gets more inefficient. If we increase the transmission speed or max distance by a factor of 10, the min frame size to detect collisions would go up to 640 bytes, which means that most small frames (of say 40 bytes) which have only about 50 percent padding overhead currently, will now have 500 percent overhead. Since 80 percent of the traffic sent is little frames, this is inefficient.

(As an aside, the 80-20 rule for frame sizes is a useful rule of thumb: 80 percent of the traffic consists of little frames of size 40 bytes and 20 percent consists of larger frames of 576 bytes or more. Frames of size 570 bytes or so are common because of some disk block sizes (512 byes) plus higher layer headers.)

Thus the cost of statistical multiplexing in Ethernet is proportional to the product of the propagation delay and the transmission speed — in other words, the pipe size. To underscore this, consider the two extremes. If the propagation delay is very large, neither carrier sense or collision detection gives you useful information. If the transmission speed is large relative to the prop delay, then you send a large number of bits before you learn anything useful. As we have seen, the pipe size of a system is a fundamental quantity. Note that this is why the 100 Mbps Ethernet uses a smaller distance.

4 The 802 committee

The IEEE 802 committee is in charge of standardizing LANs. It standardized at least three LANs: 802.3 which is a slight upgrade of the old Ethernet, token bus (which essentially consists of running a token ring-like protocol on a coaxial cable bus using a logical ring), and 802.5 (which was a slight upgrade of IBM's token ring).

The token bus was pushed vigorously by GM for many years but has now died out. We have already studied 802.3. We will study the IBM token ring (which we will sometimes refer to as 802.5). We will also study an enhanced token ring called FDDI that offers better throughput at

high speeds and support for guaranteed latency for multimedia traffic.

5 Token Rings

A token ring is easily explained by another way to conduct a meeting. The meeting participants are arranged logically in a ring. The current speaker is the one with the token. After he or she is done, they pass the token to the next person in the ring. People who do not want to speak simply pass the token. This scheme is quite tedious if only a few people want to transmit (i.e., its latency is poor at low loads) and there are a large number of people that the token must pass through. However, at high loads, its throughput is very good. The following is a good performance rule-of-thumb: Ethernet provides good latency at low loads, and token rings provides good throughput at high loads.

Token Rings became popular in the late 70's and 80's because of the outcry against Ethernet for reasons described above. The most rational reason for Token Rings are its good performance at high loads and the fact that its easy to build fiber optic token rings (as all the stations are connected by point-to-point links.) Token rings appear to be simpler and also to offer predicatable service; however neither statement is true when tokens get lost, as we will see below.

5.1 Requirements: One-bit delays and Frame Stripping

Two subtle aspects of real token rings are the need for *small (as low as one-bit) token passing delays* and the need for *frame stripping.*

Consider first the token passing delay. A token consists of about 3 bytes in the IBM ring (this includes space for priorities as well as framing delimiters). The IBM Ring can operate at 1 Mbps. If each station were to wait to store an entire token before forwarding it, the token passing delay at each station would be the time to receive 3 bytes or 24 bits at 1 million bits/second, which is 24 usec. If we had 200 stations in the ring the total propagation delay contributed by token forwarding would be about 5 msec which is very large. (The wires themselves will only contribute 3.5 usec per km, which is a delay of only 7 usec for a ring with total wire length of 2 km.) This is unacceptable.

Stations that do not want to transmit can essentially repeat all bits with a one-bit delay. (Since there are point-to-point links between token ring attachments, an attachment has to read in an incoming bit and clock it out onthe outgoing line only one bit later.) The problem is only when a station wants to transmit and must look for the 3 byte token. If all stations want to transmit, and each of them delays each frame by 3 byte delays, the delay will add up.

All rings use approximately the same idea for avoiding store-and-forward delays. They use what we call *cut-though* forwarding: a node in the ring begins forwarding a frame before it has completely received it.

The second requirement is for frame stripping. Consider if a station A sends a frame F. Suppose no one wants to transmit and the frame F continues to circulate around the ring. One might think that this does not matter and that this frame will eventually be removed when some station wants to transmit and F runs into the token holder. Thus F is not consuming ring throughput unnecessarily

— its only using throughput that was not being used anyway.

However, while F will not consume ring throughput, its going to cost lots of unnecessary processing at the intended receivers and even cost wasted bandwidth at other links. Suppose F is a message to from A to a router R and this message is eventually destined to a distant station through a string of routers. If F circulates 1000 times, R will receive and forward frame F 1000 times. This causes 1000 extra copies of F to be processed by R and at all subsequent routers and links. It gets worse if F is multicast to a bunch of nodes. (Nodes other than intended receivers will filter out frame F in hardware based on the destination address.) Notice that this only causes performance and not correctness problems. F will typically carry a sequence number which will allow the final destination to remove all duplicates. Thus we need to strip every frame F sent on the ring.

One immediate idea is for the receiver of a frame to strip a frame that's sent to it. After it recognizes the Destination Address, the receiver can strip the rest of the frame. However this prohibits multicast, a useful LAN feature because the first receiver will strip the frame and prevent later receivers from receiving a copy. So can the last receiver strip the frame? That does work, but its often hard to know who the last receiver is and would require a complex protocol. (The software at a node can dynamically set the hardware to listen to a multicast address M; this may change the last receiver and the new last receiver must tell the previous receiver, which would be complex!) Thus the only feasible idea is *source stripping.*. Every source is responsible for cleaning up the mess its made!

6 IBM Rings

The IBM Token ring solves the two problems above as follows:

- *1 bit token passing delays:* A free Token and the Start of a Frame differ by at most 1 bit. When a node that wants to transmit sees the "token" bit that determines a token, it flips the bit on transmission. After the part common to the start of a frame, it sends its own data.
- *Frame Stripping:* After sending its frames, the sender does not pass the token until the first bit of its last frame sent arrives back. The station then keeps stripping the received bits until it sees the end of its last frame. This scheme allows the sender to avoid checking the source address in received frames, but it does waste some bandwidth.

6.1 Implementing One-bit delays

To see the first solution in more detail, we need to see the frame formats of tokens and data frames. This is shown in Figure 5. The numbers represent field lengths in bytes. The first two bytes of a token and a data frame are identical *except for a single bit.* The first byte is a framing delimiter (Start Delimiter or SD) that is a special physical layer symbol; similarly the last byte is a special physical layer symbol called the End-Delimiter. (The IBM ring used Manchester encoding and used invalid manchester patterns such as two consecutive highs or lows to create SD and ED.)

Thus framing is easy with help from the physical layer. The AC or Access Control field differ by one bit for a token and a data frame. The source and destination addresses and checksum (FCS) are just as in the Ethernet.

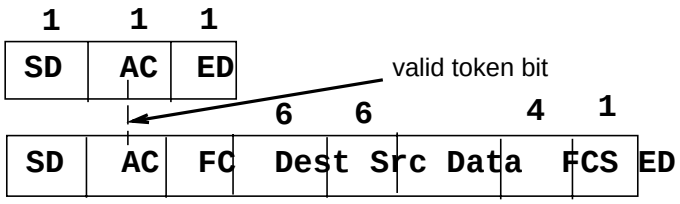

Figure 5: Token Ring Headers

Thus a station that wants to send data waits till it sees an SD and an AC with the token bit clear. When it sees that it flips the token bit before repeating it. It then keeps sending the rest of the AC bits. Then it sends the FC bits and the remaining bits of the frame it wants to send. Thus the forwarding delay is kept at 1-bit.

This form of cut-through forwarding helps latency but is dangerous. A single-bit error can cause the start of a data frame to look like a token and corrupt the frame. However, a single bit error corrupts a frame anyway! In general cut-through forwarding tends to propagate errors.

6.2 Stripping

The IBM ring has a fairly easy protocol to strip frames. Consider the simplest case when the transmitters send 1 frame per token opportunity. If the frame is large compared to the ring length, the first bit of the frame will come back to the sender before the frame is finished being transmitted. For example for a 10 Km ring, with a propagation delay of 50 usec, and with a speed of 1 Mbit/sec, there are only 50 bits that can be stored on the ring. Even if we have 100 stations that contribute 1 bit delay each for repeating, the total storage of the ring is 150 bits, which is 20 bytes. Almost all frames will be larger than this. These are typical numbers for the IBM Ring.

If the first bits of a frame do not arrive before the frame is finished being transmitted, then the transmitter *waits until the first bits return before releasing the token.* Thus if the transmitter has sent only one frame, it can keep stripping (after it has released the token) until it sees the next End Delimiter signifying the end-of-frame. If it sends multiple frames, then it can keep a count of the number of frames sent and strip that many frames after releasing the token. (Each of these schemes have some strange consequences when errors occur; however, ignore those effects initially.)

In general multiple frames can be sent, subject to a token holding time TH. A station can send multiple frames on each token opportunity such that the total transmission time does not exceed TH.

6.3 State Machine

A simplified state machine for a sender in the IBM ring is shown in Figure 6. A station stays in the REPEAT state until it sees a token. It changes the token to a SOF (for Start of Frame) and then

goes into TRANSMIT FRAME-1 state where it sends data. From this state there are two possible exits. If the transmitter receives the SOF (Start of its own Frame) before it finishes, it goes into TRANSMIT FRAME-2. If the transmission finishes before the SOF is received, the transmitter goes into a TRANSMIT FILLER state, where it transmits a "fill symbol" which is sort of like a pad character in Ethernet.

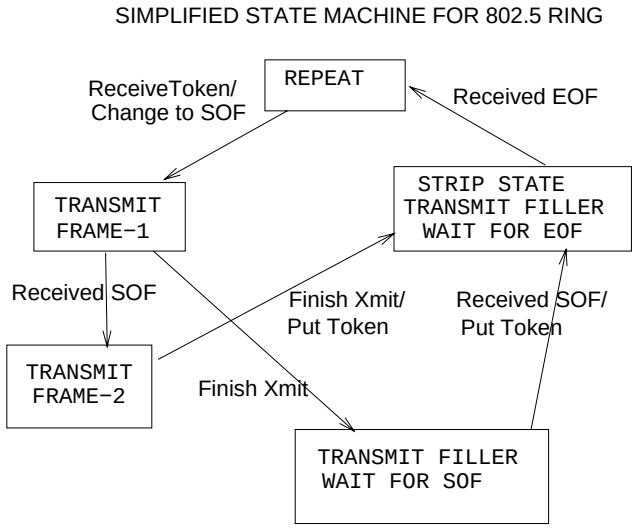

Figure 6: Token Ring State Machine

In TRANSMIT FRAME-2, the transmitter waits till it finishes transmitting and then goes into the STRIP state. In the STRIP state, the transmitter ignores everything it receives until it receives its last EOF (the figure shows the case when only a single frame has been sent; in case $N > 1$ frames have been sent, the sender must wait till N EOFs return). Similarly, the transmitter stays in the TRANSMIT FILLER state until it receives the first bits of its first frame back; then it can release the token and go into the STRIP state. Finally, you can exit the STRIP state (and go back into REPEAT) after receiving the last EOF. Notice that the token is only released just before entrance into the STRIP state.

6.4 Ring Reliability

An important question in understanding protocols is: how do rings initialize? This is closely related to the question — how do they deal with failures. The most prominent failure is the loss of a token due to a bit error or due to a node crashing. Node crashing seems even worse, because it seems to imply that the entire ring breaks.

All reliability in the IBM Ring revolves around a special node called the monitor node (the monitor node can be elected if the previous node crashes.) The monitor must be initialized with an estimate of the total token passing delay, TR, around the ring. (This is equal to a token holding time at each node plus the propagation delay.) The monitor sets a timer equal to $k * TR$, every time it sees a free token, where k is a small constant. If a free token does not arrive before the

timer expires, the monitor infers that the token is lost.

To be on the safe side, the monitor first sends a special PURGE symbol for a period equal to $k * TR$. When other nodes receive the PURGE, they go into REPEAT mode. Thus at the end of this period the monitor can be reasonably sure that all other nodes are in REPEAT mode. Then it releases a new token. Notice that this protocol is highly dependent on correct time estimates, unlike stop-and-wait and sliding window protocols. Thus it will not work correctly if the timers are set incorrectly.

How do rings deal with node crashes? The main innovation (see Fig 7) is that the physical topology of the ring is more star shaped than circular. The wires go from the workstations radially to a wiring closet or concentrator). Several concentrators can then be hierarchically connected through a central concentrator. The wiring concentrator can monitor whether nodes are alive. If it detects the node has crashed, it bypasses the node (by short-circuiting the ring at the wiring closet).

Star-shaped rings are crucial for reliability. They do increase the net wiring cost from about $2\pi R$ (if the physical topology is a circle with radius R) to $2nR$ (for a star, where n is the number of nodes). Since $n > \pi$ for $n > 3$, this increases wiring costs. However, star-shaped rings fit in with the hierarchical way buildings are wired (wires from each workstation or PC to a wiring closet on each floor, and then wires connecting the closets). So the extra wiring cost is a reasonable price.

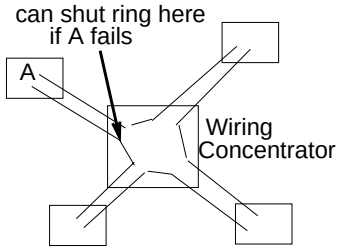

Figure 7: Star-shaped Rings

Note also the concept that the physical topology and the logical topology are not the same thing. The logical topology is crucial for the media access protocol but the physical topology can be chosen for mechanical, installation, and maintenance reasons. Thus Ethernet is a logical bus and 802.5 is a logical ring, but they are both installed in star or tree shaped physical topologies.

A crashed node can not only lose a token; it can also fail to strip frames it has created. This leads to orphan frames that can wander around for a long time. To prevent this, there is a bit in the AC bit, that can be called the "monitor seen" bit. Transmitters send frames with this bit cleared; if the monitor sees a passing frame with this bit clear, it passes it through but sets the bit. If it sees a frame with this bit set, the monitor assumes it has seen this frame already (i.e., its an orphan frame) and strips the frame on behalf of the (probably) crashed source. Notice that the monitor will never strip a frame until its been around at least once.

6.5 IBM Rings versus FDDI

In the late 80's the IBM Ring was considered too slow and the protocol not scalable to higher speeds. Since high-speed fiber was easily available a new protocol called FDDI was proposed. It had very similar ideas to the IBM ring but there are some contrasts.

In the IBM Ring, the source waits for head of frame to return before releasing a token. This is not wasteful at low speeds and distances — e.g., 10 mbit/sec, 10 km, 200 m/usec= 50 stored bits. But FDDI has a speed that is 100 times larger (100 Mbit/second) and 5 times the distance extent (50 km). Thus the number of stored bits will now be (in the worst case) about 25,000 - 50,000 bits. If a station sends a little frame (say 4000 bits) and waits till the little frame comes around, then we have a great deal of wasted throughput.

In FDDI, the source sends token as soon as it sends its last frame. Essential at higher speeds (100 Mbit/sec) and distances ($<$ 100 km). However, it now needs a different stripping algorithm. It has to strip based on source addresses: it strips all frames with its own source address. However, the source address comes at least 15 bytes into the frame. Thus the source can only know its one of its own frames until the first 15 bytes have passed through. It can then add an EOF to the frame but it has still created a *fragment*.

In the IBM Ring: each source sends as much as it can up to a fixed token holding timer. It is hard to guarantee delay bounds for delay-critical traffic (voice and video) when the number of stations is large. FDDI allows video and voice and offers predictable delay for delay sensitive traffic. It does so by measuring a token rotation timer; a station only sends data if the token returns early. Thus data traffic defers to real-time traffic if too much data has already been sent. We will discuss this in more detail next time.

7 Conclusions and Concepts

While we studied the details of three real LANs (Ethernet, token ring, and FDDI), three concepts are worth stressing in the midst of all these details. The first is the idea of statistical multiplexing (as contrasted with strict multiplexing) as exemplified in all real LANs. Stat muxing improves both latency and throughput. The second concept is the difference between cut-through versus store-and-forward processing of frames (as exemplified in token forwarding); this is becoming more crucial as transmission speeds increase and the effect of increased propagation delays is getting more important.

The third is the concept that the pipe size (transmission speed multiplied by propagation delay) seems to be a fundamental factor affecting the efficiency of distributed statistical multiplexing schemes. (In Ethernet, the pipe size determines the min frame size; in token rings it determines the wasted throughput if only 1 station on a ring want to transmit continuously.) Again this effect occurs because a single conversation must occur at a time on a LAN. We have to pay a price of a round-trip delay to determine that is safe to transmit, and in this time we could have transmitted some number of useful bits.

Stat muxing, the pipe size and cut-through forwarding are important ideas that are useful at other layers and in other contexts.