6.826 Principles of Computer Systems **Fall Semester, 1992**

Lecture 15: November 4, 1992

Lecturer: Professor Butler Lampson *Scribe: Thomas Lee*[1]

1 Lecture overview

Today's lecture continues the previous lecture's use of the ISO Reference model as a framework for discussing how to think about communication in a distributed system. In particular, today's topic completes the discussion on the **Network** layer. Next week's lectures should address the **Transport** and **Session** layers.

The two architectures considered today are broadcast networks and switch networks. As each architecture is considered, it may be helpful to keep the three variables mentioned in the previous lecture's discussion of networks in mind. Those variables are addressing, arbitration, and availability.

Before proceeding, however, it might be useful to step away from specifics for a moment to add to our general framework for studying communications.

2 Multiplexing

Multiplexing involves the use of a single channel at some layer of abstraction to carry the data for a number of channels in the higher layer. For example, multiple network layer channels may use the same datalink channel. Each message has a header with an address indicating where individual messages are being sent. The header may also be used to differentiate between whether the document following is an ethernet document or a token ring document.

In some instances, an explicit header may prove entirely unnecessary. For example, if ethernet is encapsulated inside a TCP/IP packet, there is no need for "explicit" address headers. In such circumstances, it is only essential that there be an unambiguous way to parse the mutliplexed message.

3 End-to-end Reliability

Given two disks, A and B, connected via a network, the goal is to move several blocks of bits from Disk A to Disk B. One possible way to do this reliably is to verify each step of the transfer separately: reading from Disk A, sending over the network, and writing to Disk B. Such a process is unduly complicated.

[1]These notes make use of two lecture notes from Fall '91: Lecture 14, scribed by Clifford Tse, and Lecture 15, scribed by Uri Blank and Rainer Gawlick.

Instead, we should consider error detection and retry from end-to-end. This technique involves calculating a check- sum that is associated with a set of blocks to be sent. After all of the blocks are sent, Disk A forwards the checksum to Disk B. Side B, reading off of its disk, recalculates the checksum and compares its own calculation to the checksum received from Disk A. If there is any discrepancy between the two calculations, retry the entire send. This technique is referred to as "end-to-end" because the check is based upon data at the two ends.

The cost of this protocol may be calculated as:

$$cost = l + (c)(p)$$

Where l represents the cost of sending once, c represents the cost of a "retry" and p represents the probability of "retry". (Note that this analysis is very similar to the fast-path performance analysis.) Optimally, we would like cp to be very small.

A note of caution: the probability of need to retry, p, is equal to the probability that any one of the n number of blocks being sent contains an error. The successful reception of any one block is an independent trial and the probability of retry is equal to the probability of *one or more errors*. It is *not* the probability that every block fails.

One possible strategy for minimizing cp is to calculate a checksum for each block rather than for the set of all blocks, and require that only the defective blocks be resent. Another strategy is to obviate the need for retransmission by *forward error correction*. That is, code the data such that it can be reconstructed even if it is corrupted during transmission. However, this tolerates only a limited category of errors.

4 Broadcast networks

Broadcast networks are commonly used for LANs. Often, all stations are connected to a common transmission medium in a bus structure. Each packet is broadcast to the network and each station listens to the network for packets addressed to that particular station. Two common types of broadcast networks are the Token Ring and Ethernet, both of which we shall explore in greater detail below.

4.1 General considerations

Addressing in broadcast networks is a relatively simple proposition. Arbitration, on the other hand, is extremely complex—a phenomenon in direct contrast to switched networks, a second type of network which will also be expanded upon below.

Because logically, broadcast networks simply send every message to every node, it is possible to use flat, unrouted addressing. Each node needs a unique address to send, but relies upon each receiver to check the address field of every message to determine whether a particular message was sent to the particular node examining the message.

Despite the relative simplicity of addressing in broadcast networks, the question of *how to address?* is still unanswered. One possibility is to simply preface every message with a short address field. However, shorter address fields exacerbate the problem of insuring that no address is repeated. This problem is compounded by the increasing number of nodes on the network today.

One logical solution is to therefore use large address fields. In particular, broadcast networks commonly use a 48 bit address field, the uid. Every ethernet widget has a unique id. This at least provides on the order of some $10^{1}4$ addresses. (And when they finally do run out, who cares — the company that is making money by assigning addresses will have already made a killing by then).

While addressing is simplified in broadcast networks, arbitration is greatly complicated. This complexity is due to a common trait of broadcast networks — they have only one bus. One bus means that messages get scrambled unless the network guarantees that there is only one sender at a time.

Resolution of the arbitration problem introduces the notion of scheduling. There are two issues to consider when scheduling a serializable resource:

1. explicit mechanism—a central node which allocates service
2. access—insuring that every other node is wired to the central service

Because of the tremendous configuration problems introduced by trying to wire each node independently to a central authority, researchers at IBM et al. introduced token rings.

4.2 Token rings

In a token ring, nodes are connected via a ring interface. The ring is actually a collection of individual point-to-point links that are configured in a circle. A special bit pattern called the *token* is circulated around the ring when the system is at idle. To send a message, a node must wait until the token reaches it. The node removes the token from the ring, sends its message and then appends the token. By keeping ahead of the token, a particular message is sent to every node in the ring.

Implementing the token ring architecture, then, has several principle requirements:

1. A token that must go *everywhere* in the ring.
2.
   - There should be *exactly one* token when the network is idle
   - Zero tokens at idle $\Rightarrow$ no work will ever be done
   - Two tokens at idle $\Rightarrow$ too many tokens
3. The ring must be sequential. Two nodes may not be permitted to grab the token at once.
4. The token should follow rather than precede messages in the ring to preclude the interfence with message delivery.

At least in theory, the system seems acceptable as long as there is only message with a limited amount of data at any one time. But what happens if the one token gets lost? This questions merely highlights the more general question of "*How do you insure that the ring maintains exactly one token at idle?*"

One possibility raised in class was the introduction of a monitoring station which has sole authority to generate tokens and to keep track of the number of tokens in the ring. So what happens when the monitoring station breaks down? For that matter, what happens to the integrity of the ring when any one or more of the nodes breaks down?

Making every node a token generator is one possibility. Preventing every node from generating tokens would required assigning some arbitrary hierarchy among the nodes. Therefore, while the ability to generate tokens is inherent in every node, the authority to do so is possessed by only one node at any one time. While this solution accounts for token maintenance, it admittedly begs the question of ring maintenance. How is a perturbation in the ring registered and how are broken nodes bypassed and reintroduced.

Despite its many difficulties, there are at least two commercial implementations of token rings: IBM's and FDDI (Fiber Distributed Data Interface). FDDI is a fiber optic token ring with a bandwidth of approximately 100 Mbps.

One other problem faced by token rings present at the physical layer (physical with reference to the OSI/seven-layer cake) is the question of clock synchronization. Clocks are useful for generating unique uids; and with particular reference to token ring implementations, clocks are also important for determining when a token has been lost. However, as the number of nodes in a network grows, so does the problem of synchronization of the many clocks.

Because it is precluded by the challenges of clock synchronization, deterministic scheduling is generally not considered a viable means of providing availability. The alternative to deterministic scheduling is probabilistic scheduling. Probabilistic scheduling is the strategy employed by an alternative broadcast architecture to token rings: ethernet.

4.3 Ethernet

Ethernet was originally proposed by researchers at Xerox PARC in the late 70's. Ethernet has a bus structure that uses coaxial cable as its transmission medium. Nodes tap onto the cable. The medium access protocol for Ethernet is called CSMA/CD (Carrier Sense Multiple Access with Collision Detection). CSMA/CD is a probabilistic rather than deterministic scheduling technique that implements *exponential backoff*. Developers turned to probabilistic scheduling because they were searching for a distributed way to schedule and allocate resources that were difficult or expensive to allocate.

The conventional process is:

- Check wire
- If busy, do not send because the wire is busy
- If idle, send
- Monitor the wire for collisions—This is because two or more nodes may register *idle* at the same time, causing multiple messages to be broadcast simultaneously. A node detects collisions by sensing whether the signal on the bus is the one sent by that node or whether it has garbled by something else.

- If collision has occurred, stop and retransmit

It is during the retransmit phase that the exponential component of exponential backoff is introduced. Rather than retransmitting immediately, the sender node picks an exponentially distributed random variable t and waits for time t before sending again. If the maximum time to send a message is x and $t > x$ then $P($completing transmission after waiting for t units of time$)$ is fairly high because t spreads out the overlapping transmissions.

However, simply picking a large t is relatively undesirable because as the time required to send a message increases proportionately. An adaptive strategy is to retry, upon collision, after a random amount of time in the range $0..t$; where t is initially small. With each collision t is doubled. Each node continues to backoff until it succeeds. This has the advantage that it estimates the number of nodes that are trying to send, and adjusts t accordingly. The alternative is to continually backoff to high values of t, in which case, performance is poorer.

One possible application of exponential backoff, to relate to other topics studied in the course, is the spinlocks used to synchronize a multiprocessor. Exponential backoff can be applied to determine how long to wait before attempting to reacquire a lock. If the number of threads is greater than the number of processors, exponential backoff can also suggest how long a blocked thread should wait before releasing its locks because waiting for the processor will take too long.

To compute the efficiency of Ethernet, let k be the number of stations and p be the probability that a transmission will succeed. Then the average time, A, required for one successful transmission (by any station) is given by:

$$A = kp(1-p)^{k-1}$$

which is maximized when $k = 1/p$. As the number of stations in the network increases, i.e. $k \to \infty$:

$$A = \frac{1}{e} \approx 0.34$$

The expected value of trials before succeeding, $E(trials)$, is given by:

$$E(trials) = \sum_{0}^{\infty} j(1-A)^{j-1}A \approx 2.7$$

Caveats to exponential backoff:

- In situations subject to collapse, retry is a bad strategy.
- In situations where there is no central scheduler, backoff is helpful for determining how many other people you are competing with for the shared resource. If everyone is polite, then each node takes only its share. However, as in many situations, this strategy is based on the trust that all participant nodes will cooperate. A selfish node might refuse to backoff in order to improve its individual performance at the expense of all others.

5 Switched networks

Switched networks are composed of links, switches and nodes (often, switches and nodes are the same). Each node must determine the link on which to forward a packet it has received.

Addressing in switched networks, in direct contrast to broadcast networks, is complicated. Because messages do not simply go everywhere, individual switches must know how to interpret addresses (eg: where and how to route a message)

One solution to the addressing question is called *source routing*. This technique has several distinct characteristics:

- Source indicates what to do at each switch
- Routing is independent of the state of the network
- Source needs to know how to route a message from the source to the destination

As an example, consider a network where each node has 4 links. A message could be preceded by a sequence of addresses:

$$A_1 A_2 A_3 ... A_n \text{Message}$$

In this example, the message is routed through n different nodes with the receiving node being the nth node. Each A_i represents a two- bit address identifying the link that the ith node must forward the packet on. Each intermediate node reads the two-bit sequence at the head of the A_i's to determine the next intermediate node, removes the head of the list, and forwards the updated header and message.

This strategy is not perfect, of course:

- The sequence of A_i's is of variable length.
- It requires the source to know the route between it and the destination.
- It depends on the network—if the net changes or any node fails, the entire sequence will be invalidated.