6.826 Principles of Computer Systems	Fall Semester, 1992
Lecture 14: November 2, 1992	
Lecturer: Butler Lampson	*Scribe: Nikhil Gautam*

1 Lecture Overview

This lecture marks the mid-point of the course. The rest of the semester will deal with distributed systems. The lectures of the next couple of weeks will focus on describing what a distributed system is, and what the necessary communications infrastructure is to support such a system. In later lectures, other issues such as fault tolerance in the context of distributed systems will be studied.

The major topics of this lecture are listed below:

- Characteristics of a Distributed System
- ISO Reference Model
- Networks

2 Characteristics of Distributed Systems

One of the questions to ask in studying distributed systems is what there is about a system that makes it distributed. A naive response to this might be to describe it as a system which has nodes connected by wires, such as in Figure 1. The terms "nodes" and "wires" by the way are legacies of the "good old days of computer networking".

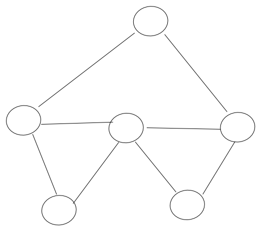

Figure 1: A Distributed System ?

However this is not a complete answer. Many systems can be described this way. For example, an IC could be a called a distributed system if we considered transistors to be the nodes. One might then be tempted to use dimensional criteria, such as size or distance, to describe a distributed system. However, this, too, is not a relevant characteristic of a distributed system.

We now proceed to study different characteristics of distributed systems. We shall see that these aspects are not necessary conditions for a system to be considered distributed. Rather they should be seen as general indications that the system with these properties is likely to be a distributed one.

1. **Communication is Costly**. In a distributed system, communication overheads are generally not assumed to be negligible, and we have to pay attention to them. How much, quantitatively, is considered to be "costly" is, of course, a matter of degree. However, if communication is seldom a matter of concern, then the system is not likely a distributed one. This aspect is not a requirement for distributed systems, but gives a strong hint that the system might be a distributed one.

   One example of a distributed system in which communication overheads have a dominant effect is multiprocessors. A major effort in designing such systems is spent in trying to reduce the effect of communication. Such issues of communication shall be studied in a more quantitative manner later on.

2. **Fault Tolerance**. We expect a distributed system to go on working even if parts of it are broken. As before with communication, this is not an absolute requirement of distributed systems. In fact, fault tolerance is not usually that important an aspect in some multiprocessors which often have simple strategies for dealing with this issue.

   However many other systems do have very strong requirements for fault tolerance. In the case of the Internet, for example, people expect not to be affected by the failure of other machines. The aim of fault tolerance is to get high availibility. This is often achieved by replication. Fault tolerance is a fundamental part of the study of distributed systems.

3. **Autonomy**. This is not a technical issue, but still a relevant characteristic of a distributed system. Often, all the nodes of a distributed system are not under the same management. Thus we are limited in our ability to apply a system-wide policy.

   Once again, it should be noted that this is not an absolute distinguishing factor of distributed systems. For example, the nodes of a multiprocessor are usually *not* autonomous.

Another point to be noted is that a distributed system can be recursively defined. A distributed system consists of nodes, but each of the nodes itself could also be a distributed system. There are other configurations possible. For example, a network of workstations is a distributed system, but each workstation itself is not considered a distributed system. Or, one may have a system in which the nodes are distributed systems, but the overall system is run synchronously, not as a distributed system.

Distributed systems also have some incidental (not so fundamental) characteristics which are described below.

4. **Concurrency**. There are two clear reasons for studying concurrency. First, we need concurrency to make our systems go fast. Second, we often find that the distributed system under study requires fault tolerance and autonomy. In such a situation, the nodes *must* be running concurrently.

Of course, there are always ways of stamping out concurrency, but this is often very difficult to do. Thus concurrency plays a major role in our understanding of distributed systems.

5. **Size**. It is possible for a system to grow large, if one accepts the axioms of distributed systems. The telephone network, and the email distribution system are examples of big systems that go into the tens of millions of nodes. One can often say that big systems are usually distributed.

6. **Heterogeneity**. Many distributed systems have different kinds of nodes, for example the different workstations in a network of workstations. From a purist's point of view, this is not a matter of concern–it should not matter whether one machine is Unix-based and another is DOS-based.

   However, in real and practical terms, we need to define certain specifications to which the different nodes have to adhere. Often it is very difficult to determine whether different nodes meet certain specifications. Some people might want to do away with specifications completely. However the resultant design would not be a system! There has to be some agreement and some interfaces for a design to be called a system.

   Another point of note here is that it is always possible to include a strange node into a distributed system, by simply building an interface adaptor box.

We shall now look a little more at the specifics of distributed systems by taking apart one such system, and looking at each layer of abstraction in it. A good example is the ISO Reference Model (also known as the seven-layer cake !).

3 ISO Reference Model

The ISO Reference Model tells a story about how to think about communication in a distributed system. One must be careful not to take this model too literally. This model defines 7 layers which are mentioned in brief below.

1. **Physical**. From volts to bit sequences.
2. **Data Link**. From bit sequences to point-to-point messages.
3. **Network**. From point-to-point messages to routing of addressed messages.
4. **Transport**. Reliable messages and acks.
5. **Session**. Issues of naming and RPC.
6. **Presentation**. Murky relationship between data format and user.
7. **Application**. Not really relevant to distributed systems.

The first four layers, Physical through Transport, are fairly standard and are typically needed in most systems. The next three layers are described more vaguely, and their existence is a little more controversial.

We shall now describe these layers in a little more detail. The last two layers, Presentation and Application, are murky in their detail and have little relevance to distributed systems, and hence will not be studied in any great detail.

4 Physical

This layer deals with taking a physical phenomenon and turning it into a stream of bits or symbols. The RS232 standard is an example of an implementation of a physical layer.

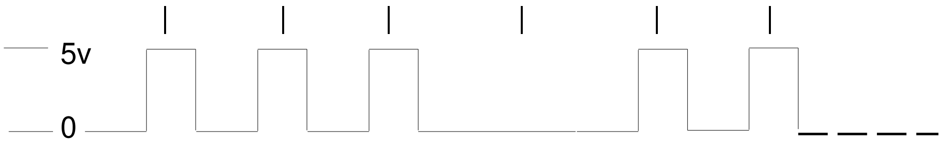

Figure 2: Bit stream

The RS232 standard gives rise-time and fall-time specifications, and also some coding scheme. For example, it defines a clock edge at which we must sample the data.

5 Data Link

This layer imposes some structure on the bits. The RS232 standard also imposes some structure, but it is still at a physical level. One of the functions of the data link layer is to parse the bit stream.

An example of such a structure is shown below. We have a sync pattern, followed by a number n, indicating how many bytes of data there are, followed by the n bytes of data.

```
000n[...n bytes ...]
```

The idea, of course, is to put a recognizable pattern as the sync. However it is possible that the user may need to put the same pattern as part of the data. There are a few techniques to deal with this problem.

One method is to have marks, i.e. have a long idle time. One can then be sure that the bits immediately after an idle time are the synch bits.

Another method is to perform bit-stuffing. This involves transforming the bits in the data to avoid sending the synch pattern in the data. In other words, the sender's data is coded into a bigger space. One glorified example of bit-stuffing is to code $0 = 01$, $1 = 10$ and $synch = 00$. This is a wasteful method, but ensures that the synch pattern can not appear within the data.

This is a 1/2 scheme, in that 2 bits of the resultant data stream have been generated from 1 useful bit. In real life, FDDI and other standards use similar, though less wasteful encoding schemes, like 4/5 and 8/10 schemes.

Yet another method of inserting synchs is to scramble the data by XORing it with a pseudo-random sequence. The synch is inserted in the scrambled datastream after every n bits. The receiver guesses the position of the synch and checks if it repeats after every n bits. Since the data is scrambled, it is unlikely that it will contain a similar periodic pattern that may be confused with the real synch.

This scheme is used in T1 trunk lines. The T1 system uses a single synch bit after 24 bytes of scrambled data. The receiver makes a guess about the synch bit and checks if it repeats after 24 bytes. If not, the next bit position is checked, and so on. The advantage of the scheme is that only 1 bit is wasted per 24 bytes of data.

6 Networks

A network is a distributed system in its own right and is often very complex. This illustrates the recursiveness of everything in distributed systems.

We start by defining the specification of a network. As mentioned earlier, the network layer is responsible for changing point-to-point messages to (address, msg) pairs. For the purposes of Spec, for now, we shall ignore the addressing aspect, and deal only with message handling.

We can think of the network as a store, which may also have several copies of the message, as suggested in Figure 3.

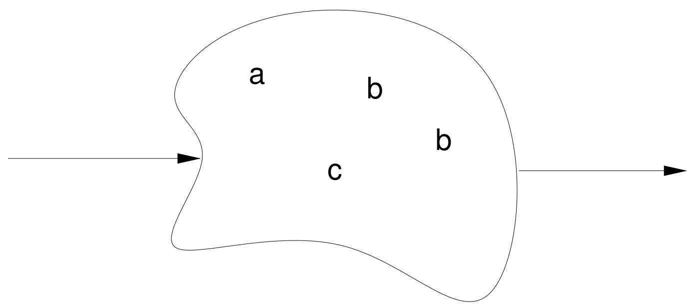

Figure 3: The network as a store

```
TYPE M = D              % D = data

net: Multiset M
```

```
Send(m) =
     net := net ++ m

Rcv() -> M =
     VAR m | m IN net =>
       net := net - - m; RET m


Lose() =
     VAR m | m IN net =>
       net := net - - m

Dup() =
     VAR m | m IN net =>
       net := net ++ m
```

Note that in this specification, the order of messages is not guaranteed. Also, a message may get lost or duplicated. Given this specification, we can explore (i) what it is good for (which we shall do next time) and (ii) how to implement it (which we shall look into later in this lecture).

There are some aspects which are not talked about in the abstraction of a network given earlier. These aspects are listed below:

1. **Addressing**.
2. **Performance**. There are several issues which characterize the performance of a network, which are not mentioned in the specifications, yet which need to be taken into account in the implementations.
   (a) *Congestion*. Congestion is defined as the overload of any shared resource. Some solutions to the problem of congestion are buffering, and throwing away of data. The result of importance here is that as the load increases in a system, the performance deteriorates.
   (b) *Latency*.
   (c) *Bandwidth*. It should be noted that the actual bandwidth depends on the entire path. Thus, one segment in the path may be running at 10Mbps, but because of shared resources on other segments, the total bandwidth may be significantly less.
3. **Failure Probability**.

There are lots of implementations for this specification which range from a simple RS232 cable, to the Internet. In order to study networks more closely, we can classify networks into two basic categories: Broadcast and Switched.

6.1 Broadcast Networks

Figure 4 shows typical arrangements of logically broadcast networks. The basic characteristic of such networks is that every message that is sent appears at all nodes. It is important to note that though the configurations look different in Figure 4, all the networks shown are logically the same.

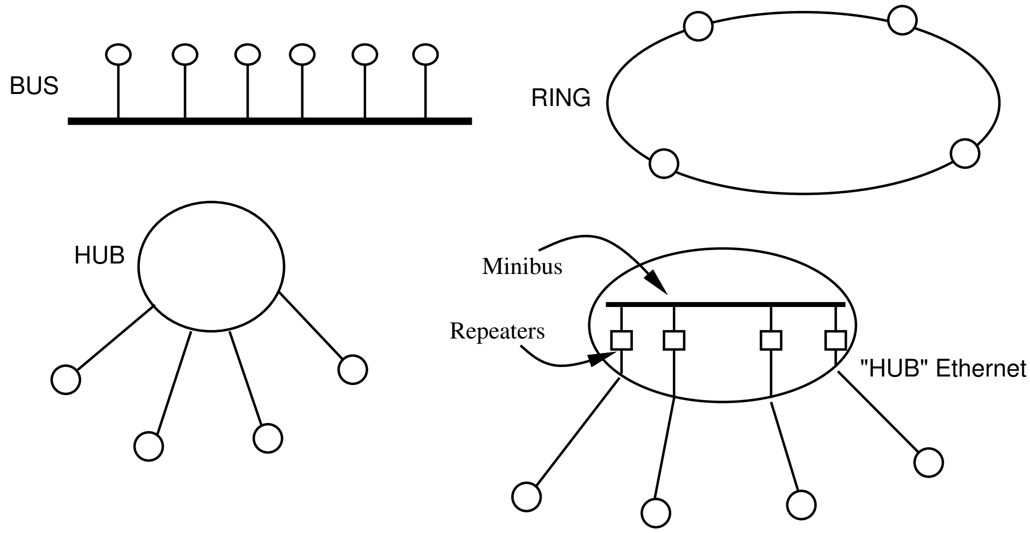

Figure 4: Different Configurations of Broadcast Networks

The issues to look at while studying broadcast networks are the following:

1. **Addressing**. This issue will be dealt with later on, in the next lecture.
2. **Arbitration**. This needs to be performed in broadcast networks because only one node should send at a time. Having arbitration gets rid of congestion. This will be studied more in detail in the next lecture.
3. **Availibility**. The network should be available for reasonable amounts of time. For example, it may be that a network doesn't work if two nodes send at the same time. Now if there is a malfunction that causes this situation to arise all the time, then the network is not available.

   We have to be especially careful in terms of availibility in the case of ring networks. Without certain safeguards, it would not be acceptable for a node in a ring to fail. A ring arrangement can be made feasible if we have relays that bypass a node in case of a failure, as shown in Figure 5. (Note that relays don't work well with optical fibres.)

Typically, the most popular configuration of broadcast networks is the HUB scheme. It is easier to deal with malfunctions in this scheme. But if a ring network is required for some reason, then even the ring can be configured in the form of a HUB to make a more reliable ring. This HUB-like ring configuration is shown in Figure 5.

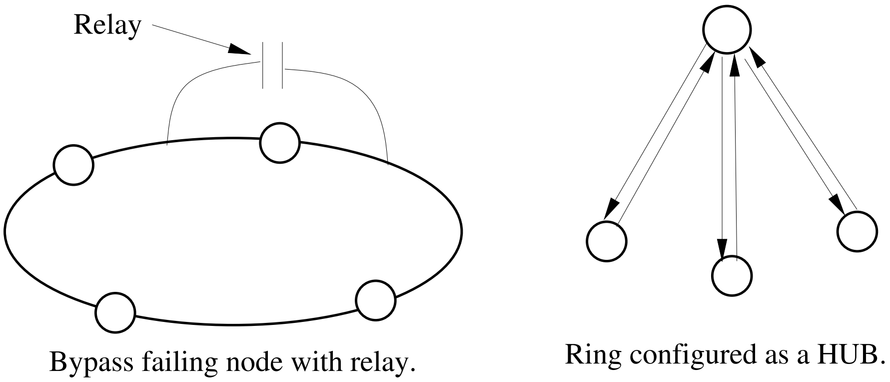

Figure 5: Ring Networks