Lecture 1 notes

Kevin Brewer

November 5, 2002

1 Networks: the basics

With the growth of the internet, networking has become an ever-present aspect of computing. Once relegated to business offices, the military and universities, networking has now entered the home through the World Wide Web and email.

With networking now such an integrated part of the computer, the inner complexities of networking are lost on most users. This course aims to explain the basic concepts involved in networking.

At its heart, the goal of networking is the transfer and sharing of information with other users. Though this may appear to be a simple process, networking draws on such diverse areas as computers, information theory, probability, signal processing, electronics, graph theory, cryptography and physics.

1.1 The core idea

Networks exist to perform one ultimate function: to provide *services* that allow users to send *information* from one host to another host.

Information is sent from one host to another by means of *data transfer*. Data transfer may be the core function of a network, but this transfer is used for a myriad of applications. Email, web browsing, voice communications, video conferencing, and even games all use data transfer to perform their specific tasks. Essentially, any information that can be quantified digitally can (and will) be sent over a network.

Notice before the use of the word service to describe the action of a network. Service defines how information is delivered between the sender and receiver. This service is performed by two key components: *hardware*(also known as the infrastructure) and *software* (also known as the protocols) which will both be discussed in greater detail later. It is important to note that the network does not know what is being delivered between the two hosts, nor do the hosts know how the information is going to be sent. On the internet, there are different types of services: these will be discussed later.

1.2 Example: the postal service

For example, look at the Postal system: the person sending the mail does not need to know how it arrives there, nor does the post office need to know the contents of the envelope. Instead, the two parties agree on an addressing system and the post office takes care of the delivery. Like a computer network, there is no single process to deliver the piece of mail. instead, the mail is processed in hops, with each stage sending it the next step to be handled there. When routers are discussed, it will be clear that a computer network uses the same strategy.

The postal service has the equivalent of hardware and software. Think of the people, machinery and buildings of the postal service to be the hardware. The software, however, lies in the procedures and protocols that the company follows. These procedures allow the post office to properly and efficiently send mail using pre-defined conventions of addressing.

Continuing the postal example, there are different levels of service as well (first class and regular). The mail arrives faster depending on which level of service is asked for. These different levels of service work using the same hardware (i.e. the postal facilities) but have different procedures to deal with the differences in service. The internet has two service types, called connection-oriented and connectionless, which will be discussed later.

Ultimately, however, the postal service serves to show that the post office is a separate entity from those who use it, just like a network. The following part describes the basic hardware components of a network.

1.3 The hardware of a network

Consider figure 1: this represents a simplified view of a network. Notice there are two key components to the network: *switches/routers* and *terminal/host/end systems*.

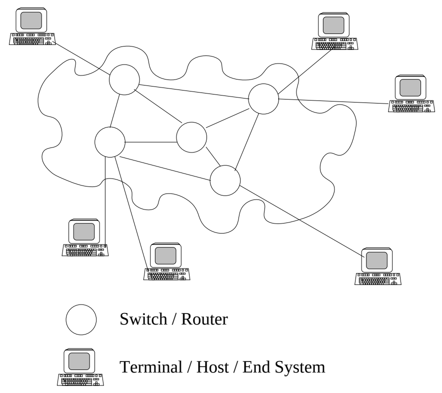

Figure 1: A basic network

Switches (also known as routers) are computers with the sole purpose of handling the data sent in between the hosts. The job of switches is to make sure the information sent from a host arrives intact at the proper destination. Thus, error control, routing (i.e. sending the data to the correct destination) and flow/congestion control are the major functions of routers. Hosts are connected to switches, and switches are connected to each other through links.

The hosts are the endpoints for a given piece of network traffic. The hosts are the sender and the receiver. Generally, the major concern of the hosts is the application being run that requires the data to be transferred. This is where the protocols are more concerned about the end-to-end information, instead of routing and error correction.

The hosts deal with the high-level data, and send it onto the network to be handled by the switches/routers. Note that in small networks, hosts may function as a switch or router as well.

All of these components are connected together with *links*. A link is the connection between each computer on the network. A link can be a phone line, a coaxial cable, fiber-optic cable, copper cable, or even a wireless transmission. These links are the physical connections between the systems that enable the network to function.

The hardware used for a network should be dependent on the needs of the network users, which includes the programs being run on the network, the number of users and the volume of data sent by each user. For instance, a single user generally does not need a dedicated T1 line, and a trans-Atlantic cable better be more than a couple of network cables! Such examples are extreme, but serve to demonstrate that one must consider the scale of the network when deciding upon the hardware.

Concerns about the hardware are not limited to the volume of users. If data needs to be sent error-free, the designer should consider more reliable links to minimize the data that has to be re-sent. Such concerns also affect the choice of routers within the system as well.

In addition to the type of components used, the layout of said components must also be considered. For example, a LAN may be fast, but if the only connection to the internet is through a slow link, then all the traffic will be bottlenecked and very slow. Security concerns are also a factor: if some systems are particularly sensitive, it might be a bad idea to set up the network such that all network traffic comes through those systems, or even to allow traffic on those systems at all.

1.4 Protocols: the software of a network

Once the hardware is in place, software still needs to be present in order allow the network to function properly. When people think of network software, they think of the end-user applications: email programs, web browsers, and other such software. In reality, these programs are just the endpoints; what networks really use are programs which rely on *protocols*. Protocols are the standard methods which a network uses to transmit data between endpoints; such protocols include things like ARQ, TCP/IP and UDP, which will all be discussed later. Section 3 explains protocols in more detail, and Section 2 will explain the difference between two different types of service protocols; connectionless and connection oriented.

1.5 Modularity of networks

Considering how many different applications use the name network (in most cases the internet), it might be initially surprising that a single network can support such a wide variety of programs. This flexibility of design is one of the key features of a network; namely, *the hardware and software of a network is generally separate from the applications running them.*

This modularity of networks versus the applications using them is a key point that will be in greater detail later. Additionally, the network itself is modular, with different tasks being covered by separate pieces of hardware or software. Later on, the concept of the *layered model of networking* will be introduced, which goes into how the various tasks and parts of the network are divided.

1.6 Differences in network requirements

Given the precious statement of networks being independent of their applications, one might mistakenly believe that all applications work the same way over a network. This is not so.

In fact, different applications use the same network, but can rely on different ways of using the network depending on the requirements. For instance, one application may require fast communication, but can handle small errors in the files transferred. Another application may not be time-dependent, but needs the data sent to be correct. The question is: can the same network handle these two different requirements?

The answer is yes. Once again, the modularity of a network becomes handy, since we can use different modules to meet different requirements for the particular application. At the most basic level, these different ways of handling data are called *network services*, which will be discussed in section 2.

1.7 The internet: A network of networks

No single group owns the internet; that is, computers (or groups of computers) that make up the internet are controlled independently of each other. The reason behind this is that the internet is actually a *network of networks* connected together under a mutual agreement to transmit data.

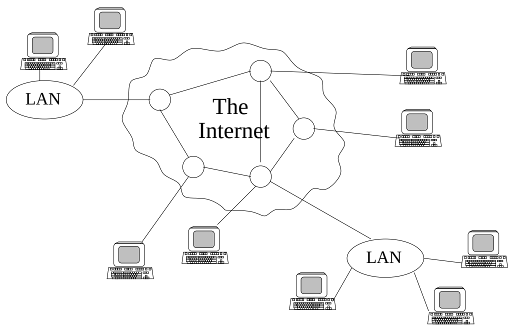

Figure 2: A network of networks

The internet is the ultimate example of how autonomous networks can function together. This is a perfect example where modularity comes in handy; since each system can have its own implementation, there is no problem dealing with individual differences between networks.

2 Network services

When applications send data, a choice must be made about the type of service used. In networking, there are two choices: connection-oriented and connectionless services.

2.1 Connection-oriented

A service is said to be *connection-oriented* if the two systems in communication engage in some initial setting up of a link, known as *handshaking*. This handshaking requires extra time to set up, but results in *more reliable data transfer*.

By a reliable data transfer, it is meant that packets are sent in order and without error. If there are errors, it is reported and the data can be re-sent. Additionally, the two systems decide what rate the data is sent and received to ensure that both sides can handle the communication; this is known as *flow control*. Finally, the two systems engage in *congestion control*, that is, they make sure the packets being sent are not bottlenecked at a single point. These three aspects allow for a reliable transfer of data between the systems.

On the internet, the connection-oriented service used is called *TCP*, which stands for Transmission Control Protocol. Later on in the course TCP will be examined to show how it achieves reliable data transfer.

An example of a connection-oriented service in our daily lives would be depositing money at a bank with a bank teller. In such a case, the teller serves to ensure that our identity is confirmed, the amount deposited is correct, and that the money has in fact been added to our account. This service would take a bit more time than an automated service, but it ensures that everything has gone right.

2.2 Connectionless

In contrast to a connection-oriented service, a connectionless service does not have any handshaking between the two systems. The result is a service that is less reliable, but faster.

Such sending is risky, because there is no allowance for reliable data transfer, flow control or congestion control. As such, the data packets may arrive out of order, may be corrupted, and the receiving system may be overwhelmed by the packets; this possibility is the tradeoff for greater speed.

In terms of computer systems, connectionless systems are most commonly used for services such as streaming video; for such an application the data must be sent as quickly as possible, and lost packets need to be ignored to allow for sufficiently fast transmission. On the internet, the connectionless service used is called *UDP*, or User Datagram Protocol.

The previous part used a bank teller as an example of a connection-oriented service. For a connectionless service, consider the case of a night deposit slot. When items are deposited, the slot has no way of verifying the identity of the depositor, so if the wrong account number is entered, there is no immediate detection of the error.

Additionally, the amount deposited is not counted, so an error there will not be detected. Finally, there is no confirmation of the deposit. In fact, the deposit envelope may be jammed in the slot somewhere and neither side will know it. This is akin to connectionless systems having not confirming connection or managing flow control, so the system may be overwhelmed.

Such a lack of features may make deposit slots seem unattractive to a customer, but they do offer a significant advantage.The advantage of a night deposit box is that the items to be deposited can simply be dumped in the slot, making the process much faster. A teller, on the other hand, would have to go through each of the items to be deposit and confirm them.

Another difference lies in the availability. In order to deposit items with a teller, the branch must be open and a teller must be available. This involves coming at a specific time and possibly waiting in line before being able to be serviced. A night deposit allows the customer to deposit at any time with little likelihood of a lineup. On a network, the same principle applies; a connectionless service need not check for the availability of other system, since it sends without needing to handshake. Though this may be less reliable, it does allow for more efficient service. These differences demonstrate how connectionless services can be faster and more available, but have less tolerance for problems.

3 Protocols

With the network architecture in place, software must be created to allow the sender and the receiver to use the hardware. The software must meet the requirements of the application using the network, and must work with the architecture provided.

Network communication is a complicated procedure; at the top level, you have two communicating devices sending messages between each other over a physical medium, such as a phone line. These messages are are subject to both time delay and loss, and neither system can be completely sure of the status of the other system. How do systems such as this manage to communicate?

The answer: *protocols*.

3.1 Definition:

A protocol is a predefined set of actions in order to achieve a certain goal. In the case of computers, it is a predefined set of signals and actions by two systems in order for them to communicate accurately and as efficiently as possible.

Protocols are not limited to computers, either: everyday life is full of various protocols designed to make routine tasks easier. When we order at a fast-food restaurant, for instance, the procedure is fairly routine regardless of which restaurant we are ordering from.

3.2 Example: SMTP

Now, consider the following internet protocol. SMTP (Simple Mail Transfer Protocol) is a protocol for sending mail to a remote system. Consider the case where the server at *toronto.edu* (represented by C) is sending mail to *sf.com* (represented by S).

```
S: 220 sf.com
  C: HELO toronto.edu
S: 250 Hello toronto.edu, pleased to meet you
  C: MAIL FROM: ¡alice@toronto.edu¿
S: 250 alice@toronto.edu. . . Sender ok
  C: RCPT TO:¡bob@sf.com¿
S: 250 bob@sf.com Recipient ok
  C: DATA
S: 354 Enter Mail, end with "." on a line by itself
  C: How are you?
  C: See you soon.
  C: .
SS 250 Message accepted for delivery
  C: QUIT
S: 221 sf.com closing connection
```

The above listing shows an example of a protocol, beginning to end. Notice that both parties have clear roles and expectations, and the procedure allows the required transaction to take place. In fact, this particular protocol resembles a conversation in a way.

It may seem like protocols were developed for computers, but it should be noted that protocols are used in our daily lives without realizing it. Consider the action of asking for change, ordering at a restaurant, or driving in traffic; each of these actions have protocols which are followed (or hopefully followed in the case of driving) in order to achieve the desired result. In human interaction a protocol means that both parties have an expected action or response to the other in order to reach a certain goal.

For example, when someone asks for the time, it's generally expected that they attract the individual's attention first (with an "excuse me" or equivalent), followed by an acknowledgment by the other person ("yes"). If the acknowledgment occurs, the asker then asks for the time ("do you have the time?") whereupon the other person provides the time, or says they do not know the time. This simple exchange is an example of a protocol that is quite similar to a networking protocol.

3.3 Protocols and networking

Protocols have since the beginning of history. Therefore, it only makes sense that the same principles used in human protocols have been extended to networking. We call these protocols *distributed protocols*, and they will be discussed in the next section, including the special considerations needed to apply a protocol for a networking situation.

4 Distributed protocols

A distributed protocol is a protocol involving two (or more) computers on a network. Such a protocol is designed in order to successfully achieve a goal, generally being some form of data transfer.

4.1 Difficulties with communication

With recent computer systems, communication between systems is relatively simple. As such, it might not be apparent that beneath the applications lies a complicated series of communications between the two computers. The reality is that structured communication between systems where information can be delayed for any length of time or even lost is a complicated process.

4.2 Example: the two-army problem

To illustrate the complexity of communication, consider the *two-army problem.* The example below shows how complicated even a simple goal can be once the paths of communication are uncertain.

4.2.1 The situation

Consider the following situation: two armies, represented by A and B, are at war. Through skillful maneuvering, army A has set itself up so that it is split in half, each half on either side of army B. Army A knows that it can defeat army B, but only if both halves of army A strike army B at the same time.

4.2.2 The goal

The commander of army A is in one half, and decides it would be best to attack at exactly 6 a.m. in order to win. It is possible that the message may not make it to the other half of the army, so a confirmation must be sent to be sure. The commander of A, knowing that both sides must attack at the same time (or else army A will definitely fail) also reasons that both sides must be completely sure that the other side knows of the attack.

4.2.3 The question: can this be done?

Initially, this seems to be a simple problem, but closer examination will reveal otherwise. To recap, army A is cut into halves A_1 and A_2, and both must be sure the other side is attacking. A_1, planning the attack, sends a message to attack, but will not attack itself unless a confirmation message is returned, since it cannot be sure whether or not A_2 has received the message.

Now, consider what happens if A_2 receives the attack message. A_2 knows that A_1 will not attack unless it receives confirmation, so it will try and send a messenger back to A_1.

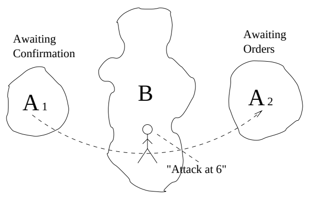

Figure 3: Two army problem, step 1: A_1 sends the message

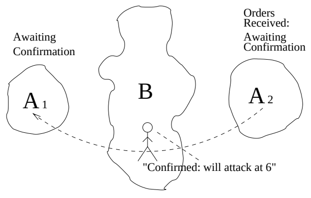
Awaiting Confirmation
Orders Received: Awaiting Confirmation
A1
B
A2
"Confirmed: will attack at 6"

Figure 4: Two army problem, step 2: A_2 confirms the message

Now, the only way for A_2 to be sure that A_1 received the confirmation (and therefore attack) is to have A_1 now send a confirmation of the confirmation back to A_2, and requiring in return a confirmation from A_2...

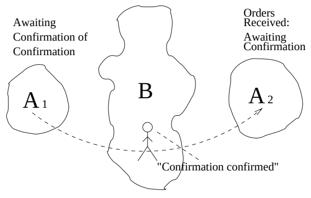
Awaiting Confirmation of Confirmation
Orders Received: Awaiting Confirmation
A1
B
A2
"Confirmation confirmed"

Figure 5: Two army problem, step 3: A_1 has to confirm the confirmation, and things begin to look bad ...

4.2.4 Implications

It should become clear that the above method of planning an attack does not allow for a sure method of confirmation. In fact, it has been proven that the above problem cannot be solved. What makes the above example even more striking is how such a situation can translate to many instances in real networking.

The problem at the heart of this situation is one of *synchronization*, that is, having both parts A_1 and A_2 change states at the same time when the medium between the two does not allow for guaranteed communication. Imagine that the state of attacking is 1 and the state of not attacking is 0. In the two-army problem, the desired result is having both sides enter state 1 at the same time. Since both sides require confirmation that the other side is also in state 1, the two sides will never enter state 1 because the other side will always be waiting for a response while in state 0.

4.3 Networking considerations

On a network, like in the two-army problem, data can be lost. Additionally, data can be corrupted. Additionally, most network applications send out multiple pieces of data; if one piece is delayed, it may arrive out of sequence and the receiver must account for this.

All of these considerations mean that networks must account for such uncertainties when sending and receiving data. These distributed protocols are not only a consideration of the end systems, but of each of the switches along the way.

4.4 Conclusions

The example of the two-army problem should make it clear that communication and uncertainty can make protocols and services tricky. Subsequent lectures will explore some simple protocols for sending data, as well as the advantages and disadvantages of each.