Notes for AMCS/CMCS 662

G. W. Stewart

Nodes and links

Where we are going
Nodes and Links
Passing data
Reliable communication
Ethernet

Where we are going

1. The next three sections of this course will be concerned with communication between computers. We will first consider the how computers may be physically connected to one another. Such connections are called direct links.

2. There is a limit to how many computers we can connect physically to one another, and therefore the problem of passing information between large numbers of computers requires an indirect approach. The solution is a network of computers—a distributed system in which computers cooperate to pass messages between sources and destinations. In the next section we will treat networks — in particular the ubiquitous Internet.

3. It isn't enough simply to get information from one computer to another. It is necessary to have a protocol, with program support, that allows individual processes to communicate. In the third section we will consider the Internet's Transfer Control Protocol (TCP), which, among other things, supports reliable communication between programs running on computers.

4. The three levels we have just discussed sit on top of one another. The layer supporting communication between processes, which is called the *transport layer,* passes its messages to the *network layer,* which in turn depends on the *link layer* to move data between machines.

5. The material in these sections is based on *Computer Networks* by Peterson and Davis (Morgan Kaufman).

Nodes and Links

6. A network can be regarded as a graph whose nodes are machines that can receive and transmit information. Nodes may be loosely divided into hosts and routers. A *host* is generally workstation or PC that is using the network for its own purposes. A *router* is a machine that is actively cooperating in getting messages through the network. Of course, a host may pitch in as a router.

7. Nodes are connected by links that carry information directly between them. There are several types of links.

Cables are useful for building local connections. They come in several types — e.g. twisted pairs (telephone wire) and fiber — and they vary in their bandwidth and the maximum permissible distance between two nodes.

Lines leased from a commercial carrier are used to connect nodes over long distances. They are expensive.

Last-mile links connect a host to a nearby network provider, with the network providing any further connectivity. They include modems running over the phone, digital subscriber lines (DSL), which also run over the phone, but at high speed, and cable modems running over TV cables.

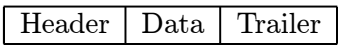
Finally, nodes can be linked without wires. The wireless technology that connects cell phones can be used to connect computers.

8. We will pass over how data is actually transmitted over these media. Generally speaking, all systems use variations in their signals to represent bits. There are several *encoding schemes* for accomplishing this, each having its advantages and disadvantages.

9. Nodes are connected to their links by *adapters* that reside on the I/O bus. They encode and decode information as it passes to and from the link. If the link is a shared resource (like an ethernet), the adapters also implement an access protocol. Adapters are controlled by programs on the machine called *drivers.*

Passing data

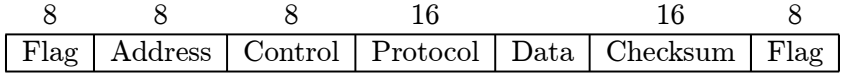
10. It is not enough for a sending node to simply put data on a link. The receiving node has to know, at the very least, where the data begins and ends. For this reason, data is transmitted in *frames,* which are *encapsulated* by a header and possibly a trailer thus:



The term frame is used largely at the link level. At other levels, they are known as *packets* and *datagrams.*

Together the header and trailer serve much the same function as an envelope for a letter. You would not expect to drop several pieces of paper with a letter on it into a mail box and have it delivered. Instead you put them in an envelope, which serves to keep the pages together, and put an address and return address on the envelope.

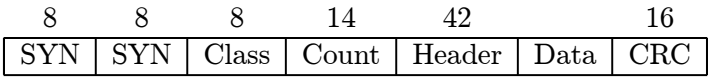
11. There are as many kinds of frames as there are link protocols. Here is one for the PPP protocol, which is frequently used for modem connections.



The numbers above the fields are the number of bits in the field.

The flag is a unique binary number that signals the beginning and end of the frame. The other fields in the header are used for various control functions. The data is a byte stream; that is, the number of bits it contains must be a multiple of eight. The checksum in the trailer is used to detect transmission errors, as we will see below. Because the two flags stand like guards at the beginning and end of the frame, it is called a *sentinel frame.*

12. Another approach to determining the data is to provide a count of the number of bytes in the data. The DDCMP protocol takes this approach.



The first two fields contain the ASCII code SYN, which signals the beginning of the message. The count field gives the length of frame in bytes. With this information the receiver can determine where the frame ends.

13. Both the sentinel and the byte counting approach are not foolproof. If there are errors in transmission, sentinels or counts can be corrupted, so that the frame and even its successor will not be properly received. Even if the frame is received, the data may have been corrupted. One cure is to compute a number from the contents of the frame, and include it in the frame. When the receiver gets the frame, it recomputes the number and compares it with the number that was sent. If they are different, an error has occured. If they are the same, no absolute conclusions can be drawn, since one function value by necessity represents many frames. But if the function is well chosen, we can say with reasonable certainty that the frame is OK.

An example of such an error-checking function is the Internet checksum. It breaks up the the frame into 16 bit words, adds the words in ones complement arithmetic and complements the answer. It is not a very good function; it can fail in some rather obvious ways. A better function is the cyclic redundancy check — the CRC field in the frame of §1.12. A discussion of CRC is beyond the scope of this course.

Reliable communication

14. Errors are a fact of life in data transmission. If the errors are so frequent that every transmission is corrupted, then there is little that can be done. On the other hand, if a reasonable percentage of the transmissions are uncorrupted, we can implement a scheme — called stop-and-wait algorithm or protocol — that turns unreliable communication into reliable communication. Because this algorithm works on direct links in which frames may be lost — usually because they are transmitted with error — but cannot arrive out of order.

Let the sender be X and the receiver be Y. The idea behind the stop and wait algorithm is that when Y receives a packet it sends back an acknowledge-

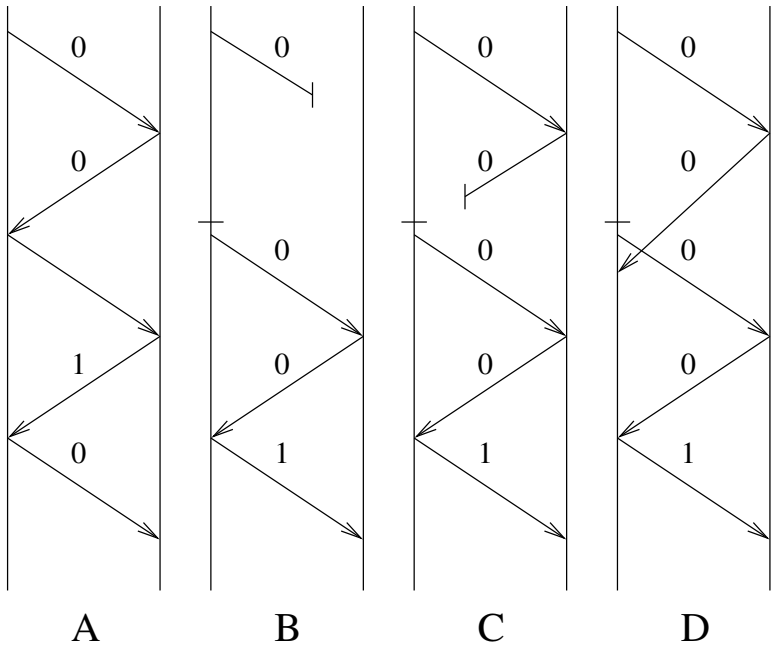


Figure 1.1. *The start-and-wait protocol*

ment (ACK) to X. X does not transmit the next packet until it receives Y's ACK. To insure that the process does not hang up, if X does not receive an ACK after a certain length of time, it assumes something has gone wrong and retransmits the packet.

15. Figure 1.1 illustrates the situations that can arise in the protocol proceeds.

A. This is normal transmission. X sends packet 0. Y gets it and sends back an ACK for 0. Once the ACK is received the process continues with packet 1, and then with the next packet, which is labeled 0 again. These packet numbers must be transmitted with the packets and acknowledgements.

B. X sends packet 0, but it fails to get to Y. After a fixed period of time, indicated by the little horizontal line, X resends the packet. This time the message is received by Y and is ACKs it. The process continues with packet 1.

C. Packet 0 is received by Y, but Y's ACK does not arrive at X. After the timeout X resends packet 0, which arrives at Y and is ACKed, this time successfully. Note that the message numbers are critical here. Without them Y can only assume that the next packet it receives is packet 1.

D. Packet 0 is received by Y, but Y's ACK arrives at X after X has retransmitted packet 0.[1] The recovery from this is essentially the same as case C.

Note that because only one packet is processed at a time, we need only two packet numbers: 0 and 1.

16. The chief difficulty with this scheme is that it may use the link inefficiently. To see what is going on we will use a simplified model of transmission. We will divide the total transmission time T for a packet into two parts. The first is the time d it takes the first bit of the packet to arrive at Y. This is called the *delay* and is a function of the length the line in question. The second is the time p required to process the packet as it flows into Y. This is at the very least the quotient of the packet size and the bandwidth of the link. The larger the packet the larger is p.

Now if the packet is short, then d dominates p. If we had an error free line, we could pipeline one packet after another so that after the pipe is full, packets arrive at a rate controlled by the bandwidth. However, the stop-and-wait protocol requires that we wait for and acknowledgement which is at a minimum $2d$. Thus the effective bandwidth will be less that the bandwidth of the line.

As the packet sizes increases p will ultimately dominate d. Since Y's ACKs are of fixed length — generally short — they will take little more than time d to send back to X. Thus as the packets get longer, the effective bandwidth approaches the bandwidth of the line. However, there are two caveats. First, the larger the packet the greater the probability of a transmission error. Second, many systems put an upper bound on the size of the packets.

17. A cure for the problem is to start pipelining packets, and to retransmit as acknowledgements fail to arrive. This line of thought leads to the *sliding window algorithm,* which is what is actually used in the transfer control protocol in the internet. Unfortunately, it is too elaborate to treat here.

Ethernet

18. Ethernets are the most widely used method of connecting a small number of hosts. The 10-Mbps ethernet consist of up to 500m of coaxial cable. Ethernets may be joined by up to four repeaters to form a net length of up to 2500m.

19. Hosts are attached to the ethernet via adapters. Each adapter has a unique address consisting of 48 bits, which serves as host's ethernet address. Each adapter can see all the messages passed on the ethernet.

[1] Although the figures suggest a uniform time to transmit packets, the stop-and-wait protocol does not assume so. In fact if the transmission medium is a network like the internet, the transmission times can be far from uniform.

20. An ethernet frame has the following form.

64	48	48	16		32
Preamble	D addr	S addr	Type	Data	CRC

The preamble is a bit pattern used for synchronization. Then follows the destination and source addresses. The type is used to indicate what protocol the data is to be passed to after it arrives. The data field is limited to 1,500 bytes, and, for reasons that will become clear later, it must have at least 46 bytes. The data is followed by a 32 bit cyclic redundancy check.

21. Since the net is a shared resource, there must be some way of coordinating the transmissions of the several hosts on the network. The ethernet protocol is based on two facts. First, each adapter can detect whether the net is idle at the point it is attached. Note that this does not insure that some other host not is sending a frame, just that no frame has propagated to the physical location of the adapter. However — and this is the second fact — the adapter can detect when a message it is sending collides with another.

When the adapter is ready to send it follows the following protocol. The protocol makes use of a unit u of time equal to 51.2 microsec, which is the maximum delay required for a bit to pass from one end of the net to the other and back (called the *round trip time.*)

1. The adapter waits until the net is idle and then starts putting its message on the net.
2. If its message collides with another, it stops sending and waits $0u$ or $1u$, chosen at random, before trying again.
3. If the second attempt fails, it waits $0u$, $1u$, $2u$, $3u$, chosen at random, before trying again.
4. If the third attempts fails it waits $0u, \ldots, 7u = (2^3 - 1)u$, chosen at random, before trying again.
5. And so on.

The pattern is obvious. After each failure the number of waiting times that the adapter must choose from doubles. This is called *exponential backoff.* The adapter only does a fixed number, say 10, of these steps before deciding that the message cannot be sent. The effect of the exponential backoff is to spread out the times at which competing adapters try to send their messages. Eventually, the spread is so large that one manages to get its message through. It is significant that in the worst case a message takes about $25u$ to get from host to another, which means that after the fifth attempt the spread is greater than the worst case transmission time.

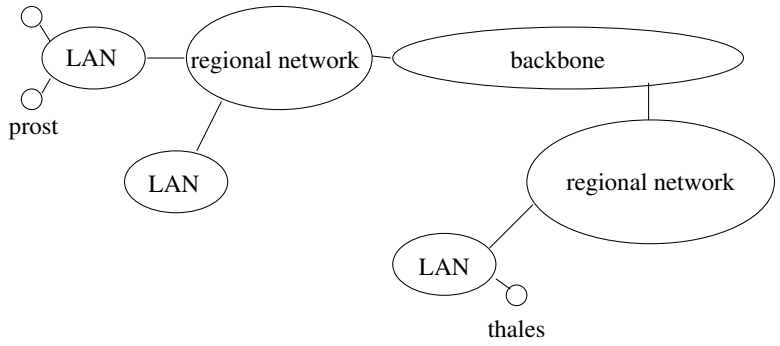


Figure 2.1. *A view of the Internet*

Section 2

Networking

The Internet
The Internet protocol
Address translation

The Internet

1. The Internet is a collection of machines (called *nodes*) that pass information between end nodes, often called *hosts.* Figure 2.1 shows schematically how a message might get from the machine prost to another machine thales, assuming that prost is in San Jose and thales is in Washington DC.[2] The message is passed from prost to a local area network (LAN). From the LAN the message goes to a regional network, and on to a backbone. The backbone is a high capacity network designed to move information swiftly across long distances. The message then works its way back down through a regional network and a LAN to thales.

2. This schema is an oversimplification, but it illustrates one of the most important properties of the Internet: it is hierarchical. In Figure 2.1 the hierarchy extends from processors at the low end through LANS and regional networks in the middle to backbones at the top.

3. An important consequence of this hierarchical organization is that the nodes passing information through the network do not need to know the complete path from one end node to another: all that is needed is an address that uniquely defines the destination. When thales sends a message to prost, its

[2]Actually, prost and thales are just down the hall from each other.

LAN does not need to know the way to San Jose. Instead it only needs to be able to pass the information to a regional network, which in turn passes it to an appropriate backbone.

The process is analogous to sending a snail mail message from DC to San Jose. The person sending the message only needs to know the address. Once the message is in a properly addressed envelope the postal service does the rest. And the postal workers don't need to know much about the system. A sorter in DC only needs to know that a letter is going to California and throw it into the appropriate bin.

Our sorter only knows about bins. He or she does not know how the letter will actually get to California. It could go by plane, train, or pony express, stopping at any number of cities. Analogously, information entering the Internet follows no fixed path to its destination. How it gets there depends on decisions made during the course of transmission. In particular, if one path on the internet is not functioning, information can be routed through other paths. In fact, the remote origins of the Internet was a project by sponsored by the military to build a communications system that could survive a nuclear attack.

4. Thus the Internet is not only hierarchical but it is decentralized. Its these two factors that make it scalable and robust. Its design has allowed it to evolve from the plaything of a small elite to a tool for the citizens of the world.

The Internet protocol

5. The internet protocol is designed to move packets of data — called *datagrams* — from a source host to a destination host. In order to do this, each host must have a unique address. An IP address consists of 32 bits. In writing they are represented in *dotted quad* notation: four decimal numbers separated by periods, each number representing the value of a byte of the address. For example, the address of my machine thales is 128.8.128.81. You can obtain the dotted quad address of a host by attempting to telnet to it. For example,

```
thales: telnet prost.cs.umd.edu
Trying 128.8.128.57...
```

Note that the internet address 128.8.128.57 is distinct from *domain name* prost.cs.umd.edu.

6. Internet addresses are divided into two parts, a network identifier and a host identifier. There is a hierarchy of addresses that depends on the length of the network fields. The following table shows the number of networks, and hosts supported by the three types of addresses.

type	networks	hosts
A	126	16,777,214
B	16,382	65,534
C	2,097,150	254

The idea was have a few very big networks, an intermediate number of medium size networks, and very many small networks. As the Internet evolves, however, these conventions are increasingly shunned in favor of subnetworking and classless addressing.

7. The address space of the current IP (version 4) is rapidly becoming inadequate. A new version of IP (IPv6) has a 128 bit address space. When and even whether a switch will be made to IPv6 is uncertain.

8. IP offers a *best effort* datagram delivery service. This means that is is possible for datagrams to be lost, to be duplicated, or to arrive in the wrong order. At first glance, it is surprising that the Internet could be based on an unreliable protocol. But there are two mitigating factors.

1. In some applications — e.g., the transmission of live music — it doesn't matter if an occasional packet is lost.
2. It is possible to build a reliable delivery system on top of an unreliable one. On the Internet, this is done at the transport level by TCP.

9. An IPv4 datagram consists of two parts — a header and the data. Here is a brief description of the header fields.

Version (4b) The version of IP (in this case version 4)

HLen (4b) The length of the header in 32-bit words.

Length (16b) Length of datagram (header included) in bytes. No IPv4 datagram can be longer than 64K bytes.

Ident (16b) Used in fragmentation and reassembly.

Flags (4b) Used in fragmentation and reassembly.

Offset (12b) Used in fragmentation and reassembly.

TTL (8b) A field (originally 'time to live') that is used to keep datagrams from circulating indefinitely. Each router increments it by one. When it reaches a default value (currently 64) it is abandoned.

Protocol (6b) The higher level protocol under which will demultiplex the datagram (e.g., TCP).

Checksum (16b) A checksum for the header.

SourceAddr (32b) The IP address of the sending host.

DestinationAddr (32b) The IP address of the receiving host.

Options and Pading Contains special options.

10. The HLen field is necessary, because IPv4 allows certain options to be specified in an appendage to the fixed part of the header.

11. Since the Internet is designed to support heterogeneity, it must allow for links between nodes having different properties. In particular, a particular link may have a frame size that is less than the maximum length than the maximum size of a datagram. In this case, the datagram must be broken up into smaller datagrams that can fit on the link. Such a process is known as *fragmentation.* The process of reconstituting the datagram at the destination host is known as *reassembly.* The Ident, Flags, and Offset fields of the datagram head support this process.

Address translation

12. When a datagram has been routed to its final physical network (say an ethernet), there remains the problem of getting the datagram to the destination host. Unfortunately, the datagram address does not correspond to the ethernet address of the host: we must translate from one the other. The Address Resolution Protocol (ARP) is a method of doing this.

13. The idea is that the router connecting to the ethernet broadcasts the internet address to all the hosts on the ethernet. Each host compares the internet address with its own. If they are different, the host does nothing. If they are the same, the host returns its ethernet address to the router, who can then forward the datagram.

14. Since a delivery of a datagram is likely to be followed by more to the same host, the connecting router maintains a table of recent address correspondences. Because addresses may change, the router removes the entries after a prespecified period of time.

The Transport Layer

TCP/IP
UDP
TCP

TCP/IP

1. At the network level communication is from host to host. Applications, however, must communicate with one another — process-to-process. Communication at this level is said to be in the *transport layer.* The two most common transport protocols are the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP). They both use IP to move data data between hosts. Collectively (and a little illogically) they are called TCP/IP.

2. In order to move data between processes it is necessary to specify the processes. It is not possible to use process identifiers assigned used by individual operating systems, since their formats vary from system to system. TCP/IP solves this problem by assigning a *port* to each process. Since the concept of port is particular to TCP/IP, it is independent of the operating system conventions on individual hosts.

3. TCP/IP is a *client-server* protocol. The client process on one host initiates contact with a server process on another host. For example, when you type

```
ftp thales.cs.umd.edu
```

the client ftp processes obtains a port and uses it to contact the ftp server on thales. It is able to do this because there is a *well-known port* (in this case port 21) that the ftp server always uses.[3] Once the client has given its port number to the server a two way channel has been set up. (Actually, the situation is a more complicated. Since the ftp server has to monitor port 21 for other clients, it uses other ports to use to communicate with individual clients.)

UDP

4. UDP is little more than a wrapper that allows datagrams to be delivered between processes. Specifically, the sending process encapsulates the data with a header that contains

1. the source port (16b),
2. the destination port (16b),

[3] For a listing of ports look in `/etc/services`.

3. a checksum (16b),
4. length in bytes (16b).

The packet is further encapsulated with an IP header and sent to the destination host. Of course, packets from other processes may be multiplexed. When a packet arrives at the destination host, it is passed to UDP for demultiplexing and delivered to the appropriate process.

5. Because UDP is just a wrapper for IP, it shares IP's best-effort service. Datagrams may fail to arrive or arrive in the wrong order.

TCP

6. The Transfer Control Protocol is the workhorse of the Internet. As a part of the transport level, it is a reliable way of streaming bytes from one process to another. But it also implements both flow control between processes and congestion control on the Internet. Here, in brief, are some of its properties.

7. TCP provides a reliable transmission service. Data is guaranteed to arrive uncorrupted and in the correct order. Of course, this is too good to be true without qualification. Arrival may be slow, and if the Internet is too badly jammed nothing at all may arrive. The data is error free only in to the extent that the error checks at the lower levels are valid.

8. TCP is a byte stream protocol. To see what this means, let's examine what happens in a typical session between process X on host A and process Y on host B. For definiteness we will focus on the transmission from X to Y, but the same considerations apply in the reverse direction.

1. X and Y establish a TCP connection.
2. X begins sending data. Of necessity, X must send data in discrete blocks, which often have significance for the application A is implementing (e.g., in may be a string or a group of floating-point numbers).
3. As each block is sent it is buffered in a TCP buffer in A. At this point the identity of the blocks from X is lost. All that is in X's buffers is a sequence of bytes.
4. At appropriate intervals TCP assembles the contents of X's buffer into a TCP *segment* and transmits it via IP to host B. The sending and receiving processes in any datagram are unique; but since there may be more than one connection between the hosts, segments may be multiplexed.
5. As segments arrive at B, IP passes them to TCP which demultiplexes and buffers them. At this point the identity of the segments is lost, and we are back to a sequence of bytes.

6. Y requests data. TCP honors the request by returning some bytes from the Y's buffer along with a count of the number of bytes delivered.
7. When they are through, X and Y close the TCP connection.

From the above it is seen that there is that the only relation between the the data that X sends and that Y receives is the sequence of bytes are the same for both. If, for example, A is sending a sequence of 8 byte floating point numbers, a request for data from B may result in the reception of the last three bytes of one number, the next seven full numbers, and first five bytes of the next number. It is B's responsibility to extract the actual words from this stream of bytes. This is something an art, which we will not treat here.

9. Since IP provides only a best effort service, TCP implements a sliding window algorithm to insure that segments really get to their destination. The algorithm includes flow control. Briefly (and superficially), if segments arrive at B too quickly, B reduces the size of the window, which causes A to reduce its rate of transmission.

10. TCP also helps control *congestion* on the network. Congestion control must be distinguished from flow control. The latter prevents an eager sender from overwhelming a receiver. Congestion, on the other hand, occurs when there are too many datagrams for the network to handle, so that routers are forced to drop datagrams. Without congestion control, the individual processes would keep retransmitting as fast as they can, thus making the congestion worse. TCP provides algorithms to detect congestion and slow the transmission rate accordingly.

11. A process or application does not implement the TCP protocol for itself. Instead it communicates using an application programming interface (API). The most widely used API for TCP/IP is the socket interface. An excellent introduction to socket programming is *The Pocket Guide to TCP/IP Sockets* by Donahoo and Calvert.