

CS123 Computer Communications: Lecture 1, Overview and Layering

In this first lecture, I hope to provide a very quick overview of all the networking lectures we are going to study in this class. I do so because when I study things I find it much easier to first see the big picture sketched in outline before I proceed to understand the detailed drawings of each of the figures in the picture. This is particularly important because I will use a bottom-up traversal of networking topics. In particular, I will start with the most unfamiliar topics (to CSE majors): sending bits over a wire using energy. When you are struggling to grasp the arcane details of how noise affects the maximum bit rate of a modem (Shannon limit), it may help you to realize that this is part of sending a group of bits over a single hop, which is part of sending a group of bits between two ends of a network, which in turn is used to send arbitrary data across the Internet using say the web.

Thus in this lecture I will use a very quick broad-brush, *top-down* traversal of networking functions (corresponding to the first two lectures) followed by a more detailed *bottom-up* traversal in the remaining 18 or so lectures. While I hope this overview will provide you with enough material to have some idea about how say email or a web page moves across the Internet, I also hope to convince you that there are many hard and interesting issues that I have glossed over. The study of these remaining issues will occupy our attention for the remaining 18 lectures.

1 Increased Productivity via Layering

The best way to understand the structure of networks is to see it like an onion composed of several *layers* or onion skins, and to understand briefly what each layer does. Before we do so, it may help to understand that layering is just a manifestation of a very universal idea: *division of labor*. So please be patient with this brief digression into philosophy.

In the business world, we hear the buzzword *outsourcing* all the time. Business pundits urge companies to concentrate on their core competencies and to outsource other functions to others who can provide this function more efficiently. Similarly, any study of history shows that civilization as we know it only took place when society invented the concept of *division of labor*. While Paleolithic man spent all his time on hunting and Neolithic man on growing food, today only a small percent of the U.S. is actively engaged in growing food. This frees up the rest of us to engage in various specialities (e.g., make movies, teach classes, attend classes, design wireless chips) that allows society to be more productive as a whole.

Computer systems have taken the outsourcing and division of labour mantras quite seriously. Even if the same group of people have to solve a big problem, computer scientists like to break up a big problem into manageable pieces, using what they call *divide and conquer*. They then solve each smaller problem in turn and combine the results. Thus to sort a large array, we break up the array into 2 equal parts, recursively sort the parts, and then merge the two sorted subarrays. Similarly, in your home PC, you may run Quicken's Turbo Tax, which runs on top of Microsoft

Windows, which runs on top of a Dell PC, which in turn consists of a board that has chips made by Intel and others.

The PC illustrates a particularly interesting form of division of labor that we call *layering*. In layering, a hierarchy of software and hardware layers interpose between the raw hardware and the final user, with each layer *improving upon the services of the next layer below* to offer a superior service to the next layer up. For example, the Windows Operating Systems abstracts the details of the Intel processor, the disk and display features etc., by a series of services presented in a so-called Application Programmer Interface (API). This enormously increases the productivity of the Application Programmer writing Turbo Tax because the Windows programming interface is much simpler to deal with than the details of the raw hardware. However, the end user deals with a point-and click interface that is even simpler than the Windows API.

Thus each layer “improves the services” of the layer below by offering a simpler and more functional interface to the layer above. Simple interfaces allow increased productivity not just for the end-user but for each group of people that work on implementing a layer. For example, some people feel that its easier to develop application programs for Unix than for Windows because of the simplicity and orthogonality of the UNIX API.

In summary, the PC takes complicated chips with esoteric interfaces such as “MOV R, P5” (e.g., Pentium instruction) that only a nerd like you and I can follow, and reduces them eventually (using a series of layers) to a simple point-and-click interface to Turbo-Tax (that anyone can master in minutes). All computer systems such as operating systems and databases attempt to pull off the same magic. In this class we will study how the various network layers take a collection of physical channels that can send energy (that only nerd like us can understand) and put them together to make a Web interface (that anyone can understand).

2 From Hats to Networking

Just to show you that there is nothing particularly clever or innovative about the way networking is structured, let me start with a hypothetical structure for how hats may be shipped between countries in the real world. I will then show that the hat transfer structure has a direct one-to-one analogy with the way networks are structured. The analogy introduces a new idea that you may not have seen in PC or OS layering — *distributed* layers with communicating peer layer entities. The analogy also suggest other areas of interest and potential problems that occur in real life and may carry over to networks. Thus in Section 2.1, I describe the Hazel’s Hats structure. I move on to describe the corresponding Internet structure in Section 2.2. I end this section by doing a more careful analysis of the similarities and differences between hat and data transfer in Section 2.3.

2.1 Transferring Hats using the Postal System

Figure 1 shows the layered structure of a hat transfer application that wishes to transfer hats, on demand, between the Hazel’s Hats store in Boston and the Hazel’s Hats factory in Morocco, Africa. The two Hazel’s Hats entities are shown at the top of the picture. The basic problem to be solved is to transfer hats from the factory in Africa to the store in Boston, whenever requested by the

HAT TRANSFER ANALOGY

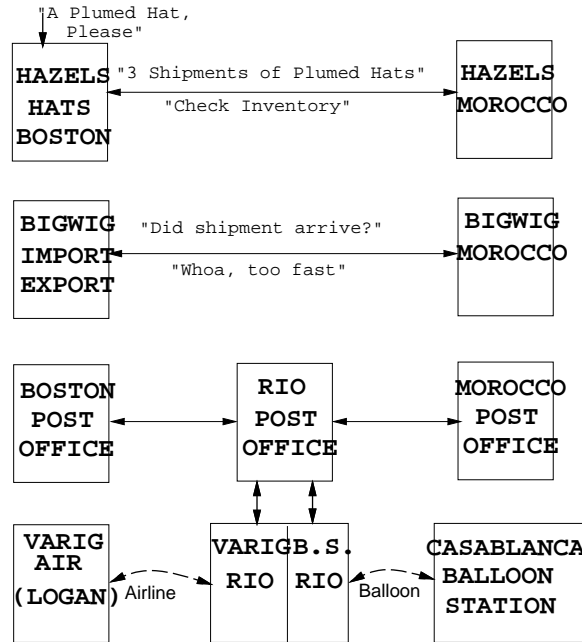


Figure 1: Transferring Hats from Morocco to Boston via a hierarchy of subcontracts

store in Boston. How should this problem be solved?

Concentrating on core competencies (selling and manufacturing hats), Hazel’s decides to out-source the actual transfer process to an import-export agency called Bigwig.¹ Bigwig imports and exports things for various companies. Bigwig is an expert in packaging and breaking up large consignments into smaller packages that can be shipped easily; for example 100 hats may be packed into 50 hatboxes. Bigwig also keeps track of package deliveries, ensuring that any lost or missing packages get tracked down and possibly resent. Bigwig also understands the formalities of moving between countries such as customs.

Again concentrating on core competencies, Bigwig feels no compulsion to physically ship the packages itself. After all, there are other agencies such as the global mail system, UPS, FedEx etc. After some inspection of alternatives, Bigwig chooses the postal mail system. Please note that although Bigwig does not physically ship packages, Bigwig does “add value” in terms of reliability, packaging, dealing with customs etc. If a layer does not add value in the networking or business world, we might as well dispense with that layer.

The interface to the post office is a simple one. Any hat package that needs to be sent from Bigwig’s in Morocco is labeled with the address of Bigwig’s in Boston, and the necessary postage is paid. The biggest “value-add” of the postal service is to compute a route from Morocco to Boston, and to update this route in case of failures — say, a bomb threat closing some airports. Once

¹In case you wondered, the names are from a wonderful book about a rabbit society responding to challenges called *Watership Down* by Richard Adams

the route is chosen, the package is forwarded from post-office to post-office on the route. Thus in Figure 1, the package is forwarded from Morocco to Rio De Janerio in Brazil, and from there to Boston. This indirect route may be chosen, for example, because there is no direct route from Morocco to Boston.

However, to forward a package from say Morocco to Boston, the Post Office needs an actual physical carrier that can bridge the distance between the two countries. Rather than provide this service itself, it can subcontract to other carriers. For example, in Figure 1, the Post Office has chosen to subcontract first to (say) Bob's Balloon service from Morocco to Rio, and then to (say) Delta Airlines from Rio to Boston. Finally, the carriers on each hop use actual physical technology (i.e., balloons, planes) to actually carry the packages as scheduled by the carrier. The choice of technology used at each hop can depend on the characteristics of the technology: for example, balloons may be convenient because they allow access to remote areas without a landing strip.

Back to layering, the layers in the Hazel's Hats structure are Hats, Import-Export, Post Office, Carrier Logistics, and Carrier Technology. One interesting aspect of the layers in Hazel's Hats (which is not present in say the layers of a PC) is that the layers are *distributed*. The people implementing each layer are distributed at different locations and must communicate with each other to synchronize the transfers. For example, the Hazel's Hats in Boston may need to send a request to the factory in Morocco for "One Dozen Plumed Hats". The two Bigwig locations have to communicate to talk about inventory problems and to acknowledge the receipt of packages that have been successfully shipped.

2.2 Transferring Email using the Internet

Figure 2 shows a corresponding picture for how email is transferred across the Internet from say a PC in Boston to a PC in Paris. The corresponding Internet fragment connecting Boston and Paris is shown in Figure 3. First, corresponding to the Hazel's Hat concern distributed in Boston and Morocco, we have a corresponding email program ² that has a piece of client software in the Boston PC and some server software in the Paris PC. The basic problem to be solved is to transfer an email message (saying say "Hi Marcel") from say Boston to Paris. How should this problem be solved?

Concentrating on core competencies (understanding email protocols and features, allowing mail forwarding, aliases etc.), the email protocol decides to outsource the actual transfer process to a *transport protocol*. The most common one used in the Internet is the so-called Transmission Control Protocol (TCP). TCP ships data reliably between computers on the Internet on behalf of several application protocols (e.g., web, email, FTP). Like Bigwig, TCP must often break up large groups of bits (e.g., a huge email attachment) into smaller segments that can be shipped through the Internet. This is because the default maximum segment size or MSS used in the Internet is quite small (used to be 512 bytes but is now 1500 bytes), much smaller than some application messages. TCP also keeps track of segments that have been delivered, retransmitting missing segments if necessary.

Just as Bigwig relied on the Post Office, TCP does not send out segments itself but relies on

²in general in place of email, we will have a networking *application* such as FTP or the web

MAIL: POE TO PROUST

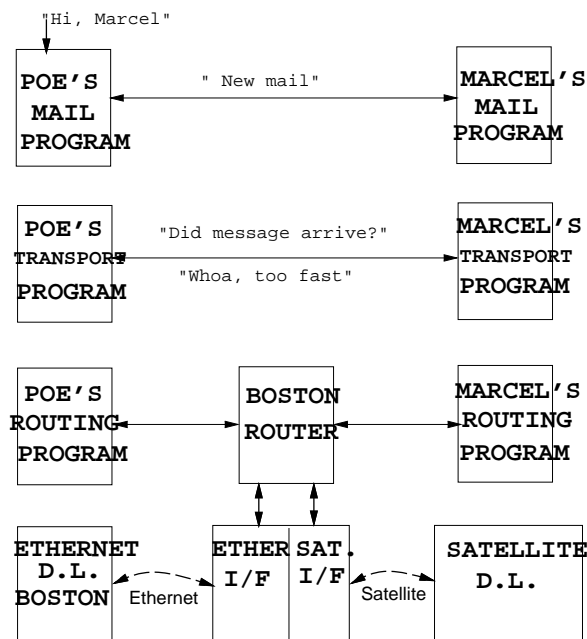


Figure 2: Transferring Email using the Internet via a hierarchy of networking layers. Compare this picture carefully with Figure 1.

the packet-switching abilities of the Internet Protocol (IP). While there are other packet switching protocols such as ATM, Appletalk, Novell Netware etc., IP is by far the dominant protocol.

The interface to IP is a simple one. Any segment that needs to be sent from the TCP in Boston is labeled with the destination IP address (a 32-bit number instead of a street address as in the post office case) of the destination workstation in Paris. Again as in the case of the Post Office, the biggest “value-add” of IP is to compute a route from Boston to Paris (and to update that route in case of failures — say, a failure of the satellite link from Boston to Paris). Once the route is chosen, the packet³ is forwarded from router to router on the route. At this level of approximation, a router is physically a computer with attachments to multiple links that behaves like an automated post-office. Thus in Figure 2, the packet is forwarded to a local router in Boston, and from there directly to the final workstation in Paris perhaps because that is the shortest route.

However, to forward a packet from say the Boston workstation to the local router in Boston, the Post Office needs an actual physical channel that can bridge the distance. Rather than provide this service itself, it can subcontract to other *Data Links* that are analogous to the local carriers in Figure 1. For example, in Figure 2, the local Boston network has chosen to use an Ethernet local area network to link local workstations to a local Boston router. Also, the local domain in Boston has chosen to use a transatlantic Internet Service Provider (ISP) that uses a satellite link from Boston to Paris. The choice of technology used at each hop depends on the characteristics

³a segment together with an IP header is called a packet; do not confuse the word “packet” with our earlier analogy “package”

MAIL FROM BOSTON TO PARIS

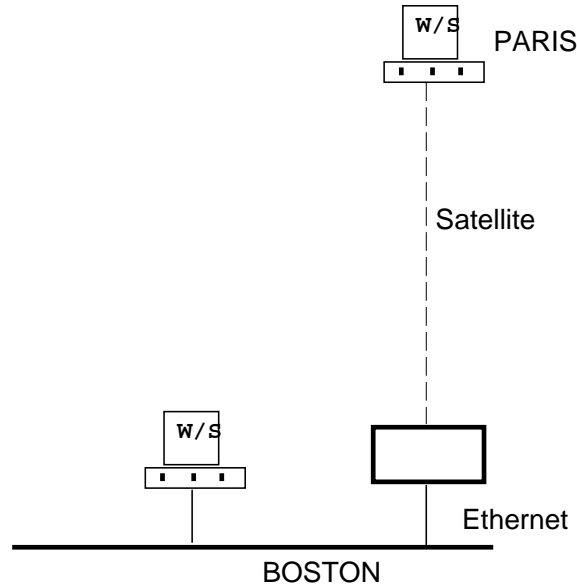


Figure 3: Internet fragment linking Boston and Paris to send email. The box called a router can be considered to be an automated post-office that routes packages of bits that we call packets.

of the technology: for example, satellite is convenient because it allows access without requiring a wired infrastructure, while Ethernet wires are cheap for small distances.

Back to layering, the layers in the Internet structure are Application Program, Transport (e.g., TCP), Routing (e.g., IP), Data Link (e.g., Ethernet), and Physical (e.g., thin Wire Ethernet or RF satellite channels). As in Hazel's Hats, the layers are distributed. The software (or hardware) implementing each layer are distributed at different locations and must communicate with each other to synchronize the transfers. For example, the TCP programs in Paris and Boston may have to synchronize so that segments are not dropped because of lack of buffers (called flow control). Also, the routing layer software in the PCs at Boston and Paris and the router in Paris need to communicate to calculate routes.

2.3 Hats versus Email: A Closer Comparison

Section 1 is a reasonable metaphor for understanding Internet transfer as described in Section 2.2. However, any analogy, metaphor, or parable is only *partial*. For example, when Jesus tells his followers to be like doves in the New Testament, he is clearly telling them to be as gentle as doves and not to grow wings. So what are some interesting differences in the Internet not captured by the Hat Transfer analogy?

First, in Figure 1, we said that Bigwig may handle other issues such as customs. While we don't have tariffs on Internet exchanges (yet), we do have security issues such as authentication to make sure the right people access the right services. Also, just as an import-export agency seals a package to ensure no tampering or peeking, computers may choose to encrypt or cryptographically seal their packets. TCP today does not do this; such services are often provided in an interposing security layer between the application and TCP. Another interesting difference at the transport layer is as follows. It is really easy for TCP to deal with losses by retransmitting a stored copy of the segment. It's much harder for Bigwig to replicate a real package (containing say cookies!) without asking the user to do so.

Second, in the post-office in Figure 1, the interface required letters to be addressed and postage paid. Today, *accounting* in the Internet is more like paying a fixed rate (e.g., \$22 per month to AOL) for sending an unlimited amount of postal mail. While this seems unthinkable in regular mail, it has sort of worked because of plentiful bandwidth in the Internet. However, many pundits feel that some form of *usage based* accounting (as in electricity, postal mail) is inevitable. What form it will take is anybody's guess. It *could* be in the form of electronic cash payments similar to stamps, but it could also be simply measuring user traffic and sending a monthly bill, as in the telephone system.

Third, the function of the Data Link layer in Figure 2 is mainly to group a set of bits into a unit called a frame while the physical channel sends only bits (or symbols) at a time; the received bits must be recovered into frames by the receiving Data Link. The sending data link also attaches checksums to detect errors, throwing out erroneous frames. If this were necessary in the postal network of Figure 2, every package would have to be chopped into slices, and each plane or balloon would only be able to deliver one slice at a time! Fortunately, that is not necessary in the world of transportation and fairly large packages can be sent "in parallel". However, communication links are essentially serial to reduce the cost of wiring, which then causes a framing problem to slice up packets into bits and then to recover the frame boundaries.

Although, the basic function of framing and error detection has no correspondence in say an airline carrier, there is another aspect of multiaccess Data Links like Ethernets that is exactly the "value-add" of say Delta Airlines: this is *scheduling*. Just as carriers must schedule their physical transportation devices to maximize the number of passengers and cargo carried, so also local area network protocols like Ethernet must schedule the transmissions of multiple computers on the same physical wire to maximize throughput (bits carried per second). In networking, we call this *access control* or *media access control (MAC)*.

Finally, the whole issue of layers communicating is handled in a unique and interesting way in networks. When Bigwig in Boston wants to communicate with Bigwig in Morocco it may choose to make a telephone call, relying on *out-of-band* communication — i.e., communication outside the normal network in which data is carried. However, in the Internet all control and other information between layers is carried *in-band* — within the network itself.⁴

For example, when a sending TCP wants to send some information about a segment (for instance, saying this is the 100th segment in the email attachment), it piggybacks the information in a special "header" attached to the data. Thus every data packet sent in a network carries some

⁴The telephone network uses an out-of-band signaling channel to send telephone digits etc. to set up a call.

reserved space that each layer can write into, and that can be read by the corresponding peer layer entity in other nodes. Of course, sometimes a full blown control message must be sent, for example, an acknowledgement for a TCP segment; however, even a TCP ack carries header information for routing and Data Link, and the TCP ack is sent through the network in exactly the same way as a data message.⁵

The two advantages of piggybacking headers on data messages are *economy* and *easier synchronization*. Clearly, it is cheaper to use the network itself to send the layer control information rather than build a separate network. However, an alternative used by the telephone company, is to logically divide its physical network into two parts, one for control and one for data. Even then, the cost of adding a few more bits for headers is also generally cheaper than sending separate messages.

Even if were not cheaper to add headers, clearly adding headers to the same message helps in synchronizing (or coordinating) things. For example, adding a sequence number 100 to the TCP header of a segment ensures that we know that this segment is segment 100; sending 100 in a separate control message would require us to then make a correspondence between the control message and data message, which would be a real pain. Similarly, destination addresses for segments are best sent as headers attached to the given message. Without headers, one can only send data as a continuous stream of bits or energy (as in the telephone network) with a corresponding loss in flexibility.

By the way, a picture like Figure 2 explains why a set of protocols is called a *stack*; it looks like the box representing each layer implementation is stacked on top of the box representing the next layer below. People will often refer to say the *TCP/IP stack*.

3 OSI Terminology: Layering, Protocols, and Interfaces

In computer systems we need to be more precise in our definitions than in say the business world because we are dealing with computers and not with humans. We will use the definitions and concepts provided by the OSI reference model, a model propagated by the International Standards Organization (ISO). While ISO also developed a set of competing protocols for networks, the so-called OSI standards, these have widely been eclipsed by the TCP/IP protocols. However, the ISO *model* and terminology are useful for describing TCP/IP, and even other protocols such as IBM's SNA or Novell's Netware.

The ISO/OSI model is shown in Figure 4. When compared to Figure 2 which represents the structure of TCP/IP, we have a similar set of layers, except that we have two additional ones, session and presentation. It turns out that in practice these layers are really non-existent so we can safely ignore them. However, they were attempts to factor out common services other than transport and routing that applications may require. For example, the presentation layer was supposed to deal with format conversions between different machine formats, say between EBSDIC and ASCII. The session layer was supposed to provide higher layer synchronization functions sometimes used

⁵It is an interesting fact that roughly 50% of Internet messages are 40 bytes or less; these small message mostly correspond to TCP acks.

in databases such as checkpointing large conversations in case of failures. Having said this little, we will ignore them for the rest of this course.

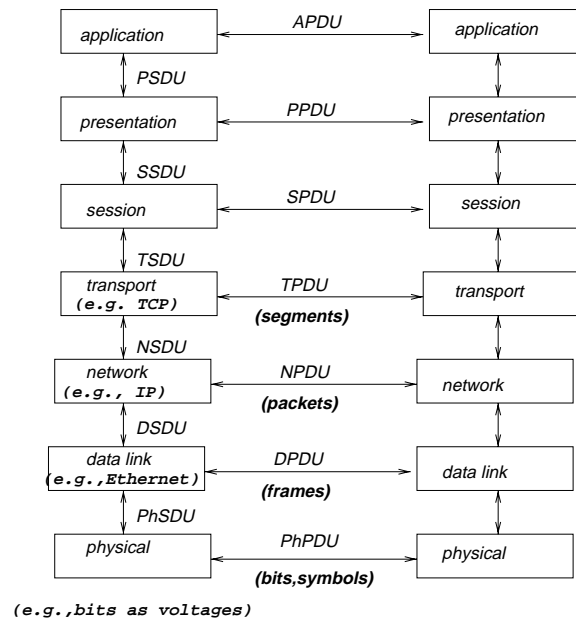


Figure 4: The seven layer OSI model contrasts with the 5 layer TCP/IP model because of two rarely used layers, presentation and session, which we will ignore in what follows. However, note the concepts of SDUs, PDUs, interfaces, and protocols.

The most lasting contribution of the OSI model is its terminology and concepts. Specially noteworthy are the following concepts:

- *Layer Numbering:* Each horizontal slice (layer) is given a number starting with 1 for physical, 2 for data link, 3 for network, and 4 for Transport. For example, devices that interconnect networks looking only at Layer N headers are called Layer N relays. Thus a router, which looks at Layer 3 headers, is sometimes called a Layer 3 relay or Layer 3 device.
- *Protocol:* An important viewpoint is that layer entities can be thought of as *logically communicating* with peer layer entities horizontally as if they had direct links between them (when in fact they do not). For example, the transport protocol at one machine communicates with the transport protocol in a remote machine. The rules governing *horizontal* communication between peer layer entities are called a protocol. For example, a telephone calling protocol is as follows: the calling party says Hi, the receiver says Hi, they talk, then say Bye, and both hang up. Network protocols must specify the semantic rules for communication in all cases (e.g., what if one end crashes), and specify the syntactic format for the messages exchanged (field order, bit order etc.) Internet protocol specifications are called *RFCs*. You can find RFCs easily on the net using Google. For example, TCP is described in RFC 793.
- *Interface:* By contrast to a protocol, an interface specifies the rules governing *vertical* communication between a Layer N entity and a Layer $N + 1$ entity on the same computer. For

example, the IP interface specifies the form in which TCP can ask for a message to be sent. While protocols have to be standardized, interface specifications can vary from machine to machine.

- *PDU*: ISO uses the word Protocol Data Units (PDUs) for the messages that are exchanged between peer entities. In general, N-PDUs are the messages (data messages plus all headers up to the Layer N header) exchanged between Layer N -entities. An OSI convention is to use the first letter of the layer to prefix the corresponding PDU. Thus in Figure 4, we use TPDU to describe a message exchanged the transport layers, NPDU between network layers etc. Notice that the physical layer has no corresponding message because it deals with either bits or small groups of bits called symbols. Internet folks use different terminology, preferring to call a TPDU a *segment*, an NPDU a *packet*, and a DPDU a *frame*. We will mostly use the Internet terminology, so it may help you to memorize these terms early on.
- *SDU*: Just an interface is contrasted with a protocol, the Data Unit passed across an interface is called a Service Data Unit or SDU. An N-SDU is passed between layer N from Layer $N+1$. Thus using the same first letter convention, the data unit passed across the interface between transport and network is called a NSDU.
- *PDU versus SDU*: Normally, an N-PDU is an N-SDU together with a Layer N header. Normally, an NPDU is exactly the same as the SDU passed to layer $N - 1$. However, there are exceptions. For example, one SDU can be split into multiple PDUs at Layer N if the Layer N protocol allows only small PDUs.

There is one more crucial idea that we must emphasize in the ISO model, the concept of *strict layering*. To see this, we examine how a message is sent from a source to a destination through an intermediate router as shown in Figure 5.

In Figure 5 we have ignored the presentation and session layers because they are rarely used. Notice that each layer adds its own header as the message is sent down the layers, starting with the application layer header AH and ending with the Data Link header DH. The physical layer does not add any header.

When the message gets to a router, the receiving Data Link will examine the header, remove the Data Link Header, and then pass it up to the network layer. The network layer looks at the ultimate destination address (which is stored in the network header) and then passes it down to the Data link layer of the next link. This layer adds a Data Link header. When the message gets to the receiver, the process at the sender is reversed with each layer *stripping* off its header and passing it up to the next layer up.

The idea behind **strict layering**, which is implicit in the ISO model, is that *each layer should only look at its header and interface data to do its job*. Thus lower layers strip off their headers before passing up messages, and lower layers should not “peek” into higher layer headers. There is an obvious software engineering reason for this: we do not want changes to one layer do not cause other layers to be reimplemented. Imagining retrofitting millions of TCP implementations when Microsoft Outlook changes its mail protocol.

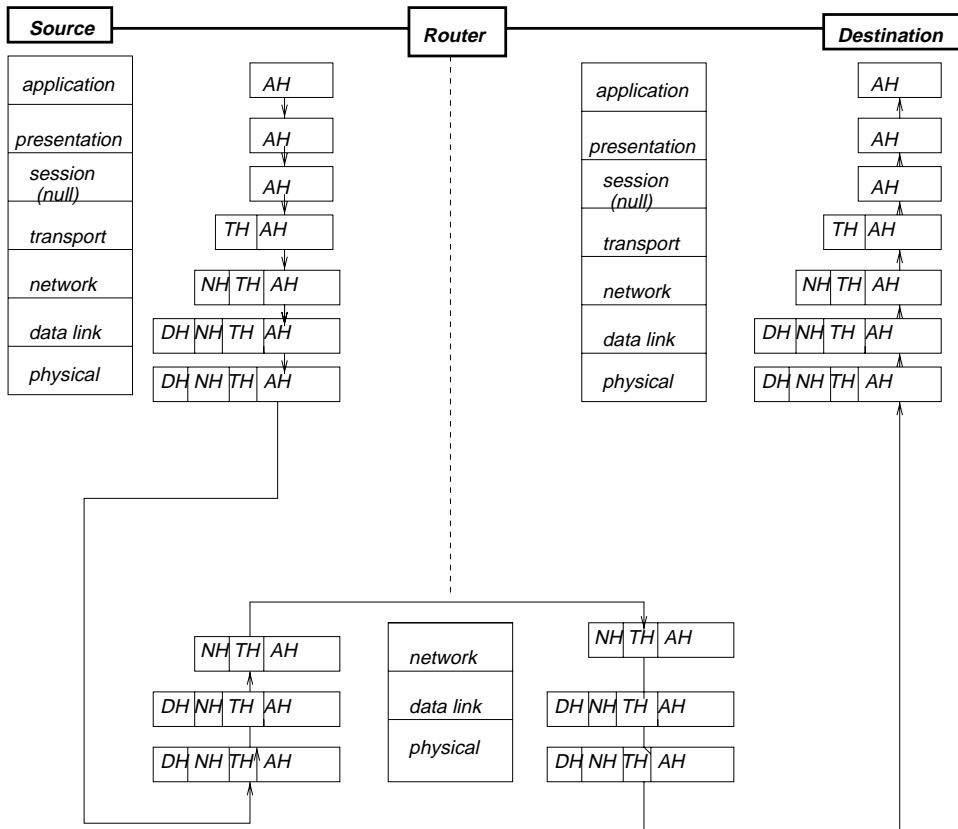


Figure 5: Adding headers while moving down the stack of layers, and removing them as we move up the stack

Strict layering is a good idea, and many implementations adhere to it. This does not mean that information cannot be passed between layers; it just must be passed via interfaces. For example, there is a recent proposal for a Explicit Congestion Notification (ECN) bit in TCP that is set by IP routers when they detect congestion. However, the point is to let the sending TCP know so that it can reduce its transmission rate. Since the IP router can only look at and write to the network header, it writes to the IP header. The bit travels with the message to the destination where the destination IP layer passes it *across the interface* to the receiving TCP. The receiving TCP then passes it in the TCP header of a returning acknowledgement to the sending TCP. Whew! That's a long, complex path for information flow. But it preserves layering.

Despite the spirit of idealism, strict layering is often abused in the marketplace. A classic example is a device called a *firewall*. Firewalls are sort of like routers that sit on the edge of a corporate network, deciding which packets to let in, hoping to keep “nastygrams” from hackers from coming in. Firewall devices often peek at TCP and higher layer headers to decide whether the packets are “good”. However, if these higher layer headers change, the firewall implementation will also have to change. Peeking at higher layer headers, as in the firewall example, is called a *layer violation*.

4 Picking the Layers

This section is for the curious, for those of you who want to really understand things, and who may end up possibly creating new networking ideas in the future. You can skip it on a first reading, but please look at Figure 6 even on a first reading.

Are layers picked out of a hat, or is there some rational reason for the division? That's a good question. The original rational for the ISO layers was as follows (paraphrased slightly from Tannenbaum's book, your optional additional textbook). In general, the criteria that follow provide a good set of guidelines for creating layered software for any complex problem.

- *Abstraction*: A layer should be created where a different level of abstraction is called for. An abstraction is a simplification of reality which is easier for users of the layer to deal with. For example, the VM layer in a Operating System provides the abstraction of infinite main memory. This is much simpler for programmers than older models where the programmer had to explicitly load data from disk into memory.
- *Well-defined function*: Each layer should ideally perform a well defined function. For example, presentation in the ISO model does format conversion.
- *Thin Interfaces*: If layers are separated arbitrarily, then lots of information may need to be passed across the interface. This would imply that the two layers we are trying to create are intimately related, and should probably be one layer. Thus layers should be created to *minimize* information flow across the interface.
- *Compact but Useful*: We should pick enough layers to divide up the original problem into manageable pieces, but not too many to make the result unwieldy.

Let us see how the TCP/IP layers and layer interfaces hold up against this checklist. A picture of the various abstractions provided by the TCP/IP layers is provided in Figure 6.

In Figure 6, consider the problem of transferring a file across the network. The abstraction provided to the user is similar to that of copying a named file within a single machine. In general, a reasonable strategy for networking interfaces is to make *unfamiliar* network operations look like *familiar* operations that can be performed within a single machine.

Typically, file transfer will be implemented by a file transfer process at the source and one at the destination. Clearly, one process should send data and the receiving process should consume it. Ideally, the transfer process should be asynchronous: the receiving process should be able work at an arbitrary speed. A common method for two asynchronous processes to exchange data in an Operating System is via a shared queue (e.g., UNIX pipes): the producer process writes to the tail of the queue, while the consumer process reads from the head of the queue. Such an abstraction would greatly facilitate writing a file transfer program: the sending file transfer process now only has to break up the file into segments and write them into the shared queue; the receiving file transfer process will assemble the pieces received from the queue into a file.

Thus a natural intermediate abstraction is the notion of a shared queue between two processes on *different* machines, extending a notion we are familiar with from the case of a *single* machine.

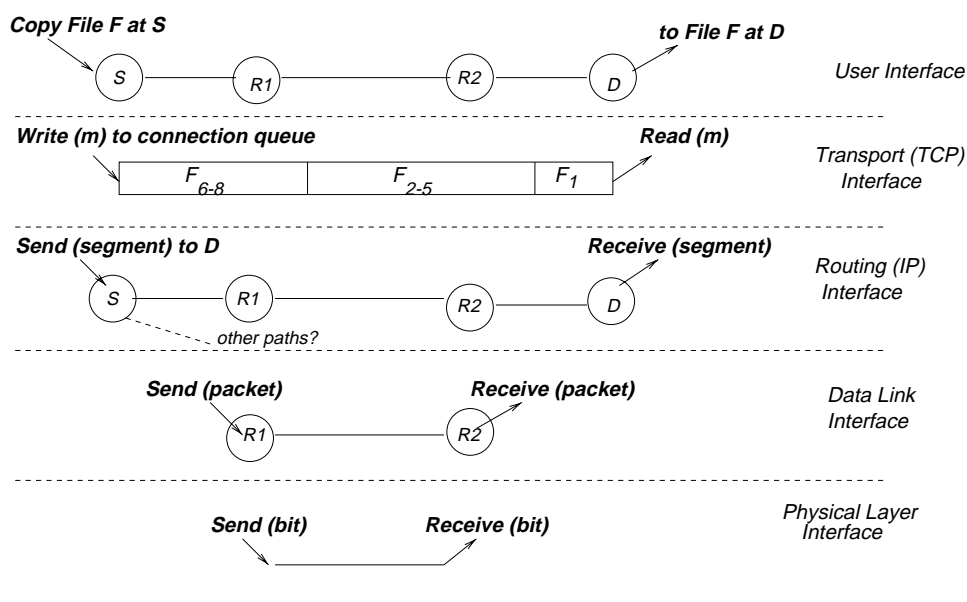


Figure 6: Picture of the abstractions provided by each of the TCP/IP layers

This is exactly what TCP provides. Most TCP implementations do so using what are called *socket* interfaces. A socket is a descriptor of a local queue at a machine. The process at the source opens a local socket queue at the sending machine, and the receiver process does the same at the receiving machine. Then a TCP connection is initiated to “connect” the local socket queues at the sender and receiver machines. The joined queues now represent a shared queue. The sender process writes to the shared queue, and the receiver reads. This is almost exactly like the case of a single machine shared queue except for a few tricky details. For example, consider what happens when one machine crashes independently of the other; this is not a problem for two processes communicating within the same machine, but must be taken into account for TCP interfaces.

TCP must ensure that all its data is sent reliably and in sequence from one end to another because that is what a shared queue guarantees. But real networks can lose data. Thus TCP takes portions of the file, adds a TCP sequence number, transmits the resulting segment to the receiver, and waits for an acknowledgement — think of an acknowledgement or ack like a return receipt in certified mail. If an ack does not come back in time, TCP retransmits the corresponding segment. In this way, TCP ensures data is delivered reliably and in sequence, exactly the way it would be in a shared queue.

However, it is clear that the job of sending each segment to the destination and picking a route for the segment, is a different abstraction and a separate function. Thus it makes to separate this abstraction (sending a segment to a specified Internet address) as the function of a separate IP network layer.

Next, having picked the best route, at each hop, the packet (segment plus network layer header) has to be sent to the next hop. In particular, because messages are clocked out as bits on channels, there is a framing problem to group received bits into separate frames (packets plus data link headers). This is clearly very different from the problem of picking routes, and might as well be

separated into a separate function.

Finally, because wires only allow sending energy and not logical concepts like bits, some entity must convert each bit into energy that can flow across the wire or channel. Again, this is a different function and a very different abstraction. Thus it is logical to separate this function into a separate physical layer.

If we apply the criteria for layer division to the TCP/IP stack, the TCP/IP stack does quite well. Each layer (see Figure 6) does represent a separate abstraction and separate function. Recall that the abstractions provided from top to bottom are copying named files, providing a shared queue, providing a method to send segments across the network, providing a way to send packets one-hop across a bit at a time channel, and converting bits to energy for transmission. The interfaces are all quite small. File transfer only has to pass TCP an identifier for the remote machine address, the remote socket identifier, and the data; TCP has to only pass the Internet address of the destination and the data to IP; IP only has to pass the Data Link address of the next hop to the Data Link; and the Data Link only passes the bit values to the physical layer. Thus all interfaces are quite thin, and the final number of layers (5) is compact and and yet useful.

This would be sufficient if we were just training you about the Internet of *today*. But in order to create the Internet of *tomorrow*, which I hope you will do, hopefully you will ask yourself the following question. Are there other partitionings, are there other useful abstractions one could use. Indeed, there are.

For example, it is not clear that the shared queue abstraction provided by TCP is the only natural abstraction for processes communicating across the network. Another obvious operating system model is a procedure call between processes. This suggests a *Remote Procedure Call (RPC)* abstraction where a client process could call a procedure on a server machine across the network, and would block till the results return. Sun has built many of its network services and file systems around a RPC model. Many people implement RPC using a RPC layer layered above TCP. Another natural abstraction that has not been as successful commercially is as follows. One could export a shared memory interface across the network via what is called *distributed shared memory*.

A second example of a competing abstraction is provided by the network layer interface. The first networks, inspired by the telephone company, used a *virtual circuit* abstraction. The abstraction is that the sending TCP first sets up a “call” to the destination TCP as if it were dialing a telephone. The call then sets up state at each router and is given a call number. Then the data is sent on that call number. Since data is often not as smooth as voice, it does not make sense to reserve bandwidth for calls; so in the data world these are known as “virtual” circuits as opposed to “physical” telephone circuits that are reserved for each call. Many important data networks of the past and present (notably ATM networks today, where ATM stands for Asynchronous Transfer Mode) use virtual circuit interfaces very different from the IP interface. In fact, its quite a challenge to run IP (postal model) over ATM (telephone model) because of the different abstractions they use.

5 How a Web Transfer occurs

The following section is a very quick overview of the web protocols, TCP, and IP routing. Skip it if you want to wait for later in the class. However, it may satisfy your curiosity for now, and allow you to wait more patiently for the detailed explanation that comes later. It should also help you relate things you do commonly (clicking on web pages, seeing the messages that flash on the bottom of your screen) with what we will cover in this class.

When you point your web browser to `www.cs.ucsd.edu`, your browser first converts the destination host name (i.e., `cs.ucsd.edu`) into a 32 bit Internet address such as `132.239.51.18` by making a request to a local DNS name server; this is like dialing Directory Information to find a telephone number. UCSD has a local DNS server that can consult global root DNS servers if needed. A 32 bit IP address is written in dot decimal form for convenience; each of the four numbers between dots (e.g., 132) represents the decimal value of a byte. Domain names such as `cs.ucsd.edu` are only used in user interfaces; the Internet transport and routing protocols only deal with 32-bit Internet addresses.

Networks lose and reorder messages. If a network application cares that all its messages are received in sequence, the application can subcontract the job of reliable delivery to a transport protocol such as the Transmission Control Protocol (TCP). TCP's job is to provide the sending and receiving applications with the illusion of two shared data queues in each direction — despite the fact that the sender and receiver machines are separated by a lossy network. Thus whatever the sender application writes to its local TCP send queue should magically appear in the same order at the local TCP receive queue at the receiver, and vice versa.

Since web browsers care about reliability, the web browser at sender S (Figure 7) first contacts its local TCP with a request to set up a *connection* to the destination application. The destination application is identified by a well known port number (such as 80 for web traffic) at the destination IP address. If IP addresses are thought of as telephone numbers, port numbers can be thought of as extension numbers. A connection is the shared state information — such as sequence numbers and timers — at the sender and receiver TCP programs that facilitate reliable delivery.

Figure 7 is an example of what we call a time-space figure with time flowing downwards and space represented horizontally. A line from S to D that slopes downwards represents the sending of a message from S to D that arrives at a later time. We will use time-space figures throughout this course to depict protocol execution scenarios.

To set up a connection, the sending TCP (Figure 7) sends out a request to start the connection called a SYN message with a number X the sender has not used recently. If all goes well, the destination will send back a SYN-ACK to signify acceptance along with a number Y that the destination has not used before. Only after the SYN-ACK is the first data message sent.

In Figure 7, the sender is a web client, whose first message is a small (say) 25 byte HTTP GET message for the web page (e.g., `index.html`) at the destination. To ensure message delivery, TCP will retransmit all messages until it gets an acknowledgement. To ensure that data is delivered in order and to correlate acks with data, each byte of data in a packet carries a sequence number. In TCP only the sequence number of the first byte in a packet is carried explicitly; the sequence numbers of the other bytes are implicit based on their offset.

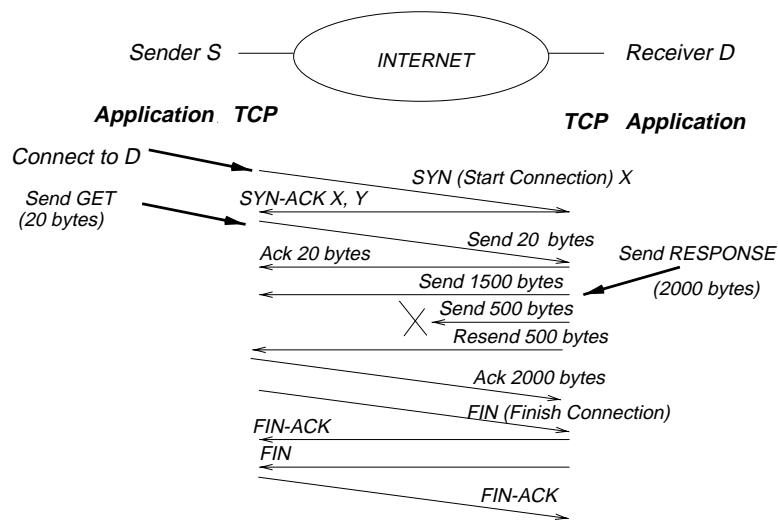


Figure 7: Time-space figure of a possible scenario for a conversation between a Web Client S and a Web Server D as mediated by the reliable transport protocol TCP.

When the 25 byte GET message arrives at the receiver, the receiving TCP delivers it to the receiving web application. The web server at D may respond with a web page of (say) 1900 bytes that it writes to the receiver TCP input queue along with an HTTP header of 100 bytes, making a total of 2000 bytes. TCP can choose to break up the 2000 byte data arbitrarily into packets; in Figure 7 we chose to use two packets of 1500 and 500 bytes. Assume for variety that the second packet of 500 bytes is lost in the network; this is shown in a time-space picture by a message arrow that does not reach the other end. Since the receiver does not receive an ACK, the receiver retransmits the second packet after a timer expires. Note that ACKs are cumulative: a single ACK can acknowledge all data received in sequence. Finally, if the sender is done, the sender begins closing the connection with a FIN (for FINISH) message that is also acked (if all goes well), and the receiver does the same.

Once the connection is closed with FIN messages, the receiver TCP keeps no sequence number information about the sender application that terminated. But networks can also cause duplicates (because of say retransmissions) of SYN and DATA packets that appear later and confuse the receiver. This is why the receiver in Figure 7 does not believe any data that is in a SYN message until it is validated by receiving a third message containing the unused number Y the receiver picked. If Y is echoed back in a third message, then the initial message is not a delayed duplicate since Y was not used recently.

This preliminary dance featuring a SYN and a SYN-ACK is called TCP's 3-way handshake. It allows TCP to forget about past communication at the cost of increased latency to send new data. In practice, the validation numbers X and Y do double duty as the initial sequence numbers of the data packets in each direction. This works because sequence numbers need not start at 0 or 1 as long as both sender and receiver use the same initial value.

The TCP sequence numbers are carried in a TCP header contained in each packet. The TCP header contains 16 bits for the destination port (recall that a port is like a telephone extension

that helps identify the receiving application), 16 bits for the sending port (analogous to a sending application extension), a 32 sequence number for any data contained in the packet and a 32 bit number acknowledging any data that arrived in the reverse direction. There are also flags that identify packets as being SYN, FIN etc. A packet also carries a routing header and a link header that changes on every link in the path.

If the application is (say) a videoconferencing application that does not want reliability guarantees, it can choose to use a protocol called UDP instead of TCP. UDP is TCP without the acks and retransmissions. Thus the only sensible fields in the UDP header corresponding to the TCP header are the destination and source port numbers. Like ordinary mail versus certified mail, UDP is cheaper in bandwidth and processing but offers no reliability guarantees. We will provide more information about TCP and UDP later, but if you are curious we highly recommend the book “TCP Illustrated” by Richard Stevens.

5.1 Routing Protocols

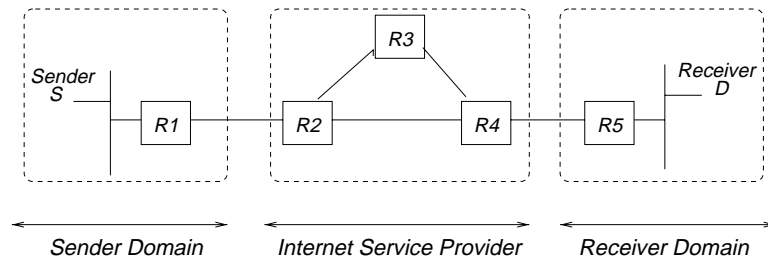


Figure 8: A sample network topology corresponding to the Internet of Figure 7

Figure 8 shows a more detailed view of a plausible network topology between the web client S and web server D of Figure 7. The sender is attached to a Local Area Network such as an Ethernet to which is also connected a router $R1$. Routers are the automated post offices of the Internet that consult the destination address in an Internet message (often called a packet) to decide on which output link to forward the message.

In the figure, the sender S belongs to an administrative unit (say a small company) called a domain. In this simple example, the domain of S consists only of an Ethernet and a router $R1$ that connects to an Internet Service Provider (ISP) through router $R2$. Our Internet Service Provider is also a small outfit, and consists only of 3 routers $R2$, $R3$ and $R4$ connected by fiber optic communication links. Finally, $R4$ is connected to a router $R5$ in D 's domain that leads to the receiver D .

Internet routing is broken into two conceptual parts, called *forwarding* and *routing*. We first consider forwarding, which explains how packets move from S to D through intermediate routers.

When S sends a TCP packet to D , it first places the IP address of D in the routing header of the packet and sends it to neighboring router $R1$. Forwarding at endnodes such as S and D is kept simple, and consists of sending the packet to an adjoining router. $R1$ realizes it has no information about D and so passes it to ISP router $R2$. When it gets to $R2$, $R2$ must choose to send the packet

to either $R3$ or $R4$. $R2$ makes its choice based on a forwarding table at $R2$ which specifies (say) that packets to D should be sent to $R4$. Similarly, $R4$ will have a forwarding entry for traffic to D that points to $R5$. We will describe how forwarding entries are compressed using prefixes later. In summary, an Internet packet is forwarded to a destination by following forwarding information about the destination at each router. No router knows the complete path to D , but only the next hop to get to D .

While forwarding must be done at extremely high speeds, the forwarding tables at each router must be built by a routing protocol. For example, if the link from $R2$ to $R4$ fails, the routing protocol within the ISP domain should change the forwarding table at $R2$ to forward packets to D to $R3$. Typically, each domain uses its own routing protocol to calculate shortest path routes within the domain. Two main approaches to routing within a domain are *distance vector* and *link state*.

In the distance vector approach, exemplified by the protocol RIP, the neighbors of each router periodically exchange distance estimates for each destination network. Thus in Figure 8, $R2$ may get a distance estimate of 2 to D 's network from $R3$ and a distance estimate of 1 from $R4$. Thus $R2$ picks the shorter distance neighbor $R4$ to reach D . If the link from $R2$ to $R4$ fails, $R2$ will time out this link, set its estimate of distance to D through $R4$ to infinity, and then choose the route through $R3$. Unfortunately, distance vector takes a long time to converge when destinations become unreachable.

Link state routing avoids the convergence problems of distance vector by having each router construct a *link state packet* listing its neighbors. For instance in Figure 8, $R3$'s link state packet (LSP) will list its links to $R2$ and $R4$. Each router then broadcasts its LSP to all other routers in the domain using a primitive flooding mechanism; LSP sequence numbers are used to prevent LSPs from circulating forever. When all routers have each other's LSP, every router has a map of the network and can use Dijkstra's algorithm to calculate shortest path routes to all destinations. The most commonly used routing protocol used *within* ISP domains is a link state routing protocol called OSPF.

While shortest cost routing works well within domains, the situation is more complex for routing between domains. Imagine that we modified Figure 8 so that the ISP in the middle, say ISP A , does not have a direct route to D 's domain but instead is connected to ISPs C and E , each of which has a path to D 's domain. Should ISP A send a packet addressed to D to ISP C or E ?

6 A Look Ahead at Some Hard Problems

The rest of this course is a systematic study of the layers we described in overview, layer upon layer, precept upon precept, in strictly bottom up fashion. It is worth noting at this time that all layers (and in fact all computer systems from operating systems to databases) have common problems: synchronization in the face of errors and asynchrony, addressing, multiplexing, interconnection. Watch for these common problems as we go along.

So what do we do for the next 20 lectures, given that the networking picture we drew seemed so "easy". Well, there are some hard problems whose solutions we will describe. Sample problems

you will learn the answer to in each layer are

- *Transport:*
 - **Congestion Control.** How does a TCP sender know how to speed up or slow down depending on current Internet speed? How is congestion sensed and reacted to. This is the famous TCP *Slow-start mechanism* we study in the last two lectures.
 - **Connection Management:** How does TCP prevent old conversations between the same pair of machines from mixing in with new conversations. TCP uses a very famous mechanism called *3-way handshakes* that we briefly alluded to above.
- *Routing:*
 - **CIDR:** How does IP allow various sizes of networks in allocating addresses. We want to allow large ISPs and yet small mom-and-pop networks so we want a flexible division of the bits in a network number, versus bits for the host within a network. We will study how the CIDR proposal in IP really helped save the Internet from running out of addresses. We will also study how it requires a more complex forwarding procedure called *longest matching prefix lookup*.
 - **BGP:** How does IP calculate routes between multiple competing providers? Within domains like UCSD, it is tricky but not impossible to calculate shortest path routes, even after failures. However, its much more tricky to calculate routes across ISPs that don't necessarily like each other and have different policies. We will briefly look at the Border Gateway Protocol (BGP) and its idiosyncrasies.
- *Data Link:* At a party you can describe Ethernet as follows. Anybody who wants to can send. If two guys send at the same time, there is a collision. Both toss coins and one wins and transmits. This is a reasonable first picture but it hides masses of subtle details like:
 - **Min Packet Sizes:** How does Ethernet ensure that if one nodes detects a collision, all nodes do? If this were not the case, all kinds of things go wrong. A key mechanism is the use of a minimum packet size of 64 bytes. We will study the need for this crazy restriction around Lecture 12.
 - **Dynamic Backoff:** How can Ethernet sort out 2 sender collisions quickly while being able to sort out even 32 sender collisions? Ethernet does so using a dynamic coin-tossing scheme. Look out for it.
- *Physical Layer:*
 - **Clock Recovery:** How does a receiver reconstruct bits from physical signals despite speed differences? Bits are recovered from physical signals by sampling. But when should the sampling instants occur? This is a crucial problem and is solved by adding redundant *transitions*. We will study this in Lecture 4.
 - **Media Issues:** When should a manager use wireless versus fiber versus satellite? What are the tradeoffs between infrared and radio? We will study comparative media in Lecture 5.

7 Conclusions

Perhaps you are a little overwhelmed with the amount of overview information in this chapter. That's to be expected. It's more than enough if you get a rough idea of the overall stack because we will treat each of these topics in detail in subsequent lectures. The only thing you will be tested on from this chapter will be the OSI terminology and the concept of strict layering. So make sure you really follow Section 3. You may find it helpful to also read the first chapter of Perlman's book.

More importantly, we hope you will keep the main pictures in mind. We specially recommend the broadest overviews in Figure 1 and Figure 2, the OSI terminology in Figure 4 and Figure 5, and finally a picture of the various TCP/IP abstractions in Figure 6. Please take a moment to look at these pictures again and read their captions. And before we begin a new layer (physical, Data Link, routing, transport) take a moment to review those pictures again, just as you might consult a map before making a major transition in a road journey.

This chapter also introduces you to some of the peculiarities of my personal style of teaching and writing: these include constantly giving big pictures (e.g., Hazel's hats), stressing important ideas (e.g., layering, strict layering), questioning everything (e.g., why are headers needed), informality and potential verbosity (e.g., some irreverence), and references to fields of study outside networking (e.g., theory, operating systems, history, literature). Style can attract, but it can also repel. If the style bothers you, try separating out the substance from the style: there's some good stuff to be had if you look for it.

Finally, stay tuned as we march on to the details. The trailer is over. Let the movie begin!