MIT 6.02 **DRAFT** *Lecture Notes*
Spring 2010 (Last update: April 26, 2010)
Comments, questions or bug reports?
Please contact 6.02-staff@mit.edu

LECTURE 21
Network Layering

Thus far in 6.02, we have developed a set of techniques to solve various problems that arise in digital communication networks. These include techniques to improve the reliability of communication in the face of inter-symbol interference, noise, collisions over shared media, and of course, packet losses due to congestion. The second set of techniques enable a network to be shared by multiple concurrent communications. We have studied a variety of schemes to improve reliability: using eye diagrams to choose the right number of samples per bit, using block and convolutional codes to combat bit errors, and using retransmissions to develop reliable transport protocols. To enable sharing, we studied frequency division multiplexing, various MAC protocols, and how packet switching works. We then connected switches together and looked at how network routing works, developing ways to improve routing reliability in the face of link and switch failures.

But how do we put all these techniques together into a coherent *system*? Which techniques should run in different parts of the network? And, above all, how do we design a *modular* system whose components (and techniques) can be modified separately? For example, if one develops a new channel coding technique or a new reliable data delivery protocol, it should be possible to incorporate them in the network without requiring everything else to change.

In communication networks (as in some other systems), **layering** is a useful way to get all these techniques organized. This lecture is not about specific techniques or protocols or algorithms, but about developing a conceptual framework for which entities in the network should implement any given technique. Layering has stood the test of time, proving itself to be a powerful way to design and evolve networks.

■ 21.1 Layering

Layering is a way to architect a system; with layering, the system is made up of a *vertical stack of protocols*. The services, or functions, provided by any given layer depends solely on the layer immediately below it. In addition, each layer of the stack has a *peer interface* with the same layer running on a different node in the network.

As implemented in most modern networks, including the Internet architecture, there

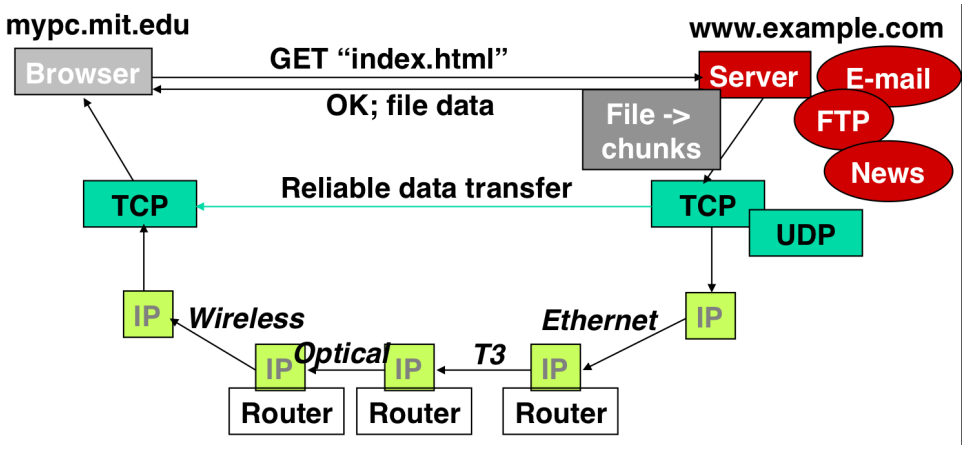

Figure 21-1: An example of protocol layering in the Internet.

are five layers: physical, data link, network, transport, and application. In reverse order, the application is ultimately what users care about; an example is HTTP, the protocol used for web transfers. HTTP runs atop TCP, but really it can run atop any protocol that provides a reliable, in-order delivery abstraction. TCP runs atop IP, but really it can run atop any other network protocol (in practice, TCP would have to be modified because historically TCP and IP were intertwined, and the current standard unfortunately makes TCP dependent on some particular details of IP). IP runs atop a wide range of network technologies, each implementing its own data link layer, which provides framing and implements a MAC protocol. Finaly, the physical layer handles modulation and channel coding, being responsible for actually sending data over communication links. Figure 21-1 shows an example of how these layers stack on top of each other.

In the purist's version of the Internet architecture, the switches in the network ((i.e., the "insides" of the network) do not implement any transport or application layer functions, treating all higher layer protocols more or less the same. This principle is currently called "network neutrality", but is an example of an "end-to-end argument", which says that components in a system must not implement any functions that need to be done at the ends, unless the performance gains from doing so are substantial. Because the ends must anyway implement mechanisms to ensure reliable communications against all sorts of losses and failures, having the switches also provide packet-level reliability is wasteful. That said, if one is running over a high loss link or over a medium with a non-negligible collision rate, some degree of retransmission is a perfectly reasonable thing to do, as long as the retransmissions aren't persistent, because the end point will end up timing out and retransmitting data anyway for packet streams that require reliability.

Figure 21-2 illustrates which layers run where in a typical modern network.

■ 21.1.1 Encapsulation

A key idea in network layering is **data encapsulation**. Each layer takes the data presented to it from its higher layer, treats that as an opaque sequence of bytes that it does not read or manipulate in any way, adds its own headers or trailers to the beginning and/or end of the presented data, and then sends this new chunk of data to the layer below it. The lowest

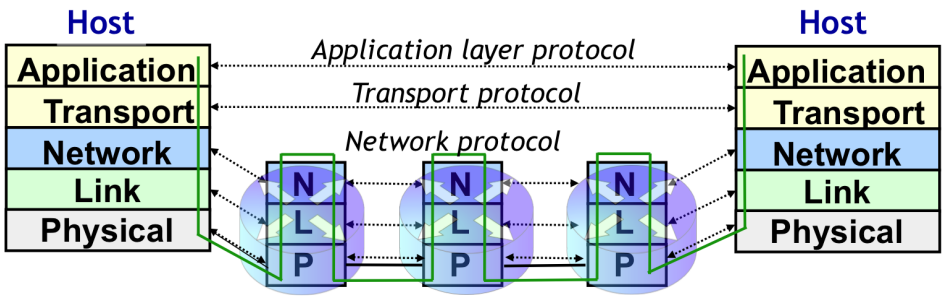

Figure 21-2: Layering in a typical modern network. The "insides" of the network, i.e., the switches, do not implement any transport and application layer functions in the pure version of the design.

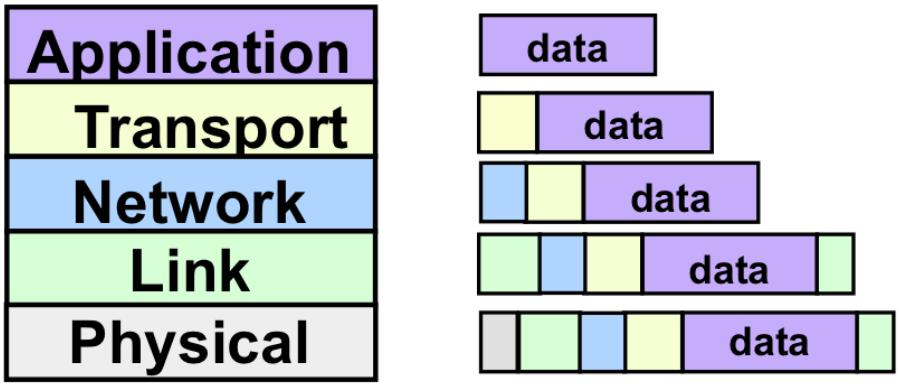

Figure 21-3: Encapsulation is crucial to layered designs; the ability to multiplex multiple higher layers atop any given layer is important.

layer, the *physical layer*, is responsible for actually sending the data over the communication link connecting the node to the next one en route to the eventual destination. Figure 21-3 illustrates how data encapsulation works.

For example, let's look at what happens when a user running a web browser visits a web link. First, an entity in the network takes the URL string and converts it into information that can be understood by the network infrastructure. This task is called *name resolution* and we won't be concerned with it in 6.02; the result of the name resolution is a network address and some other identifying information (the "port number") that tells the application where it must send its data.

At this point, the browser initiates a *transport layer* connection to the network address and port number, and once the connection is established, passes on its request to the transport layer. The most common transport layer in the Internet today is TCP, the Transmission Control Protocol. In most systems, the interface between the application and the transport layer is provided via the "sockets" interface, which provides primitives to open, write, read, and perform other functions, making the network look like a local file with special semantics to the application. In principle, the application can change from TCP to some other protocol rather easily, particularly if the other protocol provides semantics similar

to TCP (reliable, in-order delivery). That is one of the benefits of layering: the ability to change a layer without requiring other layers to change as well, as long as the service and semantics provided by the layer remain intact.

TCP (or any transport layer) adds its own header to the data presented to it via the sockets interface, including a sequence number, a checksum, an acknowledgment number, a port number (to identify the specific instance of the transport protocol on the machine that this connection is communicating with), among other fields. When TCP sends data, it passes the bytes (including the header it added) to the next lower layer, the *network layer* (IP, or the Internet Protocol, in the Internet architcture). IP, in turn adds its own information to the header, including a destination address, the source address of the originating node, a "hop limit" (or "time to live") to flush packets that have been looping around, a checksum for the header, among other fields. This process then continues: IP hands over the data it has produced to the *data link* layer, which adds its own header and trailer fields depending on the specific communiction medium it is operating over, and the data link layer hands the data over to the *physical layer*, which is ultimately responsible for sending the data over the communication medium on each hop in the network.

Each layer multiplexes data from multiple higher layer protocols. For example, the IP (network) layer can carry traffic belonging to a variety of transport protocols, such as TCP, UDP, ICMP, and so on. TCP can carry packets belonging to a large number of different application protocols, such as HTTP, FTP, SMTP (email), and numerous peer-to-peer applications. The way in which each layer knows which higher layer's data is currently being carried is using a demultiplexing field in its header. For example, the IP layer has a "protocol" field that says whether the packet belongs to TCP, or UDP, or ICMP, etc. The TCP layer has a "port number" that identifies the application-layer protocol. At a lower layer, the link layer has a field usually called the "ethertype" that keeps track of whether the data belongs to IPv4, or IPv6, or AppleTalk, or any other network layer protocol.