Ten most important ideas/concepts to take away so far....

Nuts and bolts of computer networks: Computer networks consist of end systems, packet switches, and communication links. End systems – also called hosts – include desktop PCs, laptops, hand-held network devices (including cell phones, PDAs, and blackberries), sensors, as well as servers (such as Web and mail servers). Just as cities are interconnected by a network of roads and intersections, end systems of a computer network are interconnected by a network of communication links and packet switches. Communication links can be wired and wireless.

Distributed applications: A computer network enables distributed applications. A distributed application runs on end systems and exchanges data via the computer network. Distributed applications include Web surfing, e-mail, instant messaging, Internet phone, distributed games, peer-to-peer file sharing, television distribution, and video conferencing. New distributed applications continue to be invented and deployed on the Internet.

Packet switching: When one end system sends data to another the end system, the sending end system breaks the data into chunks, called packets. Similar to the process of delivering post-office mail, the Internet transports each packet separately, routing a packet to its destination using a destination address that is written into the packet. When a packet switch receives a packet, it uses the packet's destination address to determine on which link it should forward the packet. Thus a packet switch performs "packet switching," forwarding incoming packets to outgoing links on a packet by packet basis. Also, packet switches typically "store and forward" packets – that is, before a switch begins to forward a packet on an outgoing link, if first receives and stores the entire packet.

Protocol: As defined on Page 8, a protocol defines the format and order of messages exchanged between two or more communication entities, as well as the actions taken on the transmission and/or receipt of a message or other event. Computer networks make extensive use of protocols. Figure 1.2 provides an analogy between a human protocol and a computer network protocol for messages exchanged between a Web browser and a Web server. In this example, the Web browser first sends an introductory message to the server; the server responds with its own introductory message; the browser then sends another message, requesting a specific Web page; finally, the server sends a last message, which includes the requested Web page.

Circuit-switching: Computer networks constitute one major class of computer networks. Another major class of communication networks is traditional digital telephone networks. Traditional digital telephone networks do not use packet switching to move data from source to destination, but instead uses a technique known as circuit switching. In circuit switching, before transmitting data between two end system, the network establishes a dedicated end-to-end connection between the two end systems and reserves bandwidth in each link along the connection. The reserved connection bandwidth is "wasted" whenever the two end systems are not sending data.

Physical media and access networks: The communication links in a computer network may have different physical media types. Dial-up modem links, DSL, and most Ethernet links are copper wire. Cable links are made of coaxial cable. Long-haul Internet

backbone links are made of fiber optics. In addition to these wired links, there is a plethora of wireless links, including wi-fi links, blue-tooth links, and satellite links. An access link is a link that connects the end system to the Internet. Access links can be copper wire, coaxial cable, fiber optics or wireless. A tremendous variety of media types can be found in the Internet today.

Network of networks: The Internet consists of many interconnected networks, each of which is called an Internet Service Provider (ISP). Each ISP is itself a network of packet switches and communication links. Thus, the Internet is a network of networks. The ISPs are roughly organized in a hierarchy. ISPs at the bottom of the hierarchy access ISPs, such as residential ISPs, university ISPs, and enterprise ISPs. The ISPs at the top of the hierarchy are called tier-1 ISPs and typically include long-haul intra- and inter-continental fiber links. Tier-n ISPs provide service – for a price – to tier-(n+1) ISPs. Each ISP is independently managed. However, the ISPs employ a common protocol suite called the Internet Protocol, which is better known as IP.

Propagation and transmission delay: Propagation and transmission delays play critical role in the performance of many distributed applications. Perhaps the best way to understand propagation and transmission delay and why they are different animals is to play a little with the transmission versus propagation delay applet at the textbook's online site. The propagation delay over a link is the time it takes a bit to travel from one end of the link to the other. It is equal to the length of the link divided by propagation speed of the link's physical medium. The transmission delay is a quantity that relates to packets and not bits. The transmission delay for a link is equal to the number of bits in the packet divided by the transmission rate of the link. It is the amount of time it takes to push the packet onto the link. Once a bit is pushed onto a link it needs to propagate to the other end. The total delay across a link is thus equal to the sum of the transmission delay and the propagation delay.

Queuing delay and packet loss: Many packets can arrive at a packet switch roughly at the same time. If these packets need to be forwarded on the same outbound link, all but one of these packets will have to "queue," that is wait for their turns to be transmitted. This waiting introduces a queuing delay. Furthermore, if the queue of packets becomes very large, the packet switch's buffer may become exhausted, causing packets to become dropped or "lost". Queuing delay and packet loss can severely impact the performance of an application.

Protocol layers. A typical computer network makes use of many, many protocols – easily hundreds of protocols. To deal with this complexity, the protocols are organized into layers. These protocol layers are arranged in a "stack". For example, the Internet organizes its protocols into five layers – namely, from top to bottom, the application layer, the transport layer, the network layer, the link layer, and the physical layer. The protocols of layer n use the services provided by the protocols at the layer n-1 (the layer below). This abstract concept, often difficult to grasp at first, will become clearer as we delve into the different protocol layers.

With the application-layer as the highest layer in the protocol stack, one can say that *all* other layers in the stack exist only to provide services to the application. Indeed this is the case, as applications are the *raison d'etre* for computer networks. For without networked applications, there would be no need for computer networks in the firs place!

Encapsulation. This concept is so important that we need to extend Chapter 1's top-10 list into a top-11 list. When the sender-side application-layer process passes an application-level data unit (an application *message*) to the transport layer, that message becomes the *payload* of the transport layer segment, which also contains additional transport-layer header information, e.g., information that will allow the transport layer at the receiver side to deliver the message to the correct receiver-side application. Conceptually, one can think of the transport layer segment as an envelope with some information on the envelope (the segment's header fields) and the application layer payload as a message within the envelope. The transport layer passes the transport layer segment to the network layer. The segment becomes the payload of the network-layer *datagram*, which has additional fields used by the network layer (e.g., the address of the receiver). Conceptually, one can think of the transport layer segment as envelope within an envelope, with some information on the outside of the network-layer envelope. Finally, the network layer datagram is passed to the link layer, which encapsulates the datagram within a link-layer *frame*.

Continuing with our envelope analogy, a receiver-wide protocol at layer n will look at the header information on the envelope. As we will see, the protocol may pass the envelope back down to the lower layer (e.g., for forward to another node), or open the envelope and extract the upper layer payload, and pass that upper-layer envelop up to layer n+1. Like layering, the concept of encapsulation often strikes students as rather vague at first. However, we'll see the technique being used so often throughout the book, that it will soon become second nature,

Chapter 2: The Application Layer

10 most important ideas/concepts to take away from this Chapter

Application-layer protocol. In chapter 1 (page 8) we noted that *"A **protocol** defines the format and the order of messages exchanged between two or more comunicating entites, as well as the actions taken on the transmission and/or receipt of a message or other event."* In this chapter we have seen that processes that send and receive messages in an application-layer protocol. As a review, can you identify a few of the messages exchanged and actions are taken by the following protocols that we studied in this chapte: HTTP, FTP, DNS, SMTP?

Client/server versus peer-to peer. These are the two approaches taken for structuring a network application. In the client/server paradign (page 75), a client process request service (by sending one or messages) to a server process. The server process implements a service by reading the client reqests, performing some action (e.g., finding a web page in the case of an http server) and sending one r messages in reply (in thecase of http, returning the requested object). In a peeer-to-peer approach the two ends of the protocol are equals, e.g., as in a telephone call.

The two services provided by the Internet's transport layer: relaible, congestion-controlled data transfer (TCP), and unreliable data transfer (UDP). These are the only two services aailable to an Internet application. The Internet transport layers do not provide a minimum guaranteed transfer rate, or a bound on the delay from source to destination.

HTTP: request/response interaction. The HTTP protocol is simple, a client (web browser) makes a request with a GET message, and a web server provides a reply (Figure 2.6). This is a classical client/server approach. Since HTTP uses TCP to provide for reliable transfer of the GET request from client-to-server, and the reply from server-to-client, a TCP connection must first be set up. This means that a TCP setup request is first sent from the TCP in the client to the TCP in the server, with the TCP server replying back to the TCP client. Following this exchange, the HTTP GET message can be sent over the TCP connection from client-to-server, and the reply received. See Figure 2.7. With non-persistent HTTP, a new TCP connection must be set up each time. With persistent HTTP, multiple HTTP GET messages can be sent over a single TCP connection, with the consequence performance gains of not having to set up a new TCP for each of the HTTP requests beyond the first.

Caching. Caching is the act of saving a local copy of a requested piece of information (web document, DNS translation pair) that is retrieved from a distant location, so that if the same piece of information is requested again, it can be retrieved from the local cache, rather than having to retrieve the information from the distant location. Caching can improve performance by decreasing response time (since the local cache is closer to the requesting client) and avoiding the use of scare resources (such as the 1.5 Mbps access link in Figures 2.11 and 2.12). Can you think about how you using caching in

your every day life? I'll write down a phone number on a piece of paper and keep it in my pocket, rather than have to loop up the number again in phone book.

DNS: core infrastructure implemented as an application-layer process. The DNS is an application protocol. The name-IP-address translation service is performed at DNS servers, just as any application provides a service via a server. But the DNS service is a very special network service – without it the network would not be able to function. And yet it is implemented in very much the same way as other network applications.

FTP: separate control and data. Students often ask us why we include an "old" application such as FTP. FTP is a nice example of a protocol that separates control and data messages. As shown in Figure 2.15 control and data messages are sent over separate TCP connections. This logical and physical separation of control and data (rather than mixing the two types of messages in one connection) helps make the structure of such an application "cleaner."

TCP sockets: accept(), and the creation of a new socket. The one "tricky" thing with TCP sockets is that a new socket is created when a TCP server returns from an accept() system call. We've called the socket on which the server waits when performing the accept as a "welcoming socket." The socket returned from the accept() is used to communicate back to the client that was connected to the server via the accept(); see Figure 2.28.

UDP socket: send and pray on the receiving side; datagrams from many senders on the receiving side. Since UDP provides an unreliable data transfer service, a sender that sends a datagram via a UDP socket has no idea if the datagram was ever received by the receiver (unless the receiver is programmed to send a datagram back that acknowledges that the original datagram was received). On receiving side, datagram from many different senders be received on the same socket.

Pull versus push. How does one application process get data to/from another application process. In a pull system (such as the web), the data receiver must explicitly request ("pull") the information. In a push system, the data holder sends the information to the receiver without the receiver explicitly asking for the data (as in SMTP, when an email is "pushed" from sender to receiver).

10?

Ten most important ideas/concepts to take away from Chapter 3

Logical communication between two processes: Application processes use the logical communication provided by the transport layer to send messages to each other, free from the worries of the details of the network infrastructure used to carry these messages. Whereas a transport-layer protocol provides logical communication between *processes*, a network-layer protocol provides logical communication between *hosts*. This distinction is important but subtle; it is explained in the textbook on page 186 with a cousin/house analogy. An application protocol lives only in the end systems and is not present in the network core. A computer network may offer more than one transport protocol to its applications, each providing a different service model. The two transport-layer protocols in the Internet – UDP and TCP – provide two entirely different service models to applications.

Multiplexing and demultiplexing: A receiving host may be running more than one network application process. Thus, when a receiving host receives a packet, it must decide to which of its ongoing processes it is to pass the packet's payload. More precisely, when a transport-layer protocol in a host receives a segment from the network layer, it must decide to which socket it is to pass the segment's payload. The mechanism of passing the payload to the appropriate socket is called *demultiplexing*. At the source host, the job of gathering data chunks from different sockets, adding header information (for demultiplexing at the receiver), and passing the resulting segments to the network layer is called multiplexing.

Connectionless and Connection-Oriented Demultiplexing: Every UDP and TCP segment has a field for a source port number and another field for a destination port number. Both UDP (connectionless) and TCP (connection-oriented) use the values in these fields – called port numbers - to perform the multiplexing and demultiplexing functions. However, UDP and TCP have important but subtle differences in how they do multiplexing and demultiplexing. In UDP, each UDP socket is assigned a port number, and when a segment arrives to a host, the transport layer examines the destination port number in the segment and directs the segment to the corresponding socket. On the other hand, a TCP socket is identified by the four-tuple: (source IP address, source port number, destination IP address, destination port number). When a TCP segment arrives from the network to a host, the host uses all four values to direct (demultiplex) the segment to the appropriate socket.

UDP: The Internet (and more generally TCP networks) makes available two transport-layer protocols to applications: UDP and TCP. UDP is a no-frills, bare-bones protocol, allowing the application to talk almost directly with the network layer. The only services that UDP provides (beyond IP) is multiplexing/demultiplexing and some light error checking. The UDP segment has only four header fields: source port number, destination port number, length of the segment, and checksum. An application may choose UDP for a transport protocol for one or more of the following reasons: it offers finer application control of what data is sent in a segment and when; it has no connection establishment; it has no connection state at servers; and it has less packet header overhead than TCP. DNS is an example of an application protocol which uses DNS.

DNS sends its queries and answers within UDP segments, without any connection establishment between communicating entities.

Reliable Data Transfer: Recall that, in the Internet, when the transport layer in the source host passes a segment to the network layer, the network layer does not guarantee it will deliver the segment to the transport layer in the destination host. The segment could get lost and never arrive at the destination. For this reason, the network layer is said to provide *unreliable data transfer*. A transport-layer protocol may nevertheless be able to guarantee process-to-process message delivery even when the underlying network layer is unreliable. When a transport-layer protocol provides such a guarantee, it is said to provide *reliable data transfer (RDT)*. The basic idea behind reliable data transfer is to have the receiver acknowledge the receipt of a packet; and to have the sender retransmit the packet if it does not receive the acknowledgement. Because packets can have bit errors as well as be lost, RDT protocols are surprisingly complicated, requiring acknowledgements, timers, checksums, sequence numbers, and acknowledgement numbers.

Pipelined Reliable Data Transfer: The textbook incrementally develops an RDT *stop-and-wait* protocol in Section 3.4. In a stop-and-wait protocol, the source sends one packet at a time, only sending a new packet once it has received an acknowledgment for the previous packet. Such a protocol has very poor throughput performance, particularly if either the transmission rate, R, or the round-trip time, RTT, is large. In a pipelined protocol, the sender is allowed to send multiple packets without waiting for an acknowledgment. Pipelining requires an increased range in sequence numbers and additional buffering at sender and receiver. The textbook examined two pipelined RDT protocols in some detail: Go-Back-N (GBN) and Selective Repeat (SR). Both protocols limit the number of outstanding unacknowledged packets the sender can have in the pipeline. GBN uses cumulative acknowledgments, only acknowledging up to the first non-received packet. A single-packet error can cause GBN to retransmit a large number of packets. In SR, the receiver individually acknowledges correctly received packets. SR has better performance than GBN, but is more complicated, both at sender and receiver.

TCP: TCP is very different from UDP. Perhaps the most important difference is that TCP is reliable (that is, it employs a RDT protocol) whereas UDP isn't. Another important difference is that TCP is connection oriented. In particular, before one process can send application data to the other process, the two processes must "handshake" with each other by sending to each other (a total of) three empty TCP segments. The process initiating the TCP handshake is called the *client*. The process waiting to be hand shaken is the *server*. After the 3-packet handshake is complete, a connection is said to established and the two processes can send application data to each other. A TCP connection has a send buffer and a receive buffer. On the send side, the application sends bytes to the send buffer, and TCP grabs bytes from the send buffer to form a segment. On the receive side, TCP receives segments from the network layer, deposits the bytes in the segments in the receive buffer, and the application reads bytes from the receive buffer. TCP is a byte-stream protocol in the sense that a segment may not contain a single application-layer message. (It may contain, for example, only a portion of a message or contain multiple messages.) In order to set the timeout in its RDT protocol, TCP uses a dynamic RTT estimation algorithm.

TCP's RDT service ensures that the byte stream that a process reads out of its receive buffer is exactly the byte stream that was sent by the process at the other end of the

connection. TCP uses a pipelined RDT with cumulative acknowledgments, sequence numbers, acknowledgment numbers, a timer, and a dynamic timeout interval. Retransmissions at the sender are triggered by two different mechanisms: timer expiration and triple duplicate acknowledgments.

Flow Control: Because a connection's receive buffer can only hold a limited amount of data, there is the danger that the buffer overflows if data enters the buffer faster than it is read out of the buffer. Many transport protocols, including TCP, use flow control to prevent the occurrence of buffer overflow. The idea behind flow control is to have the receiver to tell the sender how much spare room it has in its receive buffer; and to have the sender restrict the amount of data that it puts in the pipeline to be less than the spare room. Flow control does speed matching: it matches the sender's send rate to the receiver's read rate.

Congestion control principles: Congestion has several costs. Large queuing delays occur as the packet arrival rate nears the link capacity. Unneeded retransmissions by the sender in the face of large delays cause routers to use their link bandwidth to forward unneeded copies of packets. And, when a packet is dropped along a path, the transmission capacity that was used at each of the upstream links to forward that packet (up to point at which it is dropped) is wasted.

TCP congestion control: Because the IP layer provides no explicit feedback to end systems regarding network congestion, TCP uses end-to-end congestion control rather than network-assisted congestion control. The amount of data a TCP connection can put into the pipe is restricted by the sender's congestion window. The congestion window essentially determines the send rate. Unlike the simpler GBN and SR protocols studied in Section 3.4, this window is dynamic. TCP reduces the congestion window during the occurrence of a loss event, where a loss event is either a timeout or the receipt of three duplicate acknowledgements. When loss events are not occurring, then TCP increases its congestion window. This gives rise to the sawtooth dynamics for the congestion window, as shown in Figure 3.50. The exact rules for how the loss events influence the congestion window are determined by three mechanisms: Additive Increase Multiplicative Decrease (AIMD); slow start; and fast retransmit.