# EECS563 Exam 1 Study Guide

Stephen Longofono

Fall 2017

*The information herein was assembled from lecture notes and materials provided by Dr. James Sterbenz, along with book examples from Computer Networking: A Top-Down Approach by Kurose and Ross. This document is my interpretation of the materials presented in and outside the classroom for EECS 563 at the University of Kansas. Be sure to review the references for more authoritative* [1][2][3][4]

## 1 "This will be on the exam..."

This section is comprised of things which were explicitly mentioned as exam fodder.

- What five things does the network layer do?

  The network layer handles addressing, routing, forwarding, congestion/traffic control, and signalling.

- What do the utility curves look like for each application category? What are some examples of each?

  For interactive applications, the curve is steady at 1 until about 100 ms, at which it gradually drops, leveling out toward zero as delay surpasses 1 second. For real-time and deadline applications, the curve is steady at 1 until a predefined limit, at which point it is immediately zero. For best-effort applications, the curve is at or close to 1 and begins to drop off very gradually after accumulating a long delay.

  Examples of a best-effort application is email. It makes no guarantee of arrival time (or in general) and you wouldn't really notice anyway unless the delay was very large. Interactive applications include all web browsing; when we click on a link, we expect the perception of instant feedback. A real-time or deadline application might be a clock or stock price application.

- What is the bandwidth-delay product? How is it calculated?

  The bandwidth-delay product represents the maximum amount of bits that can be in-flight at any given time over a network path. It is computed as the bandwidth of the path multiplied by the propagation rate of the physical medium, that is, some maximum rate of $N\frac{bits}{sec}$ times some distance of travel $m$, divided by some maximum rate of propagation $s\frac{m}{sec}$

- Describe the Network Cube

  The network cube has the OSI layer, plus two extra (social at 8 and MAC at 1.5) on the front face. Each has both a data plane and a control plane, except for the session and MAC layers, which are control signals only. Behind each layer is an aggregate management plane. The data planes move data among all the layers. The control plane includes all signals which facilitate the movement of data, including error control, checksums, etc. The management plane monitors the transitions and determines the appropriate control signals along the way. There are also additional overlays which may stub out or add functionality which is not present.

- Describe the Internet Hourglass

  The Internet hourglass refers to the fact that IP is the lingua franca of the protocol world. Upper layers have many different kinds of applications, using a handful of transport layer services (TCP, UDP, RTP), which feed into a single Internet Protocol (IP), which the branches out to various layer 2 protocols (ethernet, 802.11, etc). The big picture is an hourglass with IP at the center.

- What is the difference between the Internet and the Web?

  The Internet comprises all the disparate networks, their infrastructure, and their interconnections. The Web comprises all the HTTP applications which operate over the internet, and the clients and servers that connect them.

- Describe the evolution and capabilities of the various HTTP versions

  HTTP 1.0 used a non-persistent, serial request model. The next major improvement was parallel TCP connections to request many documents at the same time. After that, persistent connections were used to make multiple requests with the same TCP connection. Finally, HTTP 1.1 made both persistence and parallelism standard, and allowed for truly pipelined connections. The proposals for HTTP 2 include better security, reduced overhead in messages, and pushing of content likely to be requested.

- Describe the major components and characteristics of each application layer category

  Information access applications operate asymmetrically, with large data units. Individual bandwidth is moderately large, and aggregate bandwidth is large. The major components include a client application (web browser), a server application (say, Apache2), and a protocol (HTTP).

  Telepresence applications operate symmetrically, with smaller, synchronized data. Individual bandwidth can be large (say, for video), and in general a steady rate of data transfer is important. The major components include a peer interface application, a synchronized connection, and a protocol (often RTP).

  Distributed computing and network storage applications vary widely in their needs, as do composed applications.

- Why were early peer-to-peer applications not "true" peer-to-peer?

  Early P2P applications used content servers, a content directory server, or some combination of both. Passing through these hubs made the effective network more like a well-distributed tree. BitTorrent applications introduced the concepts of peer trackers, which were readily accesible lists of IPs that have given content. Once peers joined the tracker, they directly receive chunks from random peers, in "true" P2P fashion.

- Describe the ARQ Modes and their flow diagrams

  In stop-and-wait mode, every segment sent must be acknowledged before another is sent. A timeout is used to trigger a resend if an ACK is not received. This makes sense for very low latency connections, but quickly loses value in all other situations.

  In Go-Back-N mode, every packet is still acknowledged, so long as they arrive in order. In the case that they do not arrive in order, the last in-order ACK is resent. If a timer expires, the sender "goes back n" to the packet that was not ACKed. This was improved by using fast-retransmit, wherein duplicate ACKs are automatically re-sent, allowing a faster recovery from error. GBN is a marked improvement over stop and wait, but for high bandwidth-delay product links, the penalty for going back n and starting transmission from there is steep.

  In Selective-Repeat mode, the receiver ACKs only the packets received, and the timer is used to identify packets that were lost. These packets are then resent selectively, and the previous sequence resumed. This forces the receiver to buffer out-of-order packets and trades this space for a more robust and efficient system.

- Explain the difference between flow control and congestion control

  Flow control refers to a sender throttling its own rate to avoid overwhelming the receiving party, typically communicated by the receive window field in TCP. Congestion control refers to a sender throttling its own rate to avoid overwhelming the network. Congestion may be signalled by network-layer HW (explicit) or inferred from missed ACKs (implicit).

- Explain the AIMD Fairness Plot The AIMD Fairness Plot depicts the throughput of two shared actors on a network path. The diagonal line connecting the axes represents all the possible combinations of their throughputs, the extreme case of each being along the axes. The diagonal line down the middle represents the ideal sharing of throughput . AIMD refers to the additive-increase, multiplicative-decrease algorithm for congestion control. After an initial congestion is encountered, an actor will halve its rate of sending, pulling it toward the origin (while the other actor's throughput is held constant). Thereafter, it will use a constant increment (additive increase) to its transmission rate, pulling the curve outward in parallel to the ideal. In this way, the algorithm is stable toward the ideal, allowing a fair sharing of throughput while allowing the transport layer to probe up to an appropriate rate of fire.

## 2 On History

Throughout much of history, numerous communication networks developed and raised problems that are still relevant today. Examples of older systems include beacon fires, semaphore flags, the optical telegraph, the pony express, the electric telegraph, and so on.

- Protocols

  To communicate effectively over distance, there has to be some agreed upon way to conduct that communication. Whether it is as simple as the beacon fires of antiquity, or as complicated as the modern Internet, communication requires that all involved parties know how to start, stop, and conduct communication.

- Error Handling

  The ability to send sophisticated messages is eroded in the face of errors; modern and ancient systems alike need a way to verify that what they received is correct.

- Security

  Many communication systems had their origins in national security and warfare. Counter-intuitively, the optical telegraph was favored over the electric telegraph at the time. Even though the electric telegraph allowed for private messages, it was easy to disrupt communication by cutting the lines, and thus was not adopted for wartime communication.

- Information Density

  There are tradeoffs between the sophistication of messages and the implements of a networking system. There are physical limits to what can be transmitted over long distances, which led to the development of Morse code. Modern protocols make use of information theory to compress and encode information for the same reasons.

### 2.1 Early Networks

Early modern networks focused on relaying voice information (PTSN) and entertainment (radio & TV). Some data networks existed, but were much less pervasive and did not come along until later. Two approaches were explored in early networks: circuit switching and packet switching.

- Circuit switching is akin to being physically connected via a switchboard: once a relatively lengthy setup time is over, there is negligible latency between one end and another. Circuit switching was used by PTSN networks.
- As circuit switching became more sophisticated, it became possible to share lines via frequency division multiplexing (FDM) or time division multiplexing (TDM), wherein separate sections of frequency bands or time intervals were shared between two or more direct lines. Even with these improvements, the exclusivity of the circuits has very poor efficiency, especially for data with low information density (conversation).
- Packet switched networks trade overhead in link to link latency for multiplexing flexibility and overall throughput. Early networks experienced troubles with *elephants and mice*, where large data packets would still end up clogging links while smaller, faster messages wait. As a result receive time for any given set of messages was longer than transmission time.
- Packet switching data networks won out, and evolved in parallel in many large local and regional networks. These larger regional networks and their infrastructure laid the groundwork for tier 1 ISPs we see today. Through the 70s and 80s, packetizing was introduced, which allowed for interleaving and reduced the gap between $d_t$ and $d_r$. Pipelining also reduced overhead, but there were still bandwidth limits.
- Gradually the larger networks began to merge and interconnect, with disparate protocols coming together at large gateways. Al Gore pushes a bill to fund high bandwidth backbone connections among the largest networks, ushering in a new world order of connectivity. The internet as we know it today began as the combination of MILNET, ARPANET, CSNET, and NSFNET. The latter was Gore's favorite.

## 3 On the Application Layer

### 3.1 Motivations

- The key motivation for networking is that all we do supports the application layer. Everything interesting we want to do, make, or share will operate at this level. Bearing this in mind, we are better grounded to make decisions regarding everything else. At the end of the day, all the other "layers" we separate networks into are providing some service toward making applications work.

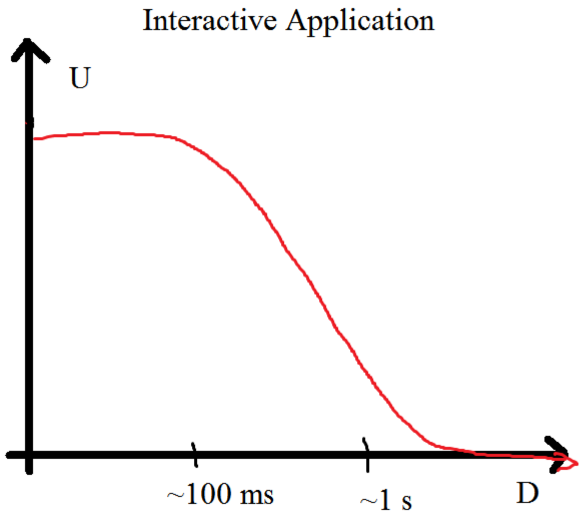


Figure 1: Utility Curve: Interactive

- The application layer and its protocols are concerned with application to application, that it, end to end coordination. Beyond the 4 basic layers below it, the application layer may also provide network-like functions when they are not specifically available (say, multicast or P2P activity).
- The key metric for all application layer entities is **delay**. All other measurements are derived from delay. One major derived metric is quality, in the context of streaming media. Delay arises from latency of physical connections, bandwidth of interconnected hubs, overhead associated with lower layers, and error.

## 3.2 Characteristics

There are a few main characteristics and categories we can use to divide the universe of applications. Each has its own actors, use cases, and general behaviors, which allow us to address the needs of each appropriately in our design of networks.

- Bandwidth needs for applications can be assessed on an individual and aggregate level. Consider the needs of an individual web browser running today's horribly inefficient "responsive" web scripts. We expect individual use to be on the order of 1 Gb/s per user. This allows us to select appropriate network and link layer hardware, and purchase the appropriate bandwidth for the expected number of users.
- Aggregate bandwidth is typically more important at a regional level. Networks carrying streaming video and HDTV collectively need huge amounts of bandwidth and throughput. Likewise, large institutional networks may see small individual bandwidth, but collectively put huge strain on a particular link of a larger network.
- Loss tolerance is an important characteristic in prioritizing and handling application data. Applications such as file transfer or email should be delivered in perfect condition; any loss or corruption is unacceptable. On the other side of the coin, a streaming video chat can tolerate dropped frames or lower quality to some degree. Understanding and addressing the loss tolerance is important to appropriate design.

## 3.3 Utility

- Utility curves are used to describe how useful an application is under duress of poor performance. In general, the abscissa is the metric (delay) and the utility is the ordinate axis.
- Focusing on delay, we have four main categories of application.
    1. best effort - Relatively insensitive to delay. The network gets it there sooner or later, and utility falls off slowly. For something like email or social networks, a few hours is not a big deal, and you might not even notice until it is delayed a day or more.

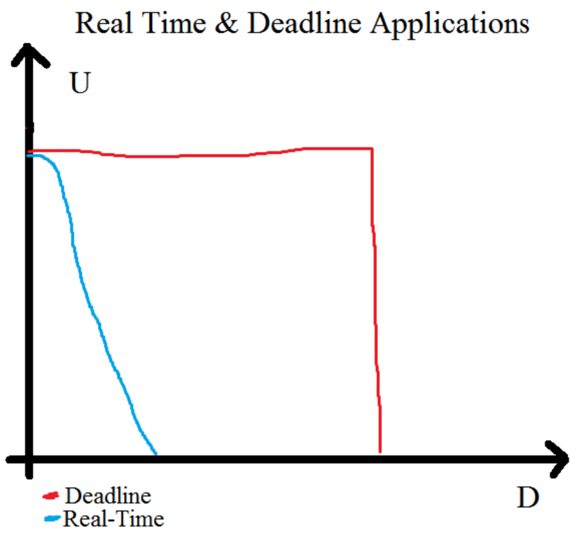


Figure 2: Utility Curve: Real Time & Deadline

2. interactive - Interactive applications rely on the immediate response associated with some network traffic. Web browsing and the like are increasingly "useless" if the response time is longer than 1 second. The borderline of our perception is 100ms, which is where the utility curve begins to fall off.
3. real-time - Real time applications require consistent, near-instant response. Any deviation from zero makes it worthless. The instruments and controls on an aircraft or a hadron collider network come to mind.
4. deadline - Applications with deadlines are perfectly useful until a critical point (the deadline) after which they are useless. An example of this is an online clock or stock price - after a certain amount of time, the information delivered is useless (since it has changed)

- Considerations of utility dictate the appropriate transport layer services, required bandwidth, and QoS.

## 3.4 Categories of Applications

As noted above, broad categories encompass most applications we encounter on a day-to-day basis. Each has its own needs in terms of services, bandwidth, utility, and use case.

- Information Access

  Information access applications follow a request and response use case. Typically, a client host initiates a connection with a remote host, and requests information.

  Response time is the most important metric: how long from the last part of the request transmission to the first part of the response with information. Interactive applications shoot for 100ms response, up to 1 second.

  On individual and aggregate levels, bandwidth is high. In general, the data is highly asymmetric, with the client requesting most if not all information.

  The components of information access include a client (say, a web browser), a server (say, a web server like Apache), and a protocol (say HTTP).

- Telepresence

  Telepresence applications are concerned with exchanging a virtual presence among peers. Things like VOIP and video conferencing are good examples.

  Data passed by telepresence applications has embedded synchronization, such that a certain rate of new data is important for useful operation. Since this extends in both directions, the bandwidth is typically relatively symmetric.

  The components of a telepresence system are often two peers (say, people in Google Hangouts) a synchronous connection (their video feeds), ad a protocol (often UDP).

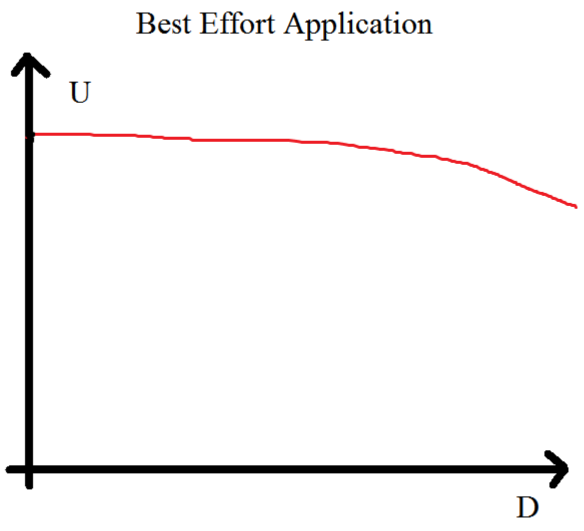


Figure 3: Utility Curve: Best Effort

- Distributed Computing & Network Storage

  Distributed processing refers to a distribution of computation, interaction, or service over a network beyond the LAN. Networked storage includes NAS over LAN, WAN and SAN systems.

  In all cases, an arbitrary exchange of control, state, and data is possible, with no one behavior fitting all. Symmetry and access vary wildly.

- Composed Applications

  Composed applications are a catch-all for everything else. Most applications can be shoehorned into some combination of the above three, making them a nice neat box for use in your favorite ivory tower.

  Any sufficiently complex application fits the bill: social networks, p2p file sharing, distance learning, and the Google office suite.

Table 1: Application Characteristics

| | **Category** | | |
|---|---|---|---|
| | **Info. Access** | **Telepresence** | **Dist.& NAS** |
| **Application Relationship** | Client/Server | P2P | Varies |
| **Bandwidth Symmetry** | Asymmetric | Symmetric | Symmetric |
| **E2E Synchronization** | None | Real-Time | Varies |

## 3.5 Case Study: Web & HTTP

The Web is a classic example of an information access application. It follows the expected clinet-server model, with high asymmetry and large granularity of data.

The key protocol on which all else rests is HTTP, which is used to shuttle around HTML, media, documents, and scripts. There are all sorts of Web applications built around this, and an every growing list of services, software, and media formats to interact with.

HTTP in its most simple form is a markdown describing the structure of a document and the objects and hypertext links which may be embedded within in. The typically client-server interaction has a client's browser request (GET) a document from a URL (uniform resource locator) at a known IP address. The latter step is facilitated by DNS. The server returns a response with or without the document, all over TCP connections.

The first generation (HTTP 1.0) was not persistent; Each document object required its own TCP connection. This required two round trips per object, plus setup time.

Next came parallel HTTP, wherein multiple connections are opened at once. This reduced overall throughput, but still incurred high overhead for complication pages with many objects.

The followup development was persistent HTTP, in which a single TCP connection is used within a timeout for multiple objects.

Finally, HTTP (1.1) made pipelined and persistent connections the default. Working with the browser, servers improve perceived performance by pushing documents linked on given pages and preloading image content.

Key parts of HTTP:

- Request Line - The method (GET/POST/HEAD/PUT/DELETE..), the URL, the version of HTTP.
- Header lines - Optional and variable length fields including encoding, language, caching info., etc.
- Body - Markdown, scripts, links, or media.

Web traffic is a significant fraction of network traffic overall, and the ever growing flow of data prompted a need for caching. Web caches store regional, timestamped copies of requested data and intercept outgoing requests to the upstream Web. If a request can be filled by a local cache with relatively low latency, the traffic is greatly reduced for upstream networks and performance improves overall. Caching addresses bottlenecking at many different levels with varied architectures. Caching exists at the browser, institutional, and ISP level. Large content providers also cache content in CDNs, which store commonly accessed contents in data centers across the country.

## 3.6 Case Study: Email & SMTP

Telepresence applications focus on providing interaction among peers, whether directly or through a common server/message broker. Interaction and bandwidth varies, and in general both become more important as communication becomes more sophisticated (email ¡ chat ¡ voice ¡ video conference).

- Components include client software, which provides a user interface to the system, and a client protocol, which dictates the system itself.
- The type of telepresence determines the utility curve; email is a classic example of best effort, while video chat is close to real-time.
- Likewise, the type determines loss tolerance; email messages cannot be dropped or corrupted, but video frames can.

Email developed as a purely P2P interaction, using SMTP. Email applications aided you in expressly defining the hops to the other endpoint, and as long as they were available, you could send a message.

This presents obvious deficiencies wrt logistics and reliability. The second generation added always-on SMTP ”servers” which hold messages until User Agents are available to receive them.

We quote servers because the SMTP server was actually just another application which actually set up transport layer connections and accepted incoming connections. This is the kind of pedantic distinction which professor might deduct points for to inflate their sense of self worth.

The next generation used institutional actual servers running POP or IMAP, which stay on to receive emails for their members. User Agents still have SMTP ”servers” to send messages, and use encrypted connections to retrieve their inbound messages from POP/IMAP servers.

Modern email is often accessed using a Web-based email client, which acts as the user interface and the SMTP ”server” in one, along with the connection to the POP/IMAP server. Mobile mail apps are more akin to the traditional setup described above.

## 3.7 Case Study: P2P File Sharing & BitTorrent

File sharing has been around for decades, but P2P file sharing has trended for the past 15 years. Today, P2P traffic is a significant fraction of bandwidth.

- The first generation used a centralized model, where all known content was indexed in a central database. Users contact the server to request a file, and the server returns the locations of peers who have the file. Thereafter, peers interact directly to transfer the file. This approach is simple, but made it easy for corporations to locate and penalize the content servers. Also, the centralized server acts as a bottleneck for all network members.

- The first generation has low communication overhead: there are two interactions with each requesting client from the directory server, and one between clients. However, the state overhead is n, since the content directory must maintain the state of all n client nodes to be accurate.
- The second generation adopts a decentralized model, wherein users join a network, and requests are sent to all active nodes (flooding). Clients which have the requested content can then answer and await connection requests.
- The second generation has low node state, as each node only has to maintain its own finite set of content. However, the complexity of communication is very high; if all n clients are active, they each send n messages over h hops, resulting in $O(hn^2)$ complexity.
- The third generation was a hybrid model, where nodes with high connectivity were used as content servers, eliminating the need for flooding along with single points of failure. This still exposed the content nodes (super/ultrapeers) to legal action.
- Modern P2P is closer to a true P2P application. Trackers are created for content: a list of how content is broken into chunks and who has what chunks (where they are). On accessing a torrent file, the tracker randomly selects a peer for each chunk, and returns the IP of the peer with the content chunk.

In all cases, the components are a peer interface. There is no set protocol, because different p2p applications use nonstandard means of achieving their ends. The closest thing to a protocol is the bit torrent P2P model with trackers and magnet links.

## 4 On the Transport Layer

The transport layer, layer 4, provides the application layer with a means of transporting data from end to end. We need such a layer because real networks have delay, error, and bandwidth problems that need to be addressed.

### 4.1 Transport Layer Services

At the most basic level, TPDUs are wrapped around ADUs, which are multiplexed into the network, sent to the destination, demultiplexed, and restored to an ADU. Additional optional services are reliability (error checking, correction, retransmits) and flow control.

### 4.2 Transport Layer Interfaces

The transport layer is used by the application layer, and it in turn makes use of the network layer and link layer. The transport layer is considered an E2E analogue of the HBH link layer. At the interface of the network layer, the TPDU is wrapper in an IP frame, and its path is determined through the network. At the link layer interface, the IP frame is wrapped in a link layer PDU, which picks the appropriate network hop to send it over.

### 4.3 End to End Arguments

The end-to-end arguments refer to a paper (Saltzer, Reed, Clark 1981) discussing best practices for network implementation. In short, the paper can be distilled into two key ideas: some functions can only be properly implemented from end to end, and some functions are worth duplicating hop by hop.

- Some functionality can only be correctly implemented by endpoint applications. The standard example is encrypted communication. In general, allowing intermediate nodes to decrypt, check, and re-encrypt a message defeats the purpose of encryption. The security of the message is compromised, so the whole E2E functionality is compromised.
- The second part is an argument for duplicating functionality for an overall performance increase. The canonical example is reliable communication. It is true that a reliable transport can only be correct if the final endpoint confirms that it received the TPDU intact. However, the topology of the network might make this E2E behavior inefficient or impossible. Consider a 3-hop path, where the first and third hops are over 100m in a LAN, and the middle hop is over 15000 km of fiber in a regional WAN. Say that on average, the RTT is 250 ms. If reliability is implmented *only* end to end, if a packet fails to make the first hop, the sender will not know until

at least 250ms have passed; The sender has to assume that it will not get a response for at least 250ms since that is the RTT. By the same token, the receiving side might see its acknowledgement fail, and again have to wait at least 250ms before it makes sense to send it again.

- If reliability is duplicated on a hop by hop basis, a failure at the first or final hop can be resolved right then and there over the shorter link between two level 2 nodes. The RTT is reduced overall, since failure can be detected earlier. That is, we move farther along the path, in fact, as far as possible, in our error checking. In practice, the performance increase is so substantial that someone wrote a paper on it, and we now learn it in this class.

## 4.4 End to End Functions

The key E2E functions provided by the transport layer are framing of upper level PDUs, multiplexing of PDUs, reliable delivery, error control, and flow/congestion control. Note that not all of the above are implemented in all transport layer protocols.

### 4.4.1 Framing & Multiplexing

TL PDUs include a header indicating the protocol and relevant data fields for that protocol. At a minimum, it will include the source IP, destination IP, and ports for each.

TL multiplexing and demultiplexing is performed using tuples constructed from the IP addresses and ports.

### 4.4.2 Transfer Modes

There are several different transfer modes available to service the many different kinds of applications.

- datagram - Each PDU has a complete header, no need to maintain connection state. This is used by UDP and allows for low overhead TL service
- connection - Each PDU has connection information to aid in mulitplexing. Three-way handshake is used to ensure both parties are able to send and receive. Explicit closing statements for a connection. Sequencing information often included.
- stream - After connecting, client makes an explicit request, and server pushes back data. Synchronization and sequence are embedded or handled externally.
- transaction - May or may not use a connection. Requests yield responses, with optional closing messages.

### 4.4.3 Error Handling

There are many ways communication can fail in a network, and the transport layer is often tasked with providing reliability. Main error types include bit errors and packet errors.

- Bit errors refer to some corruption of the bits within a packet. These errors arise from flaky hardware, interference, amplification, and other physical phenomena. Bit errors are difficult to prevent, so we concern ourselves with identifying them using checksums.
- Packet errors
  - Packet loss - packets never reach their destination
  - fragment loss - a piece of a larger file never arrives, and renders the rest of the fragments useless
  - burst loss - a link begins dropping packets, and a large burst of in-flight packets behind it are dropped
  - packet misordering - due to congestion, error, or routing, packets arrive out of order
  - packet insertion - undetectable packet header error, say from sequence number rolling over. Essentially an unexpected packet.
  - packet duplication - duplicate packets arrive

Management of errors varies greatly, but most follow some form of ARQ (Automatic Repeat reQuest). ARQ uses acknowledgements or negative acknowledgements to indicate that a packet should be resent.

- Stop-and-wait - This is the simplest approach: for every packet sent, wait for a positive ACK. Incurs a RTT for each packet, which can be a serious penalty over long distances. A timeout is used to determine when to resend; too short yields duplicate packets, too long is inefficient.
- Go-Back-N - Packets are sent in a pipelined manner, and the receiver ACKS them as they arrive in a sequential manner. If a packet arrives out of order, the last in-order ACK is re-sent to the sender. This allows the sender to backtrack to the missed packet and start over. This is a good improvement over stop and wait, but if the bandwidth-delay product is high, it has the potential to lose massive amounts of packets. Since the receiver stops saving packets once one arrives out of order, many packets are sent multiple times.
- Optimized GBN - Same as above, but adds a mechanism to quickly detect errors before a timeout occurs. In the event of n (say, 3) duplicate ACKs, the sender assumes a packet loss, and starts over. This improves the recovery time for error, but still incurs high loss if the BW-delay product is high (which is nearly always the case in modern networks)
- Selective Repeat - Improves on GBN by requiring the server to buffer out-of-order packets. The server ACKs packets as they arrive, and the sender relies on the timeout to identify loss. On detecting loss, the lost packet is resent, but then sending resumes at the sequence from before the error detection. This allows only the missing packets to be resent, at the expense of buffer space and more sophisticated server programming.

### 4.4.4 Flow Control

Flow control refers to the transport layer proactively throttling its transmission rate to avoid overwhelming the receiving party. This end to end control is distinct from congestion control, which refers to throttling transmission rate to avoid overwhelming the network.

Flow control is commonly implemented via a window, whose size determines the maximum amount of unacknowledged packets in flight at any given time. The size of the window is often set to the receiver's current buffer space as a part of the handshake, and thereafter is adjusted as ACKs are sent between sender and receiver. Flow control relies on explicit information from the receiving party to maintaing a dynamic window on the sending side.

### 4.4.5 Congestion Control

Congestion control refers to the transport layer proactively throttling its transmission rate to avoid overwhelming the network. This single-ended control is distinct from flow control, which refers to throttling transmission rate to avoid overwhelming the receiver.

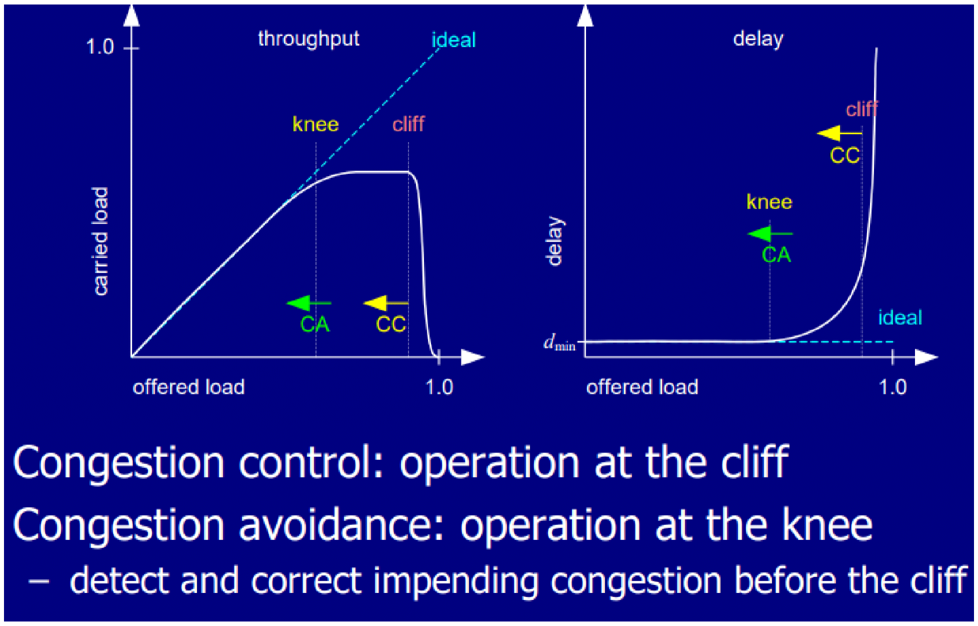
Congestion control is often implemented with implicit information (detection of loss), but can be implemented by explicit signalling from the network layer. Here, we focus on the former.

Congestion control is implemented by monitoring transmission for lost packets - any lost packet is interpreted as congestion. Note that for lossy networks (wireless and cell phone), this is a bad assumption. The sending party maintains a congestion window to limit the amount of packets in flight. Starting with a modest value, the window is slowly increased until congestion is detected, at which point it is scaled back. This is referred to as network probing.

*Aside: Congestion control versus congestion avoidance - Considering the load offered by the receiver (throughput), and the load pushed by the sender, we might expect a one-to-one mapping up to a certain point. See Figure 4 for a visual representation. Once capacity is reached, no additional pushing will be able to change the offered load. Even worse, once congestion begins, retransmissions cause the amount of packets in-flight to increase, and the offered load is reduced further until it collapses. It is desirable to avoid the "cliff" completely, and operate at the linear region of a throughput curve. Considering the same situation with respect to the network delay as a function of load, we expect an exponential relationship. As congestion begins, the delay increases unbounded, and goes to infinity long before capacity and the "cliff" is reached. Given the massive increase in delay once congestion begins, it is desirable to avoid the region of congestion altogether. This behavior is realized using the congestion window.*

In practice, TCP operates via an Additive Increase, Multiplicative Decrease policy. At the start of a connection, the congestion window is set to one. Assuming the server does not indicate congestion (either by receive window or failure to ACK), the congestion window size is increased for every ACK received. Effectively, this doubles the amount of packets sent at each transmission. This is referred to as the "slow start" period.

Once congestion is detected, AIMD is employed; the congestion window is reduced by say, $\frac{1}{2}$, and thereafter incremented linearly. The idea is that once congestion is reached, a threshold should be set somewhere below the current rate, the rate should be reduced by a factor, and we should increase more conservatively whenever the congestion window is higher than that threshold.



Congestion control: operation at the cliff

Congestion avoidance: operation at the knee

– detect and correct impending congestion before the cliff

Figure 4: Congestion: Throughput & Delay

### 4.4.6 On Fairness

One of the services provided by the transport layer is fair use of network resources among applications. The above congestion algorithm has a built-in trend toward ideal sharing of resources. Consider the graph in Figure 5, which depicts the throughput allotted to two transmitting applications in AIMD mode. It is apparent that each reaches its maximum throughput whenever the other is not transmitting, and along the line $y = x$, the throughput is perfectly shared. Consider the effects of the additive increase and multiplicative decrease. As either party increments its congestion window, the throughput pulls the line outward, in a direction parallel to the ideal. As congestion occurs, throughput is reduced dramatically, pulling the line toward the origin. The net effect is that the system is stable at the ideal, and deviations from the ideal will tend to drift back toward perfect sharing. Note that we have omitted quality of service in this model; each actor is given an equal share of capacity.

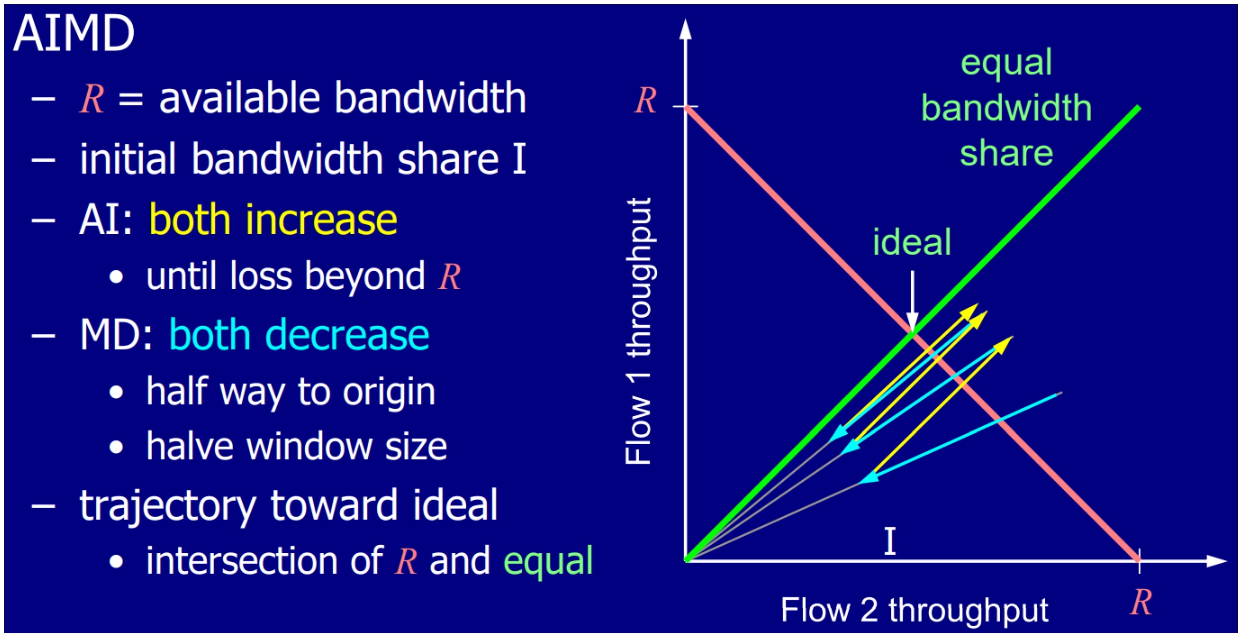
## 4.5 Transport Layer Protocols

There are a dozen or so transport layer protocols that have been developed over the years, but the four most common are UDP, TCP, RTP, and RDP. We study the first two in detail here, with a mention of RTP for completeness' sake.

### 4.5.1 UDP

The User Datagram Protocol is implemented as a barebones, simple protocol for packet transport. UDP is employed by streaming applications, loss-tolerant applications, and applications which need only small, one-time messages to be transported.

- No connection setup - each UDP segment is handled independently. There is no setup penalty, and no connection state to maintain - all you need is a socket to be listening when you send a segment.
- No reliability - UDP has no ACKS
- No controls - UDP does not support congestion or flow control
- Minimal mux - UDP uses only the destination port to demux.
- Low overhead - UDP headers include at minimum only the ports, checksum, and the total segment length, weighing in at 2 bytes each for a total of 8 bytes.
- Minimal error correction - weak checksum is performed as the 1's complement of the sum of checksums. They are added, the carry added to the result, and the 1's complement taken



AIMD

- $R$ = available bandwidth
- initial bandwidth share I
- AI: both increase
  - until loss beyond $R$
- MD: both decrease
  - half way to origin
  - halve window size
- trajectory toward ideal
  - intersection of $R$ and equal

Figure 5: Fairness of Throughput

#### 4.5.2 RTP

RTP builds on top of UDP to add in support for real-time applications. This includes timestamps, sequencing numbers, and source identifiers. All of the above extend the header length, otherwise operation, benefits, and drawbacks are as with UDP. In applications where late delivery is more wrong than failed delivery, UDP is the better choice.

#### 4.5.3 TCP

TCP offers reliable transport, flow control, and congestion control. TCP is used by applications which are not fault tolerant, and there are a huge variety of options to customize the header to your needs.

- Connection oriented - 3-way handshake adds overhead to ensure that both parties can perform duplex communication. Connection state is maintained by both sender and receiver
- Flow control - TCP implements the flow control mechanisms described above.
- Congestion control - TCP attempts to avoid congestion by the mechanisms described above.
- Headers are longer - at minimum, includes ports, sequence number, ack number, length, flags, checksum, and receive window. Header length field is included to allow variable length optional fields in the header. Minimum 20 bytes for required fields and no payload.
- Flags are used to designate specific types of packets (SYN, ACK, FIN, or combinations)
- Demuxing - Uses the 5-tuple (source port, source address, dest. port, dest. address, and protocol number). The IP addresses and protocol number are in the network layer PDU.

TCP makes use of a moving average to determine the appropriate timeout for acknowledgements. As noted above, any of stop-and-wait, go-back-n, or selective-repeat may be used, but all require a timeout to be set. TCP initializes its timeout to slightly longer than the RTT of the handshake. The optional timestamp fields are almost always included for this reason. More recent observations of RTT are given more weight, so that the timers can adapt to network conditions.

For example, assuming an initial RTT of 5 ms, and an observed RTT of 10 ms for the next ACK, the next timer would be computed by: $\text{RTT}_{\text{next}} = 0.005 \cdot \alpha + 0.010 \cdot \beta$, where $\alpha$ represents the weight of previous estimates, and $\beta$ represents the weight of new observations.

## 5 Example Questions

### Chapter 1

1. What is the difference between a host and an end system? List several different types of end systems. Is a web server an end system?

   There is no difference between a host system and an end system. End systems might be a laptop, a mobile phone, a desktop computer, or a commercial server. Yes, a web server is an end system.

2. Suppose a user shares a 2 Mbps link. Also suppose each user transmits continuously at 1 Mbps when transmitting, but each user transmits only 20 percent of the time. (See the discussion of statistical multiplexing in Section 1.3.) When circuit switching is used, how many users can be supported?

   Circuit switching networks often interleave the data of individual endpoints, ie the first quarter from node 1, the second quarter from node 2, and so on. In this case, 2 users are sharing a 2Mbps line, and when active, each needs bandwidth of 1 Mbps. Thus in the ideal case, we say we can support $\frac{2Mbps}{1Mbps} = 2$ users.

   For the remainder of this problem, suppose packet switching is used. Why will there be essentially no queueing delay before the link if two or fewer user transmit at the same time? Why will there be a queueing delay if three users transmit at the same time?

   For a packet switched network, queueing delay occurs when a router is receiving packets faster than it can transmit them. For the given network, assuming that there is no overhead in reading and directing the packets, the routers can transmit at 2 Mbps. Thus while there are two or less users transmitting packets at a rate of 1 Mbps, the rate of packets incoming will never exceed the rate of outgoing packets.

   Find the probability that a given user is transmitting.

   It is given that any user is transmitting on average only 20 percent of the time. So the probability that any given user is transmitting is 0.2.

   Suppose now there are three users. Find the probability that at any given time, all three users are transmitting simultaneously. Find the fraction of time during which the queue grows.

   The probability that three users are all transmitting simultaneously is the product of the individual probabilities. So $0.2^3 = 0.008$. The fraction of time during which the queue grows will be 0.008 of every second that the network is live.

3. Consider sending a packet from a source host to a destination host over a fixed route. List the delay components in the end-to-end delay. Which of these delays are constant and which are variable?

   The end-to-end delay includes transmission delay, queueing delay, propagation delay, and processing delay. Propagation delay and transmission delay are functions of the physical layer and layer 2 and 3 equipment, so they are constant. Queueing delay and processing delay depend on the network load, so they are variable.

4. How long does is take a packet of length 1,000 bytes to propagate over a link of distance 2,500 km, propagation speed $2.5 \cdot 10^8$ m/s, and a transmission rate 2 Mbps? More generally, how long does it take a packet of length L to propagate over a link of distance d, propagation speed s, and transmission rate R bps? Does this delay depend on packet length? Does this delay depend on transmission rate?

   Using the given information, we can calculate the propagation delay over the line:

   $d_{prop} = \frac{distance}{rate} = \frac{2.5 \cdot 10^6 [m]}{[2.5 \cdot 10^8 \frac{m}{sec}]} = 0.01 sec$

   This is how long it takes a single bit to traverse the line, under ideal conditions.

   The key word here is *propagate*. If we wanted to know how long it took to send and have received the packet in full, then transmission rate becomes important, and only the combination of the two will yield the correct answer. However, since here we are only concerned with the propagation, we assume that the packet has finished propagating when its first bit reaches the destination. Thus our propagation time is equal to the propagation delay of 0.01 seconds.

   In general, this delay is expressed as $\frac{d}{s}$, and is independent of packet length $L$ and transmission rate $R$.

5. Suppose Host A wants to send a large file to Host B. The path from Host A to Host B has three links, of rates R1 = 500 kbps, R2 = 2 Mbps, and R3 = 1 Mbps. Assuming no other traffic in the network, what is the throughput for the file transfer?

The throughput is bottlenecked by the slowest link, in this case $R1$ at 500 kbps.

Suppose the file is 4 million bytes. Dividing the file size by the throughput, roughly how long will it take to transfer the file to Host B?

$\frac{32\cdot 10^6[bits]}{5\cdot 10^5[\frac{bits}{sec}]} = 64$ seconds.

If R2 is reduced to 100 kbps, what is the new throughput and time to transfer a 4 million byte file?

The throughput is now bottlenecked by $R2$ at 100 kbps. The file now takes:

$\frac{32\cdot 10^6[bits]}{1\cdot 10^5[\frac{bits}{sec}]} = 320$ seconds.

6. What are the five layers in the Internet protocol stack? What are the principal responsibilities of each of these layers?

   The five layers are the physical, link, network, transport, and application layers. The physical layer is made up of the physical medium over which information is sent, and the hardware which encodes, decodes, amplifies, and otherwise facilitates the transfer of information. The link layer is concerned with ensuring that LPDUs were successfully sent or received at individual nodes in the network (e.g switches). The network layer is concerned with identifying the destination for NPDUs and selecting the path over the link layer to take. The transport layer is concerned with transporting TPDUs, including error checking, load control, optional reliability, from one application to another . The application layer is concerned with user-facing programs with some need to generate or receive data from other application-layer programs. All of the above have their own set of protocols and their own data units called PDUs.

7. Which layers in the Internet protocol stack does a router process? Which layers does a link-layer switch process? Which layers does a host process?

   Routers generally handle from layer 3 downward, link-layer switches from level 2 downward. Hosts typically process all 5 layers in one way or another, although they delegate lower-layer tasks to local lower-layer hardware.

8. Review the car-caravan analogy of propagation delay and transmission delay from section 1.4. Assume a propagation speed of 100 km/hour, transmission speed of 1 car every 12 seconds, and that there are 10 cars in each caravan. Suppose the caravan travels 150 km, beginning in front of one tollbooth, passing through a second tollbooth, and finishing just after a third tollbooth. What is the end-to-end delay?

   Assuming equally spaced toll booths, the distance between each one is half the total distance, or 75 km. End-to-end delay at each link is the sum of transmission delay, queueing delay, processing delay, and propagation delay across that link.

   The propagation delay is the distance divided by the propagation speed:

   $d_{prop} = \frac{75\cdot 10^3[m]}{[27.7778\frac{m}{s}]} = 2699.9978[sec] = 45[min]$

   The transmission delay is the length of the caravan divided by the rate of transmission. Setting up a proportion:

   $d_{trans} = \frac{10[cars]}{0.0833[\frac{cars}{sec}]} = 120[sec] = 2[min]$

   The analogy omits queueing and processing delay, so our total delay per link is $45 + 2 = 47[min]$ per link. There is an additional toll booth at the end, which adds an additional 2 minutes. Thus the total end-to-end delay across the two links is $2 \cdot 47[min] + 2[min] = 96$ minutes.

   Now, redo the calculation assuming that there are eight cars in the caravan instead of ten.

   $d_{trans} = \frac{8[cars]}{0.0833[\frac{cars}{sec}]} = 96[sec] = 1.6[min]$

   $d_{end-to-end} = 2 \cdot 46.6[min] + 1.6[min] = 94.8$ minutes.

9. Express the propagation delay, $d_{prop}$, in terms of distance $m$ and propagation speed $s$. Determine the transmission time of the packet, $d_{trans}$, in terms of packet length $L$ and transmission rate $R$. Ignoring processing and queuing delays, obtain an expression for the end-to-end delay.

   $d_{prop} = \frac{m}{s}[sec]$

   $d_{trans} = \frac{L}{R}[sec]$

   $d_{end-to-end} = \frac{m}{s} + \frac{L}{R}[sec]$

10. Suppose Host A begins to transmit a packet at time t = 0. At time t = dtrans, where is the last bit of the packet?

    By definition, the transmission rate is the amount of packets which can be transmitted per second. thus the last bit of the packet has just left the sending party. Note that transmission rate is also sometimes expressed as bits per second, and in that case you would need information about the packet width in bits, the distance traveled, and the propagation speed to accurately answer the question.

11. Suppose dprop is greater than dtrans. At time t = dtrans, where is the first bit of the packet?

    Since the propagation speed is higher than the transmission speed (slower rates of transfer), the first bit is still somewhere in the transmission medium. In this case, the propagation speed and distance of the link are necessary to answer exactly where the bit is.

12. Suppose dprop is less than dtrans. At time t = dtrans, where is the first bit of the packet?

    Since propagation is faster than transmission, the bit is already at the destination at time t = $d_{trans}$.

13. Suppose $s = 2.5 \cdot 10^8$, L = 120 bits, and R = 56 kbps. Find the distance m so that dprop equals dtrans.

    Setting the expressions for each delay equal, we get:

    $\frac{L}{R} = \frac{m}{s}$

    solving for $m$, and plugging in the given figures:

    $m = \frac{s \cdot L}{R} = \frac{2.5 \cdot 10^8 [\frac{m}{s}] \cdot 120[bits]}{56 \cdot 10^3 [\frac{bits}{sec}]} = 535714.2857[m]$

14. Suppose you would like to urgently deliver 40 terabytes data from Boston to Los Angeles. You have available a 100 Mbps dedicated link for data transfer. Would you prefer to transmit the data via this link or instead use FedEx overnight delivery? Explain.

    40 terabytes is $32 \cdot 10^{13}$ bits, so assuming a perfect, ideal transmission, transmitting over the dedicated link takes:

    $\frac{32 \cdot 10^{13}}{10^8} = 32 \cdot 10^5 [sec] = 53333.34[min] = 888.89[hours] = 37.04[days]$. Overnight freight is preferable.

15. Suppose two hosts, A and B, are separated by 20,000 kilometers and are connected by a direct link of R = 2 Mbps. Suppose the propagation speed over the link is 2.5 * 108 meters/sec. Find the bandwidth-delay product and interpret it.

    Bandwidth in this case is the rate $R$ of the link. The bandwidth-delay product is the bandwidth multiplied by the propagation delay:

    $d_{prop} \cdot BW = \frac{2 \cdot 10^7 [m]}{2.5 \cdot 10^8 [\frac{m}{s}} \cdot 2 \cdot 10^6 [\frac{bits}{sec}] = 16 \cdot 10^4 [bits]$

    The bandwidth-delay product represents the maximum, ideal amount of bits in the line at any given time.

# Chapter 2

1. For a communication session between a pair of processes, which process is the client and which is the server?

   In general, the client is considered the process which initiates the communication, and the other parties are servers.

2. What information is used by a process running on one host to identify a process running on another host?

   The IP address is used to identify a host, and the port number is used to identify a process on that host.

3. Suppose you wanted to do a transaction from a remote client to a server as fast as possible. Would you use UDP or TCP? Why?

   UDP is preferable, as it requires no handshake, minimal header information, and there is no flow control mechanism. It will get there as fast as it can, if it gets there.

4. Why do HTTP, FTP, SMTP, and POP3 run on top of TCP rather than on UDP?

   In general, these applications are sensitive to data loss, so they need the reliable transfer that TCP provides.

5. Describe how Web caching can reduce the delay in receiving a requested object. Will Web caching reduce the delay for all objects requested by a user or for only some of the objects? Why?

   Web caching reduces the delay directly by bringing the data physically closer to the client. By serving the data from a local (or relatively local) cache, less hops need to be made, with less overhead along the way. Indirectly, Web caching reduces traffic on connections to ISPs, IXPs, and CDNs, so that when a cache does not have the requested object, there is more bandwidth available to send the object. For this reason, delay is reduced on all objects.

6. Why is it said that FTP sends control information "out-of-band"?

   FTP uses a separate connection to send information regarding controls.

7. In BitTorrent, suppose Alice provides chunks to Bob throughout a 30-second interval. Will Bob necessarily return the favor and provide chunks to Alice in this same interval? Why or why not?

   No. In general, Bob may or may not have Alice on is list of the top four peers (with respect to upload speed). If Alice happened to beat out one of Bob's top four, then she would begin receiving chunks from Bob. However, this is not necessarily the case.

8. Consider a new peer Alice that joins BitTorrent without possessing any chunks. Without any chunks, she cannot become a top-four uploader for any of the other peers, since she has nothing to upload. How then will Alice get her first chunk?

   BitTorrent also connects with one random peer for each chunk, which allows new peers to enter the pool, so to speak. This is called "optimistic Unchoked" connecting.

9. What is an overlay network? Does it include routers? What are the edges in the overlay network?

   An overlay network is an abstraction of a network that is managed at the application layer. It does not include the routers, or in general any of the components of the actual network below it. The edges represent the connection to other nodes (peers) in the overlay network, although they need not resemble the physical or logical path of the lower network layers.

10. In Section 2.7, the UDP server described needed only one socket, whereas the TCP server needed two sockets. Why? If the TCP server were to support n simultaneous connections, each from a different client host, how many sockets would the TCP server need?

    The TCP server uses one socket with a public IP to accept incoming connections. It then generates a new socket for that specific connection and resumes listening. UDP needs only a single socket, since it does not maintain connection state. A TCP server supporting n simultaneous connections will have n sockets for the n connections, plus one more as a "welcoming" socket.

11. For the client-server application over TCP described in Section 2.7, why must the server program be executed before the client program? For the client server application over UDP, why may the client program be executed before the server program?

    The server program must be run first because it must accept and participate in the 3-way handshake for the incoming connection. UDP doesn't need the connection, so it MAY still work if the client starts first, so long as the server is written correctly.

12. T/F. A user requests a Web page that consists of some text and three images. For this page, the client will send one request message and receive four response messages.

    False. The client will send one request message, and it will receive at least four response messages, but likely many more. The text itself may exceed four messages, and images usually span many messages.

13. T/F. Two distinct Web pages (for example, www.mit.edu/research.html and www.mit.edu/students.html) can be sent over the same persistent connection.

    True. In this case, the host is the same, so multiple documents (Web pages) can be sent over any of the connections set up with the remote host.

14. T/F. With non-persistent connections between browser and origin server, it is possible for a single TCP segment to carry two distinct HTTP request messages.

    False. Non-persistent implies one object per connection. A single request will be sent for the object, and after the object is received, the connection is closed.

15. T/F. The Date: header in the HTTP response message indicates when the object in the response was last modified.

    False. The Date: field has the date. The last modified date goes in the Last Modified: field.

16. T/F. HTTP response messages never have an empty message body.

    False. HEAD requests exist to request HTTP responses with no body.

17. Consider an overlay network with N active peers, with each pair of peers having an active TCP connection. Additionally, suppose that the TCP connections pass through a total of M routers. How many nodes and edges are there in the corresponding overlay network?

    The routers are irrelevant. By induction, it can be shown that there will be $\frac{n(n-1)}{2}$ edges in the network.

18. Suppose Bob joins a BitTorrent torrent, but he does not want to upload any data to any other peers (so called free-riding). Bob claims that he can receive a complete copy of the file that is shared by the swarm. Is Bob's claim possible? Why or why not? Bob further claims that he can further make his "free-riding" more efficient by using a collection of multiple computers (with distinct IP addresses) in the computer lab in his department. How can he do that?

    Due to the random peer selection for each chunk ("optimistic unchoked" connection), Bob could get a file if his transfer rate was so low that he is not promoted to a top 4 peer in anyone's list. It is possible to do better with more computers, but i don't understand enough about the methods to comment.

# Chapter 3

1. Is it possible for an application to enjoy reliable transfer even when the application runs over UDP? If so, how?

   This could be accomplished by building reliability tools into the application. UDP does not provide any guarantees.

2. Suppose a process in Host C has a UDP socket with port number 6789. Suppose both Host A and Host B each send a UDP segment to Host C with destination port number 6789. Will both of these segments be directed to the same socket at Host C? If so, how will the process at Host C know that these two segments originated from two different hosts?

   Both will be directed to the port, and they will be distinguished by the information in the network layer PDU about the origin IP addresses.

3. Suppose that a Web server runs in Host C on port 80. Suppose this Web server uses persistent connections, and is currently receiving requests from two different Hosts, A and B. Are all of the requests being sent through the same socket at Host C? If they are being passed through different sockets, do both of the sockets have port 80? Discuss and explain.

   Assuming that they are using TCP (likely, given that they are Web servers on port 80), they are being received on port 80, but are actually being passed through different sockets. HTTP over TCP has its "welcoming socket" on 80, and after a connection is established, information in the TCP segments is used to demultiplex the data degments to other ports. These other sockets are listening to the other ports, and as far as the client is concerned, it is still communicating on port 80.

4. (T or F) Host A is sending Host B a large file over a TCP connection. Assume Host B has no data to send to Host A. Host B will not send acknowledgements to Host A, because Host B cannot piggyback the acknowledgements on data.

   False. TCP connections support reliable connections, which requires an ACK message to be sent for each received message.

5. (T or F) The size of the TCP rwnd never change throughout the duration of the connection

   False. The receive window changes dynamically based on how full the receiver's inbound queue is.

6. (T or F) Suppose Host A is sending Host B a large file over a TCP connection. The number of unacknowledged bytes that A sends cannot exceed the size of the receive buffer.

   True. The receive window is present to communicate the available space to the sender.

7. (T or F) Suppose Host A is sending a large file to host B over a TCP connection. If the sequence number for a segment of this connection is m, then the sequence number for the subsequent segment will necessarily be m+1

   False. The sequence number is a relative byte sequence. The subsequent sequence number will be the previous sequence number plus the amount of data bytes in the last segment.

8. (T or F) The TCP segment has a field in its header for rwnd

   True.

9. (T or F) Suppose that the last SampleRTT in a TCP connection is equal to 1 sec. The current value of TimeoutInterval for the connection will necessarily be ¿= 1 sec

   False. An exponentially weighted moving average is used to set the timeout. While more weight will be placed on the newest sample, it will not necessarily be greater than the value of the sample.

10. (T or F) Suppose Host A sends one segment with sequence number 38 and 4 bytes of data over a TCP connection to Host B. In this same segment the acknowledgement number is necessarily 42.

    False. The acknowledgement number is used by the receiver to communicate what sequence number it expects next. The value has no meaning in a sender's message.

11. Suppose Host A sends Two TCP segments back to back to Host B over a TCP connection. The first segment has sequence number 90; the second has sequence number 110. How much data is in the first segment?

    20 bytes. Since 90 is the byte sequence of the message, and the following message reports 110, then the previous message had $110 - 90 = 20$ bytes of data.

12. Suppose that the first segment is lost but the second segment arrives at B. In the acknowledgment that Host B sends to Host A, what will be the acknowledgement number?

    B will send the acknowledgement number for the segment it was expecting, 90. This is how loss is detected in TCP connections.

13. (T or F) Consider Congestion control in TCP. When the timer expires at the sender, the value of *ssthresh* is set to one half of its previous version.

    False. The *ssthresh* is set to one half of the current congestion window size.

14. Suppose that the UDP receiver computes the internet checksum for the received UDP segment and finds it matches the value carried in the checksum field. Can the receiver be absolutely certain that no bit errors have occurred? Explain.

    No. TCP and UDP use a weak checksum, which has many opportunities for false positives. Unless there is some other means of checking integrity included in the data itself or built into the application, the receiver cannot be certain.

15. Consider a reliable data transfer protocol that uses only negative acknowledgments. Suppose the sender sends data only infrequently. Would a NACK-only protocol be preferable to a protocol that uses ACKs? Why? Now suppose the sender has a lot of data to send and the end-to-end connection experiences few losses. In this second case, would a NACK-only protocol be preferable to a protocol that uses ACKs? Why?

    No. Due to infrequent sending, a receiver may not realize it hasn't received a packet until it gets the next one with an out-of-order sequence. In the second case, there would be reduced feedback traffic, so it may be preferable.

16. Consider the cross-country example shown in Figure 3.17. How big would the window size have to be for the channel utilization to be greater than 98 percent? Suppose that the size of a packet is 1,500 bytes, including both header fields and data.

17. Consider the GBN protocol with a sender window size of 4 and a sequence number range of 1,024. Suppose that at time t, the next in-order packet that the receiver is expecting has a sequence number of k. Assume that the medium does not reorder messages. Answer the following questions: What are the possible sets of sequence numbers inside the sender's window at time t? Justify your answer. What are all possible values of the ACK field in all possible messages currently propagating back to the sender at time t? Justify your answer.

At time t, the possible sequence numbers in the sender's window could vary greatly. the first message will undoubtedly begin with k, but due to varying segment data sizes, the rest could be any range of the given values. If the sending program was written well, it would not have allowed the sequence numbers to roll over in the same window, or even come close, to allow a buffer against this very problem.

As far as the acknowledgement values, they too could vary greatly, since it is possible that k is small and the sequence has just rolled over. The text suggests that to avoid these problems, the window and its contents should never exceed half of the range of sequence values. This can be achieved by setting reasonably small data lengths and reasonably long windows.

18. Why did the TCP designers choose not to perform a fast retransmit after the first duplicate ACK for a segment is received?

    Due to the frequency of slightly out of order packets, in SR implementations it is better to wait for another duplicate ACK to be sure that there is in fact a problem. The state machine provided in the book for SR would be able to handle multiple packets arriving at once, and this might not be a problem unless several packets are arriving out of order.

19. Host B has recieved up to byte 126 from A. A then sends two segments to B back-to-back. The 1st and 2nd segment contain 80 and 40 bytes of data. the first has sequence number 127, source port # 302 and dest port # 80. Host B sends ACK when it receives a segment from A. In the second segment, what are the 3 numbers? If the first comes before the second, what is the ACK, port and dest of the first arriving segment? If the second arrives before the first segment, what is the ACK of the first arriving segment?

    The second segment from the sender would have source port 302, destination port 80, and sequence number 207.

    The first segment from the sender would have source port 302, destination port 80, and its acknowledgment number would be irrelevant. The ack sent back in response would have its ACK set to 207, the next byte in sequence it expects to receive.

    If the second segment arrived before the first, the receiver would send the ACK it had previously, 127.

## References

[1] K. Kurose J. & Ross. *Computer Networking: A Top-Down Approach, 7th Ed.* Pearson, London, 2017.

[2] J. Sterbenz. *Introduction to Communication Networks: End-to-End Transport.* URL: `https://www.ittc.ku.edu/~jpgs/courses/intronets/lecture-transport-intronets-display.pdf`.

[3] J. Sterbenz. *Introduction to Communication Networks: History and Architecture.* URL: `https://www.ittc.ku.edu/~jpgs/courses/intronets/lecture-hist-arch-intronets-display.pdf`.

[4] J. Sterbenz. *Introduction to Communication Networks: Networked Applications and Social Networking.* URL: `https://www.ittc.ku.edu/~jpgs/courses/intronets/lecture-apps-intronets-display.pdf`.