ECE 358 Course Notes

Computer Networks

Michael Socha

4A Software Engineering
University of Waterloo
Spring 2018

Contents

1 Course Overview

1.1 Logistics

- **Professor:** Albert Wasef

1.2 Topics Covered

This course focuses on the fundamentals of networking and thinking like a network engineer. Specific topics covered by this course include:

- LAN technologies and underlying protocols
- Transport protocols (TCP, retransmission)
- IP layer concepts (e.g. routing, addressing)
- Discrete-event simulation
- Network utilities

2 Introduction - Computer Networks and the Internet

2.1 What is the Internet?

The Internet is the world's largest computer network, connecting billions of devices. Devices connected to the Internet are known as hosts (end systems), and are running some kinds of network applications. Communication links are necessary for hosts to share information with each other, which can be done through a variety of means, including cables (e.g. fiber, copper), radio, or satellite. Packet switches (e.g. routers, switches) are responsible for forwarding chunks of data through these communication links.

Standardized protocols are necessary for communication between hosts. Protocols define the format and order of messages sent as well as actions taken upon message transmission and reception. Sample protocols include TCP, IP, and HTTP. These standards are maintained by the IEFT (Internet Engineering Task Force).

The Internet can also be viewed from a more service-oriented perspective, since it can be used to provide services such as the Web, VoIP, email, etc. to applications. The Internet also provides a programming interface to applications to interact with connected hosts.

2.2 Network Edges

Edge devices provide some sort of entry point to a network. Examples include computers, mobile devices, and servers (often in data centers). Communication between devices on a network can be wired or wireless.

2.2.1 Access Networks and Physical Media

End systems can connect to an edge router through various ways, including using residential access nets, institutional access networks and mobile access networks. Important considerations in such connections include a connection's bandwidth, latency, and whether it is shared/dedicated.

2.2.2 Digital Subscriber Line (DSL)

Network connections can be made through a digital subscriber line (DSL), which allows for the transmission of data over telephone lines. A digital subscriber line access multiplexer (DSLAM) can be used to connect multiple DSL lines to a digital communications channel. Downstream transmission rates (typically ¡ 1 Mbps) tend to be much faster than upstream transmission rates (typically ¡ 10 Mbps). Optimal transmission rates are rarely reached in practice. Each line connects directly to a central office.

2.2.3 Cable Network

Network connections can also be made through a cable network, which uses the same infrastructure as cable television. Differing frequencies are used to distinguish between different channels of communication. Hybrid Coaxial Cables (HFCs) are used to form the connection, which tend to have a downstream transmission around 30 Mbps and an upstream transmission around 2 Mbps. These connections attach to an ISP router, and multiple parties typically share access to a cable headend.

2.2.4 Ethernet

Most enterprise access networks use Ethernet connections, which tend to be much faster (available speeds include 10 Mbps, 100 Mbps, 1 Gbps, 10 Gbps). Nowadays, most end systems connect to an Ethernet switch.

2.2.5 Wireless Access Networks

Wireless access networks can connect end systems to routers without a cable connection. Wireless LANs (i.e. Wi-Fi) provide network access for a fairly small range, while wide-area

access networks are provided by cellular operators and have a range of 10s of kilometers. Wireless LANs tend to have higher bandwidths than wide-area access networks.

2.2.6 Data Packet Transmission

A host sending function is responsible for:

- Taking an application message
- Breaking the message into chunks (packets) or length L bits
- Transmitting packets across a network at transmission rate R

$packettransmissiondelay = \frac{L}{R}$

2.2.7 Physical Media

The medium facilitating transmission between a transmitter and receiver is called a physical link. Physical links may be guided (i.e. solid cables, such as copper, fiber or coax), or unguided (i.e. signals may propagate freely, such as through radio).

2.2.8 Coaxial Cable

Coaxial cables are formed from two concentric copper conductors. Coaxial cables bidirectional and they are are broadband, so they can support communication across multiple channels.

2.2.9 Fiber Optic Cable

Fiber optic cables feature a glass fiber carrying light pulses, where each pulse represents one bit. Fiber optics cables support high-speed point-to-point transmission, and have a low error rate.

2.2.10 Radio

Radio is a wireless bidirectional signal carried in the electromagnetic spectrum. The environment of propagation may cause signal reflection, obstruction (by objects in path) and interference. Radio link network types include terrestrial microwaves, LAN, wide-area and satellite.

2.3 Network Core

2.3.1 Packet Switching

Through store-and-forward packet-switching, an entire packet must arrive at a router before it can be transmitted on the next link. The resulting end-end delay is $\frac{2L}{R}$ (plus any propagation delay).

Should the arrival rate exceed a link's transmission rate, the resulting packets will queue up. If the memory in which the packets are stored fills up, packets can be dropped.

2.3.2 Routing vs Forwarding

Routing determines the source-destination route taken by packets, while forwarding moves packets to the appropriate output router.

2.3.3 Circuit Switching

Circuit switching is an alternative design for a network core. Instead of queuing up packets along shared lines, end-end resources between a transmission's source and destination are reserved (i.e. circuitry used only for that specific transmission). Such an approach is commonly used in telephone networks. Circuit switching can be implemented using FDM (frequency-division multiplexing) or TDM (time-division multiplexing).

2.3.4 Packet Switching vs Circuit Switching

Packet switching tends to be preferable to circuit switching for bursty data. Moreover, packet switching is supports resource sharing, and the setup for calls is simpler than that of circuit switching. However, packet switching may have excessive congestion, resulting in packet delay and loss. Providing circuit-like behaviour (i.e. guaranteed bandwidth) to packet switching networks remains an unsolved problem.

2.3.5 Internet Structure

End systems typically connect to the Internet through access Internet Service Providers (ISPs). Since competing ISPs exist, they must be connected to one another as well to effectively send packets to one another. These connections are implemented using Internet Exchange Points (IXPs). Moreover, some content providers may setup their own networks (content provider networks) to connect their data centers to the Internet, often bypassing regional ISPs. The resulting Internet network structure is quite complex, with its evolution having been driven by a combination of business and politics.

2.4 Delay, Loss and Throughput

2.4.1 Sources of Packet Delay

$$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$$

Nodal processing includes the time to check bit errors and determine the output link. This time is usually 10-1000us.

Queuing delay is the time spent waiting at the output link for transmission.

Transmission delay $\frac{L}{R}$

Propagation delay d/s, where d is the length of the physical link and s is the propagation speed.

2.4.2 Queuing Delay

Let R be the link bandwidth, L be the packet length, and a be the average package arrival rate. The average queuing delay can be measured using $\frac{La}{R}$. If this value is close to 0, the queuing delay is small. Once the value exceeds 1, more packets are arriving than can be serviced, and the queuing delay may be infinite. Should packets be dropped, they may be re-transmitted by a previous node in the network, but this is not guaranteed.

2.4.3 Measuring Delay and Loss

The traceroute program can be used to measure the delay and loss on a network. It works by sending some test packets to a router which then returns the packets towards the sender.

2.4.4 Throughput

Throughput is the rate (bits/sec) at which bits are transferred between a sender and a receiver. Throughput can be measured as instantaneous (rate at a single point in time) or average (rate over a period of time). A bottleneck link is a link on an end-to-end path that constrains end-to-end throughput. For example, if one link on a network has a throughput of 20Mbps and another has a throughput of 50Mpbs, then the bottleneck link is the 20Mpbs link.

2.5 Protocol Layers and Service Models

2.5.1 Protocol Layers

Each layer implements some sort of service, and may rely on services provided by the layer below. Layering helps simplify dealing with complex systems. In particular, a layering system helps in identification of a system's components and developing a model of how different components interact. Layering also allows for a modular design, which eases maintenance and updating processes.

2.5.2 Open System Interconnection (OSI) Model

OSI is a conceptual model describing the layers of telecommunication or computing systems. The OSI layers are:

- **Application:** Supports network applications
- **Presentation:** Allows applications to interpret meanings of data (e.g. for encryption, compression)
- **Session:** Synchronization, checkpointing and recovery of data (i.e. controls connection between computers)
- **Transport:** Provides process-process data transfer
- **Network:** Handles routing from source to destination
- **Link:** Provides data transfer between directly connected nodes
- **Physical:** Deals with physical specs of connection

2.6 Security

The field of network security deals with how computer networks can be attacked, defending against such attacks and architectures that minimize the risk of attacks. Internet technologies were not designed at first with much security in mind, though security considerations have since been added across all networking layers.

2.6.1 Types of Transmission

- **Virus:** Self-replicating infection started by opening/executing object
- **Worm:** Self-replicating infection started by passively receiving object

2.6.2 Examples of Attack

- **Spyware:** Records user action (e.g. keystrokes, websites visited)
- **Botnet:** A collection of infected hosts running bots that can be used for spam, distributed denial of service (DDoS attacks), etc.
- **Denial of Service (DoS):** An attack where network resources (e.g. bandwidth) become unavailable to legitimate traffic due to the introduction of large amounts of bogus traffic. A distributed denial of service (DDos attack) performs a DoS attack using a collection of hosts.
- **Packet Sniffing** Promiscuous network connections having their data read by a third party as it passes by
- **IP Address Spoofing** Packets being send from a false source IP address, which can be used to hide one's identity or impersonate another host

3 Application Layer

3.1 Network Application Introduction

Networks apps are designed to run on end systems, and use the network to communicate with other hosts. An example is a Web application, where the involved network apps are the browser running on the user host and the Web server program running in the Web server host. These network apps are only designed for end systems; network-core devices function on layers below the application layer.

3.2 Network Application Architectures

3.2.1 Network Application Architectures

A client-server architecture features an always-on host, called the server, which services requests from many user hosts, known as clients. Servers respond to client requests by returning requested data to them. Clients only communicate with the web server, and not between each other. A server has a permanent IP addresses, while clients may have dynamic IP addresses.

3.2.2 Peer-to-Peer (P2P) Architecture

P2P networks do not feature an always-on server, but rather a collection of end systems that may communicate directly with one another; peers can request services from and provide services to other peers. These systems are self-scalable, with new peers being able to bring

both new service capacity and new service demands. This type of architecture poses many challenges related to management and security.

3.3 Process Communication

A process is a program running within a host. If multiple processes run within the same host, they can communicate through inter-process communication procedures defined by the underlying OS. To communicate between hosts, processes need to exchange messages with one another.

3.3.1 Client vs Server Processes

Client processes run on client hosts, where they initiate communication. Server processes run on server hosts, where they wait to be contacted to server client requests. In a P2P architecture, hosts may need to run both client and server processes.

3.4 Sockets

A host sends and receives messages on a network through a software interface called a socket, serving as the interface between the application and transport layer. Sockets are sometimes referred to as the API between an application and network.

3.5 Addressing Processes

In order to send a message across a network to a destination host, the address of that host and an identifier that specifies the receiving process (socket) must be provided. A host is identified by its IP address, while its socket is identified by a port number. Popular applications are linked to specific port numbers (e.g. Web server has port number 80).

3.6 App-layer protocols

An application-layer protocol is responsible for defining the following:

- Types of messages exchanges (e.g. request, response)
- Message syntax
- Message semantics
- Rules for when and how processes send and response to messages

App-layer protocols may be open (e.g. HTTP, SMTP) or may be proprietary (e.g. Skype)

3.7 Transport Services

Candidate transport services can be evaluated along four main dimensions:

3.7.1 Data Integrity / Reliable Data Transfer

A protocol that guarantees that data sent between applications is delivered correctly and completely is said to provide reliable data transfer. Loss-tolerant applications (e.g. multimedia) do not require perfect data integrity.

3.7.2 Throughput

Applications that require a certain level of throughput to function correctly (e.g. multimedia) are described as bandwidth-sensitive. Applications that do not have strict throughput requirements are described as elastic.

3.7.3 Timing

Some applications (e.g. real-time chat, multiplayer games) may not function well if the time to communicate between source and destination applications exceeds a certain amount of time.

3.7.4 Security

Examples of security-related features a transport service can provide include encryption, enforcing authentication or ensuring data integrity.

3.8 Transport Services Provided by the Internet

The Internet makes two transport protocols available, namely TCP and UDP

3.8.1 TCP

TCP is a connection-oriented protocol, meaning that the client and server exchange transport-layer control info before application messages are exchanged. When the application finishes sending messages, it must tear down this connection. TCP also provides reliable data transfer. Also, for the welfare of the Internet in general rather than specific applications, TCP provides a congestion-control mechanism, which throttles sending processes when the network between the client and server is congested. Flow control is also provided.

3.8.2 Securing TCP

Neither TCP or UDP provide any encryption. To remedy this issue, an enhancement for TCP, known as Secure Sockets Layer (SSL), can be used to provide process-to-process security services, including encryption, data integrity and end-point authentication.

3.8.3 UDP

UDP is a lightweight transport protocol. Unlike TCP, UDP is not connection-oriented, and does not provide reliable data transfer. Also, UDP does not provide a congestion-control mechanism.

3.9 HTTP Overview

HyperText Transfer Protocol (HTTP) is the Web's main application-layer protocol. HTTP can be used to load web pages, which consist of objects than can include HTML files, image files, Java applets, etc.. Most Web pages consist of a base HTML file that references other objects. Each object is addressable by a Uniform Resource Locator (URL), which includes a host name and a path name.

HTTP uses TCP as its underlying transport protocol. HTTP clients (e.g. Web browsers) are responsible for initiating a TCP connection with a server, after which the two hosts can exchange messages through their socket interface. HTTP is considered to be a stateless protocol, meaning that HTTP servers are not required to maintain information about past client requests.

3.10 HTTP Connections

3.10.1 Non-Persistent Connections

In non-persistent HTTP connections, at most one object is sent over each connection, after which the connection is closed and a new one must be established. These types of connections have significant overhead, with TCP connection variables having to be stored on both the client and webserver, and each object suffering from a delivery delay of 2 Round-Trip Times (RTTs).

3.10.2 Persistent Connections

Persistent HTTP connections allow for a multiple objects from the same host to be sent over a single connection. Therefore, it is possible to have as few as one RTT of overhead for all objects on a server. This connection remains open until a configurable timeout interval, after which it is closed. The default mode of HTTP makes use of persistent connections.

3.11 HTTP Messages

3.11.1 Requests

Common request types:

- **GET:** Used to request an object, with the requested object identified in the URL (most common type of request)
- **POST:** Used to submit a request with an entity body (e.g. to submit data from a filled out form)
- **HEAD:** Similar to get, but leaves out the requested object
- **PUT:** Used to upload an object to a specific path
- **DELETE:** Used to delete an object on a specific path

3.11.2 Responses

Common response codes:

- **200 OK:** Request succeeded, requested object later in this message.
- **301 Moved Permanently:** Resource has been moved, new URL in Location: header of this message.
- **400 Bad Request:** Request message not understood by server.
- **404 Not Found:** Requested object not found on server.
- **505 HTTP Version Not Supported:** HTTP protocol version not supported by server.

3.12 Cookies

Although HTTP servers are stateless, it is often useful to be able to identify users, which can be done using cookies. Cookies technology has 4 components, namely:

1. Cookie header line in HTTP response message
2. Cookie header line in HTTP request message
3. Cookie stored on user end system
4. Backend database on website indexed by cookies

The key idea behind cookies it that they can be assigned to users when they first visit a website, after which the user can be identified because they include the cookie in their HTTP requests. Sample uses of cookies include authorization and saving user state and settings.

3.13 Web Caching

The goal of web caches (also known as proxy servers) is to satisfy a client request without involving the origin server. Instead, a client sends messages to a proxy that either already cached the results, in which case it responds to the request, or does not have the result, in which it sends a request to the origin server. Note that the proxy is acting as both a server and a client.

The main benefits of web caching are reducing response time (effective when client has high-speed connection to cache) and by reducing traffic. Caches are typically installed by an ISP (e.g. a university might install a cache and configure all clients to point to it).

3.14 Conditional GET

A conditional GET can be used by web caches to ensure they are not returning state data. Conditional GETs use GET requests but include an If-modified-since: header.

3.15 DNS: Domain Name System

DNS serves to provide a mapping between host names and their IP addresses. DNS implements this through a distributed database implemented in a hierarchy of servers known as name servers. DNS is used by the application layer, where hosts communicate with a DNS server to resolve names, and can then proceed with the request.

DNS services include:

- Hostname to IP translation
- Host aliasing
- Mail server aliasing
- Load distribution (i.e. cycling through replicated web servers to avoid overloading a single one)

Although a single centralized DNS may sound like an appealing idea, it would face many problems, including:

- Single point of failure - DNS server going down would make the Internet unusable
- Huge traffic volume to a single server
- Server would be physically distant from most users
- Maintenance concerns - the DNS database would be huge and would require very frequent updates

3.15.1 Server Classes

- **Root name servers** are contacted by local name server if the name cannot be resolved. Root name servers may contact servers lower on the hierarchy in resolving the name.
- **Top-level domain (TLD) servers** are responsible for top-level domains such as com, org, net, and country top-level domains.
- **Authoritative DNS Servers** are an organization's own DNS servers, providing authoritative hostname to IP mappings for their named hosts. These servers can be maintained by an organization itself or by a service provider.
- **Local DNS Name Servers** (also known as default name servers) serve as local proxies to DNS requests provided by an ISP. Local DNS Name Servers store a cache of name-to-address translation pairs, and in the case of a cache miss, are able to forward a request into the DNS hierarchy. Once forwarded into the DNS hierarchy, querying can be iterative (each server returns the next server to query to the local DNS server), or recursive (The root DNS server initiates a series of calls down the server hierarchy and ultimately returns the resolved name).

3.15.2 DNS Records

DNS servers store resource records (RRs), including RRs that map hostnames to IPs. An RR contains the following fields: (Name, Value, Tpe, TTL (time-to-live)). The record types are as follows:

- **Type=A** Name is hostname and Value is IP address.
- **Type=NS** Name is domain and Value is host-name of authoritative DNS server that can option IP address for hosts in the domain.
- **Type=CNAME** Name is alias and Value is the corresponding canonical hostname.
- **Type=MX** Name is alias and Value is the corresponding mail server.

3.15.3 DNS Messages

Query and reply are the only kinds of DNS messages. The parts of a message are described below:

- **Header:** Contains message identifier, which is copied from a query into its reply. The header also includes a number of flags, including one for query or reply, whether recursion is desired, whether recursion is available, and whether the reply is authoritative.
- **Questions:** Includes name and type field for a query.
- **Answers:** Contains records that were queried for.
- **Authority:** Contains records of other authoritative servers.

- **Additional:** Additional info (e.g. for a Type MX reply, this contains the Type A record for the canonical hostname of the mail server).

3.15.4 Inserting Records into DNS Database

Domain names can be registered at commercial entities called registrars. Once the domain is confirmed to be available, a Type A DNS record and a Type MX DNS record are inserted.

3.15.5 DNS Vulnerabilities

DDoS attacks against root servers are generally ineffective, since root servers have packet filters, and even if they do go down, most local DNS servers store the IPs of TLD servers.

A potentially more dangerous line of attack would be to DDoS TLD servers. However, even if TLD servers go down, the impact of the outage would be mitigated by caching in local DNS servers.

Other potential lines of attack include man-in-the-middle attacks, in which DNS queries from hosts are intercepted and bogus replies returned, and DNS poisoning, where bogus replies are sent to a DNS server to fill its cache with incorrect records. DNS infrastructure can also be used to launch a DDoS attack by sending DNS queries with a spoofed source of the attack target to many authoritative servers, with to queries designed so that their response is larger than the original query (i.e. the attacker's DoS efforts get amplified).

4 Transport Layer

The transport layer allows for logical communication between applications running on different hosts. Logical communication means that the applications involved can interact with one another as though they were directly connected. Transport-layer protocols are implemented in end systems, with the sender side breaking up application messages into transport-layer packets (or segments), and the receiving side reassembling them into messages send to the application layer.

4.1 Transport vs Network Layer

The transport layer is just above the network layer. The network layer provides logical communication between hosts, while the transport layer provides logical communication between processes on these hosts.

4.2 Multiplexing and Demultiplexing

Multiplexing and demultiplexing involves extending the network host-to-host delivery service to a transport-layer process-to-process delivery service.

4.2.1 Demultiplexing

Demultiplexing involves directing incoming transport segments to a particular socket. Each segment received by the host is packaged in an IP datagram, which contains header information that includes the destination and source port numbers. IP addresses and port numbers are used to direct the segment to the appropriate socket.

In connectionless demultiplexing, only the destination port number is involved in determining which socket should receive a segment; segments with the same dest port but different source IP addresses are sent to the same socket.

In connection-oriented demultiplexing, the source and dest IP address and port are all considered, with the received using all four values to direct the segment to the appropriate

4.2.2 Multiplexing

Multiplexing involves gathering data chunks from a host's sockets, encapsulating each chunk with header information that is later used in demultiplexing, and passing the segments to the network layer.

4.3 Connectionless Transport: UDP

UDP is a "bare bones" transport protocol that does just about as little as the transport layer can possibly do. The only major functionality added on top of IP is multiplexing/demultiplexing and some basic error checking. No "handshake" occurs between the sending and receiving processes before a segments are exchanged, making UDP a connectionless transport protocol.

A few potential benefits of UDP include:

- No connection establishment, lowering overall delay. DNS makes use of this property.
- Simpler to implement.
- Small header size.
- No congestion control.

4.3.1 UDP Segment Header

A UDP segment header consists of source port, dest port, segment length, and checksum fields. Each of these fields consists of 2 bytes, forming an 8 byte header.

4.3.2 UDP Checksum

The goal of a UDP checksum is to detect errors in the transmitted segment. The checksum is formed by treating the UDP segment as a sequence of 2 byte words, summing them up (with overflow bits wrapping around and being added to the front), and taking the 1s complement. The receiver re-computes the checksum, and if it differs from the sent one, it can detect that an error occurred.

4.4 Principles of Reliable Data Transfer

Reliable data transfer (rdt) is an extremely important problem encountered in networking not only at the transport layer, but also at the link layer and application layer. Reliable data transfer is provided by a reliable data transfer protocol, and must work even if the layers below it are unreliable. This section covers unidirectional rdt; bidirectional rdt is conceptually similar but more tedious to explain.

4.4.1 RDT over Perfectly Reliable Channel

This is a trivial case where no error checking is necessary, since errors do not occur in the first place.

4.4.2 RDT over Channel with Bit Errors

Three new mechanisms can be added to help overcome bit errors:

1. **Error detection**, which can be done through computing a checksum.
2. **Receiver feedback**. Acknowledgments (ACKs) can be sent to the sender to notify them a packet was received ok, and negative acknowledgments (NAKs) can be sent to the sender to notify them of an error.
3. **Retransmission**. Upon receiving a NAK, the sender can re-send the packet.

This described protocol is known an example of a stop-and-wait protocol, since a sender will not send a new message until it receives acknowledgment that the receiver has successfully received the previous one.

A major limitation of the system above is that it assumes the ACK and NAK signals are not corrupted. This can be solved by a sender re-transmitting a packet if a corrupted ACK or

NAK is received. A sequence number must also be added so that duplicates can be discarded by the receiver.

Moreover, if sequence numbers are used, NAKs do not need to be sent, since messages that were not acknowledged successfully are the ones without an ACK corresponding to their sequence number.

4.4.3 RDT over Channel with Errors and Loss

The possibility of packet loss over a channel adds to problems of packet loss detection and handling. This can be addressed by the sender using a countdown timer mechanism for each packet, which will trigger the packet to be re-sent if an ACK signal is not received on time. Since it is possible that a packet is not lost by merely delayed, this opens up the possibility of duplicate packets being sent, but this is already handled through packet sequence numbers.

4.4.4 Pipelined RDT Protocols

Although the protocol described above correctly handles packet error and loss, it has incredibly poor performance due to its stop-and-wait behaviour. Pipelined RDT protocols can be applied to overcome this stop-and-wait limitation by allowing multiple "in-flight", yet to be acknowledged packets. As a general rule, these protocols require that a larger range of sequence numbers be used, and that buffering be supported at the sender or receiver.

Go Back N (GBN) protocols allow for a sender to have up to N unacknowledged packets in transport at a given time. The receiver sends cumulative acknowledgments, meaning that packets are only acknowledged if they arrive in order. Should a sender timer for receiving an ACK for a packet expire, the next N packets will be retransmitted. GBN is often referred to as a sliding window protocol, with N referred to as the window size.

Selective Repeat protocols differ from GBN protocols in that they allow the receiver to acknowledge received packets individually. Like in GBN, a window size of N is used to limit the number of unacknowledged packets sent. However, unlike in GBN, the sender may have already received AKCs for some of the packets in this window, since the receiver acknowledges received packets regardless of whether they are in order. Out-of-order packets are buffered in the receiver until the preceding packets are received, after which they can be delivered to the upper layer. A timeout mechanism is also used by the sender for every packet.

4.5 Connection-Oriented Transport: TCP

4.5.1 Overview

TCP is a connection-oriented transport protocol because before a sender and receiver can exchange information, some sort of "handshake" between them must first occur. The TCP protocol runs only on the end systems, and its state is not stored in the intermediary network elements. TCP supports full duplex (i.e. bi-directional) point-to-point (i.e. one sender, one receiver) transport.

4.5.2 Segment Structure

When TCP messages are sent, they are often broken up into chunks of the maximum segment size (MSS), or potentially into smaller chunks for interactive applications. The header of a TCP segment includes:

- Source and dest port numbers
- Sequence number and acknowledgment number, which are used in implementing a reliable data transfer service
- Header length, which specifies length of TCP header in words (length can vary due to options field)
- Flag fields (ACK bit for acknowledgments, RST, SYN and FIN bits for connection setup and teardown, PSH bit indicating receiver should push data to upper layer immediately, and URG used to indicate whether there is urgent data
- Receive window, which is used for flow control
- Checksum field, as in UDP
- Urgent data pointing indicating the last word of urgent data

The sequence number is the byte-stream number of the first byte in a segment. The acknowledgment number is the number of the next byte expected from the sender. TCP provides cumulative acknowledgments with respect to the bytes in a stream, since it only acknowledges bytes up to the first missing one. Upon receiving an out-of-order segment, the TCP protocol allows for either discarding the segment (rarely used in practice) or buffering it (more common).

4.5.3 Timing Estimation

TCP makes use of a timing mechanism to recover from potentially lost segments. If the timeout value is too short, many unnecessary retransmissions are likely, while if it is too long, the reaction to segment loss will be too slow.

Estimating the RTT can help in determining a good timeout value. Since RTT can fluctuate significantly for specific samples, it is typically estimated with a moving average of previous RTT values:

$$RTT_{Estimated} = (1 - \alpha) \cdot RTT_{Estimated} + \alpha \cdot RTT_{Sample}$$

A typical value for α is 0.125.

The level of variability in the RTT can be measured as follows:

$$Dev_{RTT} = (1 - \beta) \cdot Dev_{RTT} + \beta \cdot |RTT_{Sample} - RTT_{Estimated}|$$

A typical value for β is 0.25.

The TCP retransmission timeout interval is defined as:

$$TimeoutInterval = RTT_{Estimated} + 4 \cdot Dev_{RTT}$$

4.5.4 Reliable Data Transfer

TCP provides rdt on top of IP's unreliable service. Some properties of TCP's rdt mechanism include pipelined segments, cumulative ACKs, and a single retransmission timer. Retransmissions are triggered by timeout events and duplicate ACKs.

The following sender events can take place:

- **Receiving data from the application layer.** In this case, a segment is created with the appropriate sequence number. Also, if the sender's single timer is not already running for some other segment, the timer is started.
- **Timeouts**, which result in a retransmission of the segment that caused the timeout, as well as a restart of the timer.
- **Arrival of ACK**, which updates the segments known be the ACKed (using cumulative ACK logic), and if there are no unACKed segments remaining, restarts the timer.

The following receiver events can take place:

- **Arrival of in-order segment.** If all data up to the expected sequence number is already ACKed, and the receiver waits 500ms for the next segment. If it does not arrive, then an ACK is sent. If there is another segment with an ACK pending, a single cumulative ACK is immediately sent.
- **Arrival of out of order segment with higher than expected sequence number.** A duplicate ACK is immediately sent, indicating the sequence number of the expected byte.
- **Arrival of segment that partially or completely fills in lower end of gap.** An ACK is immediately sent for the segment.

Duplicate ACKs are ACKs that reacknowledge a segment for which a sender has received a previous ACK. TCP does not use NAK signals, so if a segment arrives out of order, an ACK signal is re-sent for the last in-order byte of data received. Should three duplicate ACKs be received, the sender can re-transmit the next segment even if the timer has not yet expired, which is known as fast retransmit.

4.5.5 Flow Control

TCP provides a flow control service to prevent the sender from overflowing the receiver's buffer. The available buffer size is exchanged though the receive window variable, and since TCP is full duplex, this information is sent by both the sender and the receiver.

4.5.6 Connection Management

A two-way handshake involves a host sending a connection request to another host, which in turn replies with a message. However, two-way handshakes can lead to deadlocks or half-open clients, so TCP makes use of a three-way handshake. In a three-way handshake, a host sets the SYN bit to 1 and sets an initial sequence number (isn). The receiver replies with a SYN bit of 1, an acknowledgment of the client isn + 1, and ACK bit of 1, and its own initial sequence number. This part of establishing a connection is known as the SYNACK segment. One the original sender receives this message it replies with a sequence number of the server isn + 1, and sets the ACK bit to 1.

Any two hosts in a connection can close the connection, which involves sending a TCP segment with a FIN bit of 1. The receiver of that segment responds with an ACK bit of 1. The original sender then replies with a message to acknowledge the connection shutdown, after which the connection is closed and the resources for that connection can be deallocated.

4.6 Principles of Congestion Control

4.6.1 Causes of Congestion Control

Below is a list of three increasingly complex scenarios that describe sample causes of network congestion:

1. **Two senders and a router with infinite buffers.** If the throughput of the network does not keep up with the sending rates of the hosts, the delay will grow asymptotically.
2. **Two senders and a router with finite buffers.** Packet loss or long delays due to network congestion may lead to packet retransmissions, which in turn contribute further to network congestion.
3. **Multiple senders, routers with finite buffers, multihop paths.** Network congestion can cause delays or packet loss between different hosts that happen to be

communicating through the same infrastructure (i.e. same router). More routers involved in delivering packets opens the possibility for more wasted effort, since since upstream effort in delivering a packet that ends up getting dropped later is wasted.

4.6.2 Approaches to Congestion Control

The general approaches to congestion control are:

- **End-to-end congestion control**, which is a form of congestion control handled only by end systems, which infer network congestion based on observed network behavior. This is the method of congestion control used by TCP.
- **Network-assisted congestion control**, which is a form of congestion control where network routers provide feedback on congestion to end systems. This feedback can range be something simple, like a single congestion bit, or something more complex like an explicit rate at which the sender should send.

4.7 TCP Congestion Control

Since the IP layer provides no feedback regarding network congestion, TCP implements an end-to-end congestion control mechanism. TCP congestion control introduces a new variable known as a congestion window (cwnd), which limits the number of unACKed packets sent by a sender (should be minimum of receive window and congestion window).

When a TCP connection is first established, the initiation cwnd is set very low (1 minimum segment size (MSS)). However, on each successful ACK, this segment size doubles. Once some packet loss in indicated, cwnd is either halved and then grows linearly (the newer TCP Reno, which implements this "fast recovery") or is set back to 1 (the older TCP Tahoe). In general, the cwnd increase should switch from exponential to linear once it reaches half of its previous value before timeout.

4.7.1 TCP Throughput

Once a connection experiences packet loss, fast recovery implementations of TCP halve its value, after which it linearly grows back to its max value. Thus, ignoring the relatively short start phases, the average throughput of a connection is $\frac{3}{4} \cdot \frac{W}{RTT}$, where W is the window size where loss events begin to occur, and RTT is the round trip time.

When packet loss is factored in, the average throughput of a connection becomes around $\frac{1.22 \cdot MSS}{RTT\sqrt{L}}$, where L is the packet loss ratio. A key takeaway from this is that loss ratios often need to be very small in order for throughput to be close to its nominal maximum.

4.7.2 TCP Fairness

The goal of fairness is that K TCP connections sharing a link of bandwidth R should each have an average bandwidth of $\frac{R}{K}$. TCP's congestion control mechanism makes it fair, though since UDP does not implement congestion control, it does not adhere to such fairness. However, even with TCP, this fairness restriction can be circumvented by opening multiple parallel connections between the same hosts.

5 Network Layer

5.1 Overview

The network layer handles transporting segments from one host to another. Segments from the sending host are encapsulated into datagrams (i.e. network-layer packets), and segments at the receiving host have their transport-layer segments extracted and passed up to the transport layer.

5.1.1 Forwarding and Routing

To transport packets from one host to another, the network layer makes use of two main functions:

- **Forwarding**, which involves a router moving a packet from an input link to the appropriate output link.
- **Routing**, which involves the network layer applying a routing algorithm to determine the route taken by a packet between hosts.

Each router stores a forwarding table, which provides a map between a certain incoming header field and the output link to be used. Each received packet is looked up in this field to determine its output link.

5.1.2 Connection Setup

On top of forwarding and routing, some network architectures use a third function for connection setup. In such networks, before two end hosts can exchange data, the routers along the chosen path must also undergo some handshake procedure.

5.1.3 Network Service Models

A network service model defines some characteristics of end-to-end packet transport. Examples of such characteristics include:

- Whether delivery is guaranteed.
- Whether delivery delay is bounded.
- Whether packet delivery is in-order.
- Whether some minimum bandwidth is guaranteed.