Week 3: The Transport and Application Layers

The **Transport layer** allows traffic to be directed to specific network applications.

The **Application layer** allows these applications to communicate in a way they understand.

The Transport Layer

The transport layer is responsible for lots of important functions of reliable computer networking. These include multiplexing and demultiplexing traffic, establishing long running connections and ensuring data integrity through error checking and data verification.

The Transport Layer

Multiplexing means that nodes on the network have the ability to direct traffic toward many different receiving services.

- EX: Processes -> Multiplexer -> IP

Demultiplexing is the same concept, just at the receiving end. It's taking traffic that's all aimed at the same node, and delivering it to the proper receiving service.

- IP -> Demultiplexer -> Processes
- The transport layer handles multiplexing and demultiplexing through ports.
    - A **port** is a 16-bit number that's used to direct traffic to specific services running on a networked computer.

Remember:

- A *server* or *service* is a program running on a computer waiting to be asked for data.
- A *client* is another program that is requesting this data.

Different network services run while listening on specific ports for incoming requests.

- Ports are normally denoted with a colon after the IP address.
    - Example: `10.1.1.100:80`
    - When written this way, it's known as a *socket address* or *socket number*.

File Transfer Protocol (FTP) Server

- FTP is an older method, used for transferring files from one computer to another.
    - It still sees use today.
- TFP traditionally listens on port `21` .

Multiplexing and demultiplexing, combined with ports, make it possible for a single server to host many applications (like an internal website, a mail server, a file server, and a print server).

Dissection of a TCP Segment

Just like how an Ethernet frame encapsulates an IP datagram, an IP datagram encapsulates a TCP segment.

And just like the Ethernet frame's payload section is an entire IP datagram; the IP datagram's payload section is an entire TCP segment.

A **TCP segment** is made up of a TCP header and a data section.

- (This data section is yet another payload section, where the application layer places its data.)

TCP Header

A TCP header itself is split into many fields containing lots of information.

- First are the **source port** and **destination port** fields.
  - The source port is a high-numbered port, chosen from a special section of ports known as *ephemeral ports*.
    - A source port is required to keep lots of outgoing connections separate.
    - A source port is needed so that when a web server replies, the computer making the original request can send the data to the program that was actually requesting it.
  - The destination port is the port of the service the traffic is intended for.
- Next is the **sequence number**, a 32-bit number that's used to keep track of where in a sequence of TCP segments this one is expected to be.
  - Remember that large data packets are split into many segments.
- Next is the **acknowledgement number**, another 32-bit number.
  - The acknowledgement number is the number of the next expected segment.
  - It works in tandem with the sequence number; IE if the sequence = 1, the acknowledgement = 2.
- The **data offset field** is a 4-bit number that communicates how long the TCP header for this segment is.
  - This is so the device receiving the data knows when the header ends, and the payload begins.
- Then 6 empty bits follow, reserved for the 6 TCP control flags.
- Then comes another 16-bit field, the **TCP window**.
  - The TCP window specifies the range of sequence numbers that might be sent before an acknowledgement is required.

> the sending device doesn't waste time sending data that isn't being received.

- The next field is a 16-bit checksum.
  - The **TCP checksum** operates just like the checksum fields at the IP and ethernet level.
    - "*Once all of this segment has been ingested by a recipient, the checksum is calculated across the entire segment and is compared with the checksum in the header to make sure that there was no data lost or corrupted along the way.*"
- The **urgent pointer field** is another 16-bit field.
  - It's used in conjunction with one of the TCP control flags, to point out particular segments that might be more important than others.
  - "*This is a feature of TCP that hasn't really ever seen adoption and you'll probably never find it in modern networking.*"
- Next is the **options field**, ranging from 0 to 16 bits.
  - This is another rarely-used field in the real world.
  - It's sometimes used for more complicated flow control protocols.
- Finally, there's **padding**, which is just as sequence of 0s to ensure that the **data payload** begins at the expected location.

TCP Control Flags and the Three-Way Handshake

- TCP protocol oversees the transmission of long, segmented chains of data over a connection.
  - Contrast this to Ethernet and IP protocol, which just send individual packets of data.
- TCP establishes a connection through the use of different TCP control flags (used in a very specific order).

The 6 TCP Control Flags

Note: This is in the order they appear in a TCP header, not ordered by frequency/importance

1. **URG (urgent)**

- A value of 1 here indicates that the segment is considered urgent, and that the urgent pointer field has more data about this.

2. **ACK (acknowledged)**

- A value of 1 in this field means that the acknowledgement number field should be examined.

3. **PSH (push)**

- The transmitting device wants the receiving device to push currently-buffered data to the application on the receiving end as soon as possible.
  - A **buffer** is a computing technique where a certain amount of data is held somewhere, before being sent somewhere else.

4. **RST (reset)**

- One of the sides in a TCP connection hasn't been able to properly recover from a series of missing or malformed segments.

5. **SYN (synchronize)**

- It's used when first establishing a TCP connection, and ensures the receiving end knows to examine the sequence number field.

6. **FIN (finish)**

- When this flag is set to 1, it means the transmitting computer doesn't have any more data to send, and the connection can be closed.

The Three-Way Handshake

How a TCP connection is established

A **handshake** is a way for two devices to ensure that they're speaking the same protocol, and will be able to understand each other.

Transmitter	Three-Way Handshake	Receiver
Computer A	SYN --->	Computer B
Computer A	<--- SYN/ACK	Computer B
Computer A	ACK --->	Computer B

Once three-way handshake is complete (both sides have sent SYN/ACK pairs to each other), the TCP connection is operating in full duplex.

Once one of the devices involved with the TCP connection is ready to close the connection, something known as a *four-way handshake** happens:

The Four-Way Handshake

Transmitter	Four-Way Handshake	Receiver
Computer A	<--- FIN	Computer B
Computer A	ACK --->	Computer B
Computer A	FIN --->	Computer B
Computer A	<--- ACK	Computer B

TCP Socket States

- A **socket** is the instantiation of an end-point in a potential TCP connection.
  - An **instantiation** is the actual implementation of something defined elsewhere.

TCP sockets require actual programs to actually instantiate them; as opposed to ports, which is more of a virtual/descriptive thing.

- IE: You can send traffic to any port you want, but you're only going to get a response if a program has opened a socket on that port.

TCP Socket States

TCP sockets can exist in lots of states. Here are some common ones:

- **LISTEN**
  - A TCP socket is ready and listening for incoming connections.
  - Seen on server-side only.
- **SYN_SENT**
  - A synchronization request has been sent, but the connection hasn't been established yet.
  - Seen on client-side only.
- **SYN-RECEIVED**
  - A socket previously in a LISTEN state has received a synchronization request and sent a SYN/ACK back.
  - Seen on server-side only.
- **ESTABLISHED**
  - The TCP connection is in working order and both sides are free to send each other data.
  - Seen on both client- and server-sides.
- **FIN_WAIT**
  - A FIN has been sent; but the corresponding ACK from the other end hasn't been received yet.
  - Seen on both client- and server-sides.
- **CLOSE_WAIT**
  - The connection has been closed at the TCP layer; but the application that opened the socket hasn't released its hold on the socket yet.
  - Seen on both client- and server-sides.

- **CLOSED**
  - The connection has been fully terminated and no further communication is possible.
  - Seen on both client- and server-sides.

There are other TCP socket states that exist. Also, socket state definitions vary with operating systems.

"*When troubleshooting issues at the TCP layer, make sure you check out the exact socket state definitions for the systems you're working with.*"

Connection-oriented and Connectionless Protocols

TCP is a connection-oriented protocol.

- A **connection-oriented protocol** establishes a connection, and uses this to ensure that all data has been properly been transmitted.
- A connection at the transport layer implies that every segment of data sent is acknowledged.

TCP expects an ACK for every bit of data it sends.

Connection-oriented protocols safeguard against data transmission errors, by a constant stream of acknowledgements in a connection.

- This is why sequence numbers are important.
  - If data segments are resent due to errors at lower levels, and the segments are out of order, it isn't a problem thanks to the sequence number's guidance.

The high overhead of connection-oriented protocols are expensive in terms of traffic, however.

- Establish the connection; send constant ACK stream; tear down connection at the end; etc.

This is where **connectionless protocols** come into play.

- The most common is **User Datagram Protocol (UDP)**
  - UDP doesn't rely on connections; and doesn't even support acknowledgement.
  - Instead, you just set the destination port, and send the packet.

A great use-case for UDP is streaming video:

- By removing TCP's overhead, you can send higher-quality video.
- More available bandwidth will be reserved for actual data transfer (the video frames), rather than for establishing constant acknowledgements that each video frame was delivered and received.

System Ports vs. Ephemeral Ports

Transportation layer protocols use a concept of ports and multiplexing/demultiplexing to deliver data to individual services listening on network nodes.

- These ports are represented by a single 16-bit number (thus represent numbers 0-65,535)

These ranges have been designated by the **Internet Assigned Numbers Authority (IANA)**:

- `Port 0` isn't in use for network traffic; but it's sometimes used in communication taking place between different programs on the same computer.
- `Ports 1-1023` are referred to as **system ports**, also known as **well-known ports**.
  - These represent official ports for most well-known network services.
    - EX: HTTP normally communicates over `Port 80`, FTP over `Port 21`.
  - (In most OS) administrator-level access is required to start a program that listens on a system port.
- `Ports 1024-49151` are known as **registered ports**.
  - These are used for many other network services, which are less common than those used for system ports.
    - EX: `Port 3306`, which is the port many databases listen on.
  - (On most OS) any user of any access level can start a program listening on a registered port.
- `Ports 49152-65535` are **private** or **ephemeral ports**.
  - Ephemeral ports can't be registered with the IANA, and are generally used for establishing outbound connections.
  - Remember that all TCP traffic uses a destination port and a source port.
    - When a client wants to communicate with a server, the client will be assigned an ephemeral port to be used for just that one connection, while the server listens on a static system or registered port.

Not all operating systems follow the ephemeral port recommendations of the IANA. It depends on the platform. Sometimes registered port ranges are used; but no modern operating system will ever use a system port for outbound communication.

Firewalls

A **firewall** is a device that blocks traffic that meets certain criteria.

Firewalls are the primary way to stop traffic you don't want from entering a network.

- Thus, they are critical to network security.

Firewalls are most commonly used at the transportation layer; but they also exist at other levels:

- Inspection of application layer traffic; blocking ranges of IP addresses; etc.

Firewalls at the transportation layer focus on blocking traffic to certain ports, while allowing traffic to other ports.

Firewalls are sometimes independent network devices; but it's better to think of them as a program that can run anywhere.

- For many companies--and almost all home users--the functionality of a router and a firewall are performed by the same device.
- Firewalls can run on individual hosts, too, instead of being a network device.

The Application Layer

The Application Layer

Just like every other layer, the TCP segment's payload section is actually the entire contents of whatever data applications want to send to each other. Also known as the *message*.

- It could be:
  - Contents of a web page, if a web browser is connecting to a web server;
  - Streaming video content of the Netflix app on your PlayStation connecting with Netflix servers;
  - The contents of a document your word processor is sending to a printer; etc.

Unlike other layers in the OSI model, there is no de facto protocol at the application layer--there are many across many different applications.

The Application Layer and the OSI Model

The TCP/IP Model defines five layers, based on the five layers of *encapsulation*.

The **Open Systems Interconnection (OSI)** model, by contrast, is a seven-layer model. The two other layers are an abstraction of the Applications Layer: 7. Application 6. Presentation 5. Session 4. Transport 3. Network 2. Data Link

1. Physical

The **Session Layer** facilitates the communication between the actual *applications* and the Transport Layer.

- It's the part of the operating system that takes the unencapsulated *application layer* data and hands it off to the presentation layer.

The **Presentation Layer** is responsible for making sure the unencapsulated *application layer* data is able to be understood by the application in question.

- It's the part of the operating system that might handle encryption or compression of data.

The OSI model is the most rigorously-defined, meaning it's often used in academic settings, or by network certification organizations.

- CISCO uses the OSI model, for example.

Questions for review

Keywords

Concepts