Ch.2: Building Blocks of TCP

Ciro S. Costa

Jul 07, 2015

The HTTP standard does not specify TCP as the only transport protocol but in practice all HTTP trafic on the web is deliveried via TCP. IPv4 and IPv6 defines two different protocols:

- **IP**: *Internet Protocol* > Provides the host-to-host routing and addressing.
- **TCP**: *Transmission Control Protocol* > Provides the abstraction of a reliable network running over an unreliable channel, hiding most of the complexity of network communication from applications.

TCP provides reliable, ordered and error-checked delivery of a stream of octets between applications running on hosts communicating over an IP network, as opposed to other protocols that are used when the application do not require such guarantees (e.g UDP that emphasizes reduced latency over reliability). It also includes retransmission of lost packets (any cumulative stream not acknowledged is retransmitted), flow control and congestion control.

The protocol accepts data from a data stream, dividing it into chunks and then adding a TCP header, creating a TCP segment. This segment is then encapsulated into an Internet Protocol datagram and exchanged with peers.

TCP operations are divided into three phases. Connections must be properly established in a multi-step handshake process (connection establishment) before entering the data transfer phase. After data transmission is completed, the connection termination closed established virtual circuits and releases all allocated resources.

TCP best practices and underlying algorithms that govern its performance continue to evolve, and most of these changes are only available in the latest kernels.

Three-Way Handshake

3-way handshake is needed for establishing the connection. Even before a client attempts to connect the server must first bind to and listen at a port to open it

up for connections (passive open). Once this is established a client may then initate an active open. The handshake comes:

- SYN: client picks a random sequence number X and sends a SYN packet, which may contains additional TCP flags and opts
- SYN ACK: server increments X by one, picks own random sequnece number Y, appends own set of flags and options and then dispatches the response
- ACK: client increments Y and X by one, completes the handshake by dispatching the ACK packet in the handshake.

Having done that, both the client and server have received an acknowledgment of the connection.

Because of the delay imposed by the process of creating a new TCP connection we must aim to reuse the connection.

ps.: There's a proposal of TCP Fast Open which aims to reduce the latency penalty by sending some data right in the SYN package. There are some limitations though.

Congestion Avoidance and Control

Congestion control is a method of ensuring that everyone across a network has a 'fair' amount of access to network resources, at any given time. TCP uses a number of mechanisms to achieve high performance and avoid congestion collapse, a state where network can fall by several orders of magnitude. The idea is to control the rate of data entering the network, keeping the data flow below a rate that would trigger collapse.

Flow Control

> Mechanism to prevent the sender from overwhelming the receiver with data it may not be able to process. Each side of the TCP connection advertises its own receive window (*rwnd*) - *sliding window flor control protocol*-, which communicates the size of the available buffer space to hold the incoming data. The workflow continue throughout the lifetime of every TCP connection (specifies the 'rwnd' for each TCP segment).
>
> Each ACK packet carries the latest 'rwnd' value for each side, allowing both sizes to dynamically adjust the data flow rate to the capacity and processing speed of the sender and receiver.

It is controlled by the receiving side. Ensures that the sender only sends what the receiver can handle.

(for improving the amount of possible data to be sent/received there is Window Scaling, which is a way of specifying by 'how many bitshifts to perform on the 16bit window size value').

Slow Start

Even though with flow control we are able of preventing a congestion in the receiver side, we don't prevent the congestion in the network. To estimate the available capacity between the client and the server it is necessary to measure it by explicitly exchanging data.

The server initializes a new *congestion window* (`cwnd`) variable per TCP conection setting its initial value to a conservative value. This variable defines the sender-side limit on the amount of data the sender can have in flight before receiving an acknowledgement (ACK) from the client. This is kept private to the sender. The max amount of data in flight between the client and server is then the minimum of `rwnd` and `cwnd` variables.

The idea of slow start is to start with a slow value and then quickly try to converge on the available bandwidth on the network path between them.

Slow-start is not a big of an issue for large, streaming downloads, as the client and the server will arrive at their maximum window sizes after a few hundrer milliseconds and continue to transmit at near maximum speeds - the cost of the slow-start phase is amortized over the lifetime of the large transfer.

For short HTTP connections it's not unnusual for a request to terminate before the maximum window size is reached. As a result, the performance of many web applications if often limited by the roundtrip time between server and client. **Optimize this by moving server geographically closer to the client or increase the initial congestion window size to the new RFC-defined value of 10 segments**.

Congestion Avoidance

In order to give feedback to regulate the performance TCP make use of packet loss. The feedback to the slow stard might be given in the case of two events: reach of system-configured congestion threshold (`ssthresh`) or a packet is lost, indicates that we have something happening and we must adjust our window to avoid more losses.

Bandwidth-Delay product

> Product of data link's capacity and its end-to-end delay. The result is the maximum of unacknowledged data that can be in flight at any point in time.

To surpass the problem of waiting wihout flowing data the window sizes must be properly set so that either side can continue sending data untial an ACK arrives back from the client for an earlier packet - no gaps, maximum throughput. This depends on the RTT.

Fortunately window size negotiation and tunning is managed automatically by the network stack and should adjust accordingly.

Head-Of-Line Blocking

This is a performance-limiting phenomenon that occurs when a line of packets is held-up by the first packet. If one packet is lost, then all subsequent packets must be held in the receiver's TCP buffeer until the lost packet is retransmitted and arrives at the receives. The application will face a delivery delay when it tries to read the data from the socket. Such unpredicatable latency variation in the packet arrival times is commonly referred as *jitter*.

Summary

- TCP three-way handshake introduces a full roundtrip of latency
- TCP slow-start is applied to every new connection
- TCP flow and congestion control regulates throughput of all connections
- TCP throughput is regulated by current congestion window size.
- No bit is faster than one that is not sent; send fewer bits
- We can't make the bits travel faster, but we can move the bits closer
- TCP connection reuse is critical to improve performance