Section 12: TCP and Distributed Systems

CS162

November 12-13, 2018

Contents

1 Warmup

a) (True/False) IPv4 can support up to 2^{64} different hosts.

False, it has 32 bits per IP.

b) (True/False) Port numbers are in the IP packet.

False, they are in the transport layer. (TCP/UDP).

c) (True/False) UDP has a built in abstraction for sending packets in an in order fashion.

False, this is a part of the TCP protocol. In UDP there is no notion of a sequence number.

d) (True/False) TCP is built in order to provide a reliable and ordered byte stream abstraction to networking.

True.

e) (True/False) TCP attempts to solve the congestion control problem by adjusting the sending window when packets are dropped.

True.

f) In TCP, how do we achieve logically ordered packets despite the out of order delivery of the physical reality? What field of the TCP packet is used for this?

The seqno field.

g) Describe how a client opens a TCP connection with the server. Elaborate on how the sequence number is initially chosen.

3 way handshake. Client sends a random sequence number (x) in a syn packet. Server sees this and sends another random seqeuence number back (y) in addition to acknowledging the sequence number that it received from the client (sends back x+1) in a syn-ack packet. Client acknowledges this sequence number in an ack packet by sending back y+1.

It is important to randomize the sequence number so an off path attacker cannot guess it and send spurious packets to you.

h) Describe the semantics of the acknowledgement field and also the window field in a TCP ack.

The acknowledgement field says that the receiver has received all bytes up until that number (x). The window field says how many additional bytes past x the receiver is ready to receive.

2 Vocabulary

- **TCP** - Transmission Control Protocol (TCP) is a common L4 (transport layer) protocol that guarantees reliable in-order delivery. In-order delibery is accomplished through the use of sequence numbers attached to every data packet, and reliable delivery is accomplished through the use of ACKs (acknowledgements).
- **Flow Control** - Flow control is the process of managing the rate of data transmission such that a fast sender doesn't overwhelm a slow receiver. In TCP, flow control is accomplished through the use of a sliding window, where the receiver tells the sender how much space it has left in its receive buffer so that the sender doesn't send too much.
- **RPC** - Remote procedure calls (RPCs) are simply cross-machine procedure calls. These are usually implemented through the use of stubs on the client that abstract away the details of the call. From the client, calling an RPC is no different from calling any other procedure. The stub handles the details behind marshalling the arguments to send over the network, and interpreting the response of the server.
- **General's Paradox** - The idea that there is no way to guarantee that two entities do something simultaneously if they can only send messages to each other over an unreliable network. There is no way to be sure that the last message gets through, so one entity can never be sure that the other entity will act at a specific time.

3 Problems

3.1 TCP

In order for TCP flow control to work correctly, the receiver advertises a window that the sender is allowed to send in. TCP connections are initiated through a 3-way handshake — the sender sends a SYN packet, the receiver responds with a SYN/ACK packet, and the sender finishes the handshake with an ACK packet. Information about the maximum buffer size for the receiver is transmitted during this handshake. Packets with the ACK flag set contain the sequence number expected in the next packet. For example, if the sender's first data packet (with sequence number 1) contains 200 bytes, the receiver would respond with an ACK packet containing the number 201.

Receiver:

- LastByteRcvd − LastByteRead ≤ MaxRcvBuffer
- AdvertisedWindow = MaxRcvBuffer − (LastByteRcvd − LastByteRead)

Sender:

- LastByteSent − LastByteAcked ≤ AdvertisedWindow

Consider a connection where the sender S wants to send 700 bytes to the receiver R. We make the following assumptions:

- The maximum packet size is 200 bytes.
- Maximum size of R's receiving buffer is 300 bytes.
- R consumes an in-sequence packet p right after the ACK for p is sent.
- The time it takes for a packet to travel from S to R is much longer than any processing time on either side.

List the sequence of packets for each situation below; for sent packets, include the start and end bytes, and for received packets, include the ACK number and the advertised window.

a) No packets are lost.

Send: [1, 200]
Send: [201, 300]
Recv: ACK 201, AdvWin 100
Recv: ACK 301, AdvWin 200
Send: [301, 500]
Recv: ACK 501, AdvWin 100
Send: [501, 600]
Recv: ACK 601, AdvWin 200
Send: [601, 700]
Recv: ACK 701, AdvWin 200

b) The first sent packet is lost.

"Easy" solution (slightly inconsistent, but easier to explain):
Send: [1, 200] LOST
Send: [201, 300]
Recv: ACK 1, AdvWin 0
Send: [1, 200]
Recv: ACK 301, AdvWin 300 *
Send: [301, 500]
Send: [501, 600]
Recv: ACK 501, AdvWin 100
Recv: ACK 601, AdvWin 200
Send: [601, 700]
Recv: ACK 701, AdvWin 200
*This is inconsistent b/c we said packets are consumed from the buffer after they are ack'd.

Alternate "Easy" solution (consistent, but wrong, and easier to explain):
Send: [1, 200] LOST
Send: [201, 300]
Recv: ACK 1, AdvWin 0
Send: [1, 200]
Recv: ACK 301, AdvWin 0 **
Recv: ACK 301, AdvWin 300 **
Send: [301, 500]
Send: [501, 600]
Recv: ACK 501, AdvWin 100
Recv: ACK 601, AdvWin 200
Send: [601, 700]
Recv: ACK 701, AdvWin 200
**This is inaccurate b/c we only ack upon the receipt of a packet. The receiver would never send another ack without receiving another packet.

Exact solution: Send: [1, 200] LOST
Send: [201, 300]
Recv: ACK 1, AdvWin 0
Send: [1, 200]
Recv: ACK 301, AdvWin 0
Send: [301] ***
Recv: ACK 302, AdvWin 299
Send: [302, 500]
Send: [501, 600]
Recv: ACK 501, AdvWin 100
Recv: ACK 601, AdvWin 200
Send: [601, 700]
Recv: ACK 701, AdvWin 200
***This solution is correct. Even if the advertising window is 0 and all packets have been acknowledged, the sender will still attempt to send 1 byte of new data. See "RFC 793, section 'Managing the Window'" for further details. You will not be expected to know TCP to this level of detail on the exams, the "easy" solutions would probably suffice.

c) The first ack response is lost.

Send: [1, 200]
Send: [201, 300]
Recv: ACK 201, AdvWin 100 LOST
Recv: ACK 301, AdvWin 200
Send: [301, 500]
Recv: ACK 501, AdvWin 100
Send: [501, 600]
Recv: ACK 601, AdvWin 200
Send: [601, 700]
Recv: ACK 701, AdvWin 200

3.2 Distributed Systems

a) Consider a distributed key-value store using a directory-based architecture.

i) What are some advantages and disadvantages to using a recursive query system?

Advatanges: Faster, easier to maintain consistency.
Disadvantages: Scalability bottleneck at the directory/master server.

ii) What are some advantages and disadvantages to using an iterative query system?

Advatanges: More scalable.
Disadvantages: Slower, harder to maintain consistency.

b) **Quorum consensus:** Consider a fault-tolerant distributed key-value store where each piece of data is replicated N times. If we optimistically return from a put() call as soon as we have received acknowledgements from W replicas, how many replicas must we wait for a response from in a get() query in order to guarantee consistency?

We must wait for at least $R > N - W$ responses. If we have any fewer than this number, there is a possibility that none of our responses contain the latest value for the key we are requesting.

c) In a distributed key-value store, we need some way of hashing our keys in order to roughly evenly distribute them across our servers. A simple way to do this is to assign key K to server i such that $i = \text{hash}(K) \mod N$, where N is the number of servers we have. However, this scheme runs into an issue when N changes — for example, when expanding our cluster or when machines go down. We would have to re-shuffle all the objects in our system to new servers, flooding all of our servers with a massive amount of requests and causing disastrous slowdown. Propose a hashing scheme (just an idea is fine) that minimizes this problem.

We can treat the possible hash space as a circle, where every possible hash maps to some point on the circle. We then roughly evenly distribute our servers across this circle, and have each hash be stored on the next closest server on the circle. Then, when we add or remove servers, we need only move a portion of the objects on one server adjacent to the server we just added or removed. This technique is commonly known as **consistent hashing**.