Solutions to Test Prep #2

Covers: Udacity Lessons 6-8 and associated readings

Congestion Control and Streaming

1. What causes congestion collapse to occur?

Congestion collapse occurs when dropped packets and excessive queuing delays that result from congestion in turn further exacerbate the problem, which causes more drops and delays, and so on. Dropped packets cause retransmissions that add additional traffic to the congested path, while excessive delays can cause spurious retransmissions (i.e., a timeout occurs when the packet was merely delayed, not lost). Note that normal traffic that contributes to congestion is not the cause of collapse, it is the extra traffic that is caused by congestion that leads to collapse.

2. What is the difference between *fairness* and *efficiency* in a congestion control scheme?

Efficiency is how much of the available bandwidth is used, i.e., efficient congestion control leaves little or no bandwidth wasted. (Some definitions of efficiency may refer specifically to bandwidth used to do "productive work", thus excluding overhead traffic.) Fairness is how bandwidth allocated between the different flows. Two common definitions of fair are that all flows get equal throughput, or that all flows get throughput proportionate to their demand (i.e., how much they want to send).

3. Assuming traditional TCP Reno with AIMD behavior (i.e., the version presented in the lecture videos), suppose a TCP flow's bottleneck link has 1 Gbps capacity, and that link is not being shared with any other flows. What will the *average* throughput of that flow be, in megabits per second (Mbps)?

Additive increase will increase the throughput until it equals the bandwidth, at which point a packet loss will occur and trigger multiplicative decrease. At that point, throughput immediately drops to ½ the bandwidth. Additive increase then resumes, raising throughput linearly until it reaches the total bandwidth again. Thus the average throughput is the average of ½ bandwidth and 1x bandwidth = ¾ bandwidth. Therefore, the average throughput on a 1 Gbps link will be ¾ x 1 Gbps = 750 Mbps. (A more detailed approach may look at the area beneath the throughput curve, but this results in the same math since the additive increase is linear.)

4. What circumstances lead to the incast problem? (In other words, what factors must be present for incast to occur?)

The incast problem occurs when collective communication (i.e., many-to-one or many-to-many patterns) occurs on high fan-in switches. This results in many small packets arrive at the switch at the same time, thus causing some of the packets to be lost. The last necessary factor is a low-latency network, which means the timeout delay will be much more than the round-trip-time of the network. Consequently, large delays occur in which the system is simply waiting for the timeouts to occur. This slows the whole application, since hearing from all the senders in collective communication is usually necessary before the application can proceed. As a real-world example, suppose a web app has to query a back-end database and needs to check with 100 database nodes to do this. It needs to hear back from all 100 nodes before proceeding, or else it risks missing some of the results. (This is the implicit "barrier" that occurs in some data center applications that are not explicitly using barrier synchronization.) Because they are all responding to the same query, all the nodes will reply at roughly the same time. This means a high fan-in switch will have to handle many of these database replies at the same time. Such traffic bursts may cause only a few of these packets to be lost, while the rest are delivered. However, the application still cannot proceed until it receives replies from these few, so it waits. After a significant delay, retransmissions finally occur and may be delivered, allowing the application to proceed.

5. Suppose you are working on some live video call software (think Skype or Google Hangouts) and decide to build your application protocol on top of UDP (rather than TCP). Give as many different points as you can (minimum two) to help justify that decision.

- Latency is critical – retransmissions are pointless since they will arrive too late anyway
- Dropped frames aren't a big deal – the next frame will advance the video state before a retransmitted frame could arrive anyway
- Congestion control and flow control could cause unacceptable delays, as video frames get backed up in the sender host's local buffer (what is needed instead is for the application itself to reduce the frame rate that it tries to send)

<u>Reading – CUBIC TCP</u>

6. Why does the linear growth rate of TCP-RENO (1/RTT) perform poorly for short lived flows in networks with large bandwidth and delay products?

The time period required for the congestion window to reach its maximum value is very large (on the order of minutes and hours) for TCP-RENO in paths with large bandwidth delay

products. Short lived flows may never reach a congestion event, meaning the flow unnecessarily transmitted slower than necessary over its entire lifetime to avoid congestion.

7. Describe the operation of BIC-TCP's binary search algorithm for setting the congestion window. Once stable, how does BIC-TCP react to changes in available bandwidth, i.e. what happens when there is a sudden increase or decrease in available bandwidth?

At a high level, when BIC-TCP experiences a packet loss event, the congestion window value is set to the midpoint between last window value that did not suffer from loss (WMAX) and the previous window size that was loss free for at least one RTT (WMIN). This is often referred to as a binary search, as it follows intuitively that the maximum possible stable window value is somewhere between a value that was known to be stable and the value achieved just prior to the loss event. This algorithm "searches" for this maximum stable window value by effectively reducing the range of possible value by half per packet loss event.

Once this maximum stable window size has been achieved, if there is a sudden increase in available bandwidth, then max probing phase of BIC-TCP will rapidly increase the window beyond the value of WMAX until another loss event occurs, which resets the value of WMAX. If a sudden decrease in available bandwidth occurs, and this loss is below the value of WMAX, then the window size is reduced by a multiplicative value (β), enabling a safe reaction to a lower saturation point.

8. How does the replacement of this congestion control algorithm with a cubic growth function in CUBIC-TCP improve on BIC-TCP? Discuss.

CUBIC retains the strengths of BIC-TCP, but makes many improvements. First, BIC-TCP is a rather complex algorithm that approximates a cubic function. It's growth function has both linear and logarithmic elements, and many different phases (additive increase, binary search, max probing). Additionally, on short RTT and low speed networks, BIC-TCP's growth function can be too aggressive (recall it was designed to achieve high utilization on large bandwidth, long RTT networks), making it fairly unfriendly to other TCP flows competing for bandwidth.

CUBIC replaces the growth function in BIC-TCP with a cubic growth function, based on the elapsed time between congestion events. This function maintains the multiplicative decrease utilized by many TCP variants, but records the window size at a congestion event as WMAX. Using this value of WMAX , the cubic growth function can be restarted, with the plateau occurring at WMAX. This eliminates the need for multiple growth phases and maintaining

values like SMAX/MIN. The plateau of the cubic growth function retains BIC-TCP's stability and utilization strengths.

9. What is the purpose of the following regions of the CUBIC growth function:

a. Concave

The concave region of CUBIC's growth function rapidly increases the congestion window to the previous value where a congestion event occurred, allowing for a quick recovery and high utilization of available bandwidth following a congestion event.

b. Plateau

The plateau is also known as the TCP friendly region. In this region of the growth curve, the congestion window is nearly constant as it approaches and potentially exceeds WMAX. This achieves stability, as WMAX represents the point where network utilization is at its highest under steady state conditions.

c. Convex

The convex region of CUBIC's growth function exists to rapidly converge on a new value of WMAX following a change in available bandwidth. When the congestion window exceeds WMAX, and continues to increase throughout the end of the plateau, it likely indicates some competing flows have terminated and more bandwidth is available. This is considered a max probing phase, as the congestion window will grow exponentially in this region until another congestion event occurs and WMAX is reset.

10. How does CUBIC's fast convergence mechanism detect a reduction in available bandwidth (i.e. a new flow competing for bandwidth)?

When new flows start competing for bandwidth, other flows must release some bandwidth to maintain fairness. CUBIC employs the fast convergence mechanism to accomplish this. When two successive congestion events indicate a reduction in available bandwidth (i.e. a reduced value of WMAX), the new value of WMAX further reduced (based on the multiplicative decrease factor used for resetting the congestion window) to free up additional bandwidth and reduce the number of congestion events required for all flows to converge on a fair distribution of bandwidth.

Reading – TCP Fast Open

11. What kinds of web traffic stand to benefit most from utilizing the TFO option? How does TFO improve the performance of these flows?

Short lived TCP connections (small data sizes) on links with large propagation delays. The performance of these flows are dominated by the return trip time (RTT), and as such, the 3 way handshake used in standard TCP constitutes a large amount of overhead. By enabling the client and server to communicate some of the payload (data) during the 3WHS, it is possible to reduce the number of required RTTs for the flow to complete, reducing the RTT penalty incurred by the 3WHS.

12. Describe how a trivial implementation of TCP Fast Open (in which the server replies to a all HTTP GET requests with a TCP SYN-ACK packet with data attached) can be exploited to mount a source address spoof attack. How does TFO prevent this?

An attacker can send many HTTP GET requests for large resources to a victim server, spoofing a victim host address as the requestor. The victim server would then perform the expensive data fetch operations and transmit large volumes of data to a victim host. The result is a Denial of Service attack on both victims.

TFO prevents this by using an encrypted cookie that must be requested by the requestor before initiating requests. The server uses this cookie to verify that the requester address is not a forgery.

Reading - Multi Path TCP (MPTCP)

13. What threat do network middleboxes pose to negotiating MPTCP connections? How does the design of MPTCP mitigate this?

Network middleboxes may strip out unrecognized TCP options (flags) used during the 3-way handshake used to negotiate a MPTCP connection. This means that while the sender and receiver may both be MPTCP capable with multiple viable interfaces, a middlebox along the route may ultimately prevent a MPTCP connection.

MPTCP is designed to resort to a single path TCP when both ends of the connection cannot support MPTCP. In this case, when the sender's MPTCP capable flag is stripped out by a middlebox enroute to the receiver, the receiver thinks that the sender is not MPTCP capable and proceeds with a single path TCP connection.

The sender will see that traffic returning from the receiver is not MPTCP enabled (the flag is carried on all packets until acknowledged) and as such revert to single path TCP.

14. Why are receive buffer sizes required to be larger for MPTCP enabled connections? What controls does MPTCP put in place to maximize memory usage?

The receive buffer allows out of order data to continue flowing in the event a packet is dropped and must be resent. For a standard TCP connection, the required buffer size is determined by the bandwidth delay product of the connection.
With multiple subflows across a single connection present in MPTCP, the worst case scenario is that a packet drop occurs early and must be resent across the slowest link (like a 3G mobile connection). This would require other subflows (like high bandwidth WiFi connections) to have larger buffers than would be required if it were the only connection, because it can send data much faster than the slower link that is retransmitting the lost packet.

Reading - Dynamic Adaptive Streaming over HTTP

15. Explain two of the benefits of using HTTP as a streaming media delivery protocol. Make sure you explain the "why" as well as the "what".

Possible answers include:

- HTTP is built upon TCP, which provides several benefits to streaming media including
- TCP flows can easily traverse NATs and Firewalls
- Support for TCP is widely implemented and deployed, simplifying deployment and reliability
- HTTP based CDNs and caches are currently used at scale to serve content like web pages to load balance traffic, reduce load on origin servers, and reduces latency. If HTTP is used as a streaming protocol, these CDNs and caches are able to serve the content without modification.
- HTTP based streaming puts the client in charge of the session. This enables the client to select the initial content rate, or change this rate during the stream as a reaction to available bandwidth without negotiating with the server.

Additional answers may also be correct.

Rate Limiting and Traffic Shaping

16. Would you use a leaky bucket or a token bucket to traffic shape a constant bit rate (CBR) audio stream? Explain why.

Since a constant bit rate stream isn't bursty, the traffic shaping mechanism doesn't need to handle bursts. Since the original stream is "smooth", it would be better to use the leaky bucket to keep the stream "smooth" and even out any bursts.

17. If you want to traffic shape a variable bit rate (VBR) video stream with average bit rate 6 Mbps and maximum bit rate 10 Mbps, should you use a leaky bucket or a token bucket? What values should you use for rho and beta if you want to allow bursts of up to 500 ms?

Since a variable bit rate stream has bursts, it is better to use a token bucket that will allow short bursts, but even things out to the average bit rate of the stream in the long run. Rho is the rate of tokens being added to the bucket, so it should match the average bit rate: rho = 6 Mbps. Beta determines how large and how long a burst is allowed. Since we want to allow up to 10 Mbps bursts for up to 500 ms (0.5s), we should allow (10 – 6 Mbps)(0.5s), or beta = 2 Mb = 250 kB (or 245 kB). (Note: b = bit; B = byte.)

18. Suppose you're running an ISP and get the crazy idea that implementing Power Boost for your own network would be a good idea. For the 6 Mbps service plan (i.e., customers can have a sustained rate of 6 Mbps), you'd like to allow them to burst up to 10 Mbps for up to 10 seconds. In megabytes (MB), what should you set the beta parameter of your token bucket to? (Round to the nearest tenth of a MB, i.e., one decimal place.)

Similar to the last problem, (10 – 6 Mbps)(10s) = 40 Mb = 5 MB (or 4.77 MB)

19. Read about the following two Active Queue Management (AQM) techniques: Random Early Detection (RED) and CoDel. Although they vary in specifics, these two algorithms share a common basic approach to solving the buffer bloat problem. Explain what that approach is and why it works.

Their approach is to drop packets even when their buffers are not full. RED determines whether to drop a packet statistically based off how close to full the buffer is, whereas CoDel calculates the queuing delay of packets that it forwards and drops packets if the queuing delay is too long. By dropping packets early, senders are made to reduce their sending rates at the first signs of congestion problems, rather than waiting for buffers to fill.

20. If you want to find out if a remote host (i.e., not *your* server) is currently under a DoS attack, would you use active or passive measurement? Explain why.

Active measurements, such as ping, are required here. Only the server's owner or ISP would be able to use passive measurements, since they control the machines over which the server's traffic is handled. Excessive ping delays to the server are a sign of congestion on the server's

link. (It's hard to be sure that it's due to a DoS attack without additional context, but it's a sign that something is wrong...)

21. If you want to compute the traffic intensity, I=La/R, on a router interface (i.e., the ratio between arrival rate and forwarding rate), would you use Counters, Flow Monitoring, or Packet Monitoring? Explain why.

The sending rate is a known quantity (it's just the maximum rate of that device's interface). The average length of packets and the average arrival rate of the packets can be determined from simple counters. (We do not need to inspect the packet contents, so packet monitoring is unnecessary. Since we are only concerned with all packets on a particular interface and do not care about which flow each packet belongs to, flow monitoring is also unnecessary. However, if you knew that traffic intensity was high and wanted to determine which source is responsible for most of the traffic, flow monitoring would come in handy in that case.)

<u>Reading – Sizing Router Buffers</u>

22. Discuss the drawbacks to over-buffering routers. If memory is widely available at low cost, why is it a bad idea to use massive buffers to ensure high link utilization?

Using massive buffers in internet routers increases the size, power consumption, and design complexity of routers. Large buffers are typically implemented in off chip DRAM, where small buffers can be implemented on chip.

Additionally, large off chip DRAM is slower to retrieve data than on chip SRAM. This means that retrieving buffered packets takes longer, which means the latency on the link will grow. During periods of congestion with a large amount of buffered packets, latency sensitive applications like live streaming and networked video games will suffer.

Further, TCP congestion control algorithms can also suffer under these conditions. Using large amounts of cheap memory may eliminate the need to worry about proper buffer sizing, but it induces hardware efficiency issues and presents problems for low latency applications.

23. Under what conditions was the "rule-of-thumb" for buffer size ($B = RTT \times C$) originally conceived? How does this fundamentally differ from current, real world conditions?

The "rule-of-thumb" is derived from an analysis of a single long lived TCP flow. The rate is designed to maintain buffer occupancy during TCP congestion avoidance, preventing the bottleneck link from going idle.

These conditions are not realistic compared to actual flows in backbone routers. For example a 2.5 Gb/s link typically carries 10,000 flows at a time, of which the life of the flow varies. Some flows are only a few packets, and never leave TCP slow start, and hence never establish an average sending rate.

Of the flows that are long lived, they have various RTTs and their congestion windows are not synchronized, which contrasts directly with a single long lived flow with a stable RTT and single congestion window.

24. Statistical modeling of desynchronized long lived flows indicates that smaller buffer sizes are sufficient to maintain link utilization as the number of these long lived flows increases. However, not all flows can be expected to be long lived. Discuss why short lived flows (less than 100 packets) do not significantly detract from these findings.

Even when the vast majority of flows across a link are short lived, the flow length distribution remains dominated by the long lived flows on the link. This means that the majority of the packets on the link at any given time belong to long lived flows.

Required buffer size in the case of short lived flows depends on actual load on the links and the length of the flows, not the number of flows or propagation delays. This means that roughly the same amount of buffering required for desynchronized long lived flows will also be sufficient for short lived flows as well.

Reading – Controlling Queue Delay

25. Explain how standing queues develop in network buffers at bottleneck links. Why is a standing queue NOT correctly identified as congestion?

Queues develop at bottleneck links as a result of the bottleneck's reduced forwarding speed. As some of the packets in the queue are forwarded, the TCP sender will begin to receive ACKs and send more packets, which arrive at the bottleneck link buffer, refilling the queue. The difference in the bottleneck link speed and the link RTT (driving the congestion window of the TCP flow) will result in a certain number of packets consistently occupying the buffer, until the flow completes, which is referred to as the standing queue.

Standing queues are NOT congestion because it results from a mismatch in congestion window and the bottleneck link size. A standing queue can develop in single flow environments, and under usage limits that would eliminate actual congestion.

26. Consider the CoDel active queue management algorithm. How does the algorithm decide whether or not drop a flow's packets? What effect does dropping the packet have on the TCP sender?

CoDel assumes that a standing queue of the target size is acceptable, and that at least one maximum transmission unit (MTU) worth of data must be in the buffer before preventing packets from entering the queue (by dropping them). CoDel monitors the minimum queue delay experienced by allowed packets as they traverse the queue (by adding a timestamp upon arrival).

If this metric exceeds the target value for at least one set interval, then packets are dropped according to a control law until the queue delay is reduced below the target, or the data in the buffer drops below one MTU.

Dropping a flow's packet triggers a congestion window reduction by the TCP sender, which helps to eliminate buffer bloat.

Content Distribution

27. If your browser has a page in the cache and wants to know if the page is still fresh and can be used, or is too old and should be replaced with fresh content from the server, which HTTP method should it use to find out?

(If you are familiar with the If-Modified-Since header field, which we have not discussed in this class, please assume that we are not using If-Modified-Since.)

The HEAD method in HTTP requests a document just like the GET method except that the server will respond to a HEAD request with only the HTTP response header; the response body (which would normally contain the document data) is not included. This saves the delay of transmitting the actual document, e.g., if it is a large file, but allows the browser to check the Last-Modified field in the response header to find out if it's been changed since the time when the cached version was retrieved.

28. Consider the HTTP protocol. What will cause a server to send a response message with the status code...

a. 404 Not Found ?

The requested file does not exist on the server. That is, the file indicated by the path part of the GET method line cannot be found at that path.

b. 302 Moved Temporarily (also sometimes called 302 Found) ?

The requested file is not at this location (i.e., the path part of the GET method line), but the browser should instead use the URL provided in the Location field of the response to retrieve the file. However, the file may be found at this location in the future (unlike a Moved Permanently response), so the URL in the Location field should be used this once, but not necessarily again in the future.

c. 200 OK ?

The operation in the request message succeeded. What that operation is exactly depends on the request method. For example, if the request method was GET then 200 OK means that the document was retrieved and its content should be in the body of the 200 OK response. (200 OK responses to other methods do not necessarily contain a body, though. This also depends on what the method was.)

29. Consider the HTTP protocol. What would the following header fields be used for?

a. Last-Modified

This is the date and time that the requested document file was last modified on the server. It can be used to check if a cached copy is fresh (newer than the Last-Modified time) or stale (older than the Last-Modified time, indicating that it's been changed since the cached copy was retrieved).

b. Host

This is the domain name of the web request (e.g., from the domain part of the URL). One way this may be used is if a single web server (with a single IP address) is hosting websites for more than one domain. The web server can check the Host field to see which domain's pages should be retrieved for each request it gets.

c. Cookie

This is included in request messages that are sent to a domain that previously gave the browser a cookie. That cookie would have been provided by the Set-Cookie field in a response message, and after that (until the cookie expires) the browser should include the exact same cookie given by Set-Cookie in any request message it sends to the same domain. This allows the server to know that a request is coming from the same client

that made another earlier request. For example, when you request to view your shopping cart, the web server may use cookies to know that you are the same person who earlier clicked on an item to add to your cart, so it can show you a cart containing that item.

30. Of the various methods to redirect web requests to CDN servers, DNS redirection is the most common. Why would this method be more popular than the alternatives?

DNS-based redirection is much faster than HTTP redirection, as the latter requires a couple extra round trips to servers. (It's actually more than just one extra round trip because you need to establish a TCP connection to a second different server.) It also gives the CDN provider more control over who will be redirected where than a technique like IP anycast would. Finally, it is not too difficult to implement (even if slightly more complex than the other two) and it uses tools that are widely supported (i.e., DNS) and do not need any modifications to support this technique (i.e., DNS works out of-the-box).

31. How does BitTorrent implement the tit-for-tat algorithm? Be sure to explain in detail, including the roles of both choking and unchoking.

A BitTorrent client sends data only to the top N peers who are sending to it, plus one peer who is optimistically unchoked. Let's say for example purposes that N=4. Your BitTorrent client will choose the 4 peers who are sending to it at the fastest rate and it will send data to them in return. It will not send to other peers, and they are said to be choked. Thus it provides tit-for-tat by sending to those who send the most to it, and choking those that are not sending to it, or are sending slowly.

However, this creates a problem where two peers who might be able to send to each other are mutually choked. Neither will begin sending to the other because the other is not sending to it. Therefore, each client will optimistically unchoke one peer at any given time for a brief period. If the client sends fast enough to the optimistically unchoked client to get on its top-4 then the peer will send data back in return. If the client receives enough data from the peer for it to be in the top-4 then that peer becomes one of the new top-4 and the slowest of the previous top-4 will be choked. Thus they both end up in each other’s top-4. (The peer is no longer "optimistically" unchoked, and is merely unchoked. A new peer is selected to be optimistically unchoked.)

On the other hand, if the client does not get into its peer's top-4, or if it does but the peer does not send fast enough in return to get in the client's top-4, then they will not end up in each

other's top-4. After some time, the client will stop optimistically unchoking that peer and stop sending to it. It will choose a new peer to optimistically unchoke.

This process repeats forever (until the client has the entire file, that is) in order to keep

exploring different peers for better matches than the client's current top-N. The game theoretic result is that clients will end up sending to peers that are able to send back about the same amount – fast peers will get paired up, while slow peers are matched with each other. This happens because a fast peer will readily drop a slow peer from its top-N in favor of another fast peer, matching fast peers together. Slow peers will not get matched with fast peers because the fast peers will soon learn to choke them, but they will pair up with other slow peers because neither peer can find a better match who is willing to unchoke them.

32. In a distributed hash table, how many steps (hops) are required to lookup an item if the finger table is a constant size (i.e., its size does not depend on the number of DHT nodes in the system)? Explain why that is the right answer.

A lookup will require O(N) hops in this case. Suppose a constant size of 1, as an example. Each node only knows how to find the next one, so it basically forms a ring topology. In the worst case, the requested item is on the last node in the ring before getting back to the node that originated the request. So the request has to go all the way around the ring, taking N-1 hops. Based on similar reasoning, if a larger, constant number of nodes is in the finger table, a proportionately smaller amount of time may be required. However, for any given constant size finger table, as the number of nodes in the system grows, the number of hops required will still be on the order of O(N).

33. For a more typical DHT setup where the finger table has O(log N) entries, for N DHT nodes in the system, explain why the number of steps to access an item is O(log N).

O(log N) entries in the finger table means that each node knows about the node halfway around the ring back to it, about the node halfway to that one, the one halfway to that one, and so on until the last entry in the finger table that is just the next node. This means that for any given item that could be on any node, each node knows the address of at least one node that is at least half way around the ring from itself to the item. Since each hop cuts the distance to the item in half, the number of hops required to get to the item from any starting point in the DHT is O(log N). (This should be understood by analogy to binary search, divide-and-conquer, etc.)