CNT 5106C Computer Networks Fundamentals, Spring 2020

Instructor: Prof. Ahmed Helmy

Homework 2: Application Layer Continuation and Transport Layer

Due Date: Feb 25th, through Canvas

Instructions: Be precise and to the point. Many questions require answers using a sentence or two). Some questions will ask you to elaborate, use visual aids or graphs, or show traces/code. Use your own words and phrases, do not copy from any other source.

Q1- DNS-related: Consider a scenario of a user browsing the web from the machine storm.cise.ufl.edu, accessing an article in a website at URL:

www.nytimes.com/2019/09/26/technology/ai-computer-expense.html

The user performs three accesses: - using http, - using https and - using port 8080.

A. Show the sequence of DNS servers queried to resolve the URLs (assume no caching)
B. Write the complete URL for each of the three accesses [you may have to edit the URL]
C. Write the four tuple [src address, src port, dst address, dst port] for each of the accesses

Answer:

A. First, the end machine contacts the local DNS server (at ufl.edu, or even cise.ufl.edu if there is one). Second, since there is no cache the root DNS server is contacted to get the server with info about '.com'. Third, the top-level domain (TLD) server for '.com' is contacted to get information about 'nytimes.com'. Fourth, the authoritative server for 'nytimes.com' is contacted to resolve the IP address for www.nytimes.com.
   In all cases (http [port 80], https [port 443], or port 8080) the same IP address is used for www.nytimes.com, so the sequence of DNS servers queried will not be different.

B. For http, it is very simple just add "http://" in front of the URL above, same for "https://"
   To add a port number (different from the default) we need to added it explicitly in the form 'http://www.example.com:8080/path', so in our case it is:
   http://www.nytimes.com:8080/2019/09/26/technology/ai-computer-expense.html

C. We need to get four values: source IP address, source port number, destination IP address and destination port number.
   The easiest is the destination port number: http is 80, https is 443, and the 3rd one is 8080
   Then the source port is a random number (r) generated by the host: r_1, r_2, r_3
   The source IP address is that of storm.cise.ufl.edu, and the destination IP address is that of www.nytimes.com. To get these addresses you can use a networking tool (such as ping, traceroute, nslookup, ip address, etc.).
   The IP address for www.nytimes.com is: 151.101.5.164
   The IP address for storm.cise.ufl.edu is: 128.227.205.236
   Hence, the 4-tuples become:
   http: [128.227.205.236, r_1, 151.101.5.164, 80]
   https: [128.227.205.236, r_2, 151.101.5.164, 443]
   port 8080: [128.227.205.236, r_3, 151.101.5.164, 8080]

Q2- Discuss how the following technologies (or their variations) help in improving the performance of content distribution networks (CDNs):

A. HTTP
B. DNS

Answer: A. HTTP variant, 'DASH' for dynamic adaptive streaming over HTTP, adds a level of indirection, and provides a manifest file with pointers to different files (each containing an encoding for the streaming media, corresponding to different resolutions at different data rate). The client monitors the quality periodically and picks the file (from the manifest) that provides the best quality at that time. This allows CDNs to account for the Internet's heterogeneity, and scale better by not needing to transmit at higher rate than the clients (and their connections) can handle.

B. DNS is used to provide a level of indirection for services that are using a 3rd party host (or cloud service) for their streaming. Instead of the DNS authoritative server returning an IP address to the video server, instead it provides a URL to the 3rd party's website and path. This way the information about the 3rd party details can be hidden from the user until they access the video, providing flexibility to change the hosting arrangements in a way that is transparent to the user.

Q3- Elaborate on the data '*push*' vs '*pull*' in the context of

A. http vs SMTP
B. peer-to-peer network hierarchy communication (e.g., super nodes, group leaders)
C. Proxy and web caching
D. CDNs

Answer: A. http uses data pull as it relies on the client initiating the request in order to transfer the data from the server. [Extra: this may lead to unnecessary data pulling from the data-origin servers, and can be mitigated using web caching].
SMTP, on the other hand, uses data push from the email sender to its email server, then another push to the intended recipient's email server. The last hop (from the recipient to its email server) pull is used to get the data to the email agent/reader. [Extra: this may lead to recipients inboxes getting unsolicited emails (spam), which can negatively affect user experience, but also wastes a lot of the network capacity in sending useless data. This can be partially mitigated using spam filters].

B. In P2P hierarchies involving super-nodes (like Skype), group leaders or cluster heads, the goal is to attempt to reduce the search overhead and delay. One example and commonly used practice is to push information (about existing resources, connected users [in skype], or files/chunks) from the group members to their group leader. When a search query is triggered by a user, it only needs to ask its group leader, and if it doesn't have the information, then other group leaders. So by using a proactive push to the group leaders, the pull (to answer search query) becomes faster and easier. [Extra: imagine we also push information between group leaders about their group members, this would add overhead initially, but the search would be even easier to get pointers to the existing information as only one group leader needs to be pulled for data. That's a trade-off between pushing and pulling.]

C. Proxy and web caching: the first time an object is requested it is *pulled* from the data-origin server. If the object is 'cachable' then it is cached in the proxy and is locally *pulled* next time a similar request is sent. This saves network bandwidth and reduces user delay. In case of encrypted or dynamic data, customized or copyrighted content, then the object cannot be cached.

D. CDNs: The duplication/copying of content (e.g., videos) by the distributor/company on the distributed servers/cluster in a CDN can be costly (taking up space/storage, e.g., with various encodings per movie, and bandwidth). So the CDN companies perform a pattern analysis, and attempt to replicate the content where and when it is popular, to increase the hit ratio, while keeping the cost of replication down. Pushing the popular data/movies to where it is likely to get accessed saves network bandwidth, reduces delays for the users, and utilizes storage efficiently. For non-popular content the pull model is used where the request to sent to the origin servers to complete the request.

Q4- Someone suggested to use a local file called *hosts.txt* on each machine instead of DNS. Discuss the advantages (at least 2) and disadvantages (at least 2) of such suggestion.

Answer: advantages: 1- removing reliance of any 3rd party server (including DNS servers), along with any vulnerability such servers can be subject to, 2- reducing delays in name resolution as the name to address mapping is done at the local machine, avoiding DNS delays, and 3- having control (or at least clarity) over the machines accessed (since the mapping can be found in the local file rather than it being hidden by layers of DNS servers.

disadvantages: 1- any change to the name-to-IPaddress mapping (due to newly created companies, mergers, or closures) would have to be reflected in the hosts.txt files, by actually changing the files on all the hosts. This doesn't scale, 2- dynamic mapping needed for load balancing may not be easily implemented, and 3- dynamic mapping due to content customization (e.g., based on locale) becomes harder to implement, as the information has to be pushed to all the hosts, instead of it being pulled from a few servers that can be updated easily.

Q5- What is '*saw-tooth*' behavior in TCP, and what is causing it?

Answer: TCP uses additive increase multiplicative decrease (*AIMD*), increasing cwnd by '1' for every RTT when there is no packet loss, and decreasing the window to half with every packet loss. The slow (additive) increase, followed by fast (multiplicative) decrease in the window size, results in this saw-tooth behavior in the window size, and subsequently in the TCP throughput.

Q6- A TCP flow and a UDP flow walk into a network, sharing a bottleneck link …. Complete the story, detailing the packet rate dynamics if the link gets congested, and comment on the end result.

Answer: TCP performs congestion control and responds to packet loss by reducing its window size, effectively cutting down its throughput and share of the bottleneck link. UDP on the other hand does not slow down and can effectively squeeze-out the TCP flow by forcing more losses over the bottleneck link, which slows down the TCP flow even further. The end result is that the UDP flow wins the major share of the bottleneck link and the TCP flow loses by having close-to-zero throughput. This is under the assumption that UDP rate is higher than the bottleneck link and

that the UDP flow is not blocked or rate-limited by firewalls.

Q7- TCP is supposedly fair, dividing the bandwidth between competing TCP flows. You want to transfer a huge file fast, suggest a way of doing so using TCP to get over the fairness delays, and approximate your new bandwidth share.

Answer: By using parallel TCP sessions, each sending a chunk of the file. If every session gets R/M rate (where R is the capacity of the bottleneck link and M is the number of flows crossing that link), then by using N parallel flows instead of one, an end host can get NR/(N+M).

[Extra note: TCP's rate (or bandwidth) depends on the congestion window and RTT [TCPrate = *cwnd* / RTT Bytes/sec].
When different flows with different RTTs share a given bottleneck link, TCP is only RTT-fair i.e. low-RTT flows will get a higher share of the bandwidth that high-RTT flows [*]

* Pawan Prakash, Advait Dixit, Y. Charlie Hu, and Ramana Kompella. 2012. The TCP outcast problem: exposing unfairness in data center networks. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* (NSDI'12). USENIX Association, Berkeley, CA, USA, 30-30.]

Q8- Given that most data-link layers perform error checking (and correction to some extent) why do we need checksum in UDP (and TCP)?

Answer: there two main reasons:

1- As a safety precaution in case the data-link layer does not perform error checking
2- Packet/bit corruption can occur at any layer. For example, even if the packet does not incur loss on the wire (which the data-link layer error-detection should take care of), corruption of bits can occur in the memory/storage/buffer after the packet is processed and passed up to the network and transport layers.

Q9- Comment on response to *packet-loss* vs *ack-loss* in

A. Go-back-N
B. selective repeat

Answer: A. Go-back-N uses 'cumulative acknowledgements' so when an 'ack' is lost (say ack *x*, acknowledging packet *x*), the next packet (*x*+1) will trigger ack *x*+1 which acknowledges everything up to (and including *x*+1). So ack-loss in Go-back-N will not trigger any re-transmissions from the sender (if the next packet is ack'ed).
This is a very different behavior than packet loss in Go-back-N.

B. Selective repeat does not use cumulative acknowledgements (unlike Go-back-N). Hence, receiving ack *x*+1 means that packet *x*+1 was received, but does not indicate anything about packet *x*. In this case, the loss of packet *x* will produce the same behavior (at the sender) as does the loss of ack *x*.

Q10- Congestion Signaling:

A. What is meant by implicit congestion signaling and explicit congestion signaling? Give

examples of congestion control protocols that use each type signaling.

B. Discuss the advantages and disadvantages of each of the above schemes.

C. What kind of signaling does TCP use to detect network congestion? Explain the different signals that TCP uses for that task.

Answer: A. *implicit* congestion signaling (as in TCP Tahoe, Reno, SACK or Vegas) uses inference based on measurements from the edges/ends of the network, such as delays, round trip times, gaps in sequence numbers, or duplicate acks, to attempt to detect congestion without aid from the network. On the other hand, *explicit* congestion signaling requires (and relies on) a signal from the network nodes (i.e., routers) when congestion occurs. It can use congestion thresholds on queue average size (and variance) to sense the onset of congestion, and sends a signal to the end hosts indicating network congestion (in the form of a bit, rate, or window credit, depending on the protocol). Examples include TCP ECN (with explicit congestion notification bit/field), ICMP source quench, or ATM ABR (available bit rate) service. (one example is enough)

B. Implicit signaling:
i- advantage: operates end-to-end without any change to the network, adhering to the end-to-end design principle of the Internet, allowing complex functionality to be performed at the edges (rather than the network), and facilitating evolution and innovation at the edges.
ii- disadvantage: since it attempts to estimate network congestion, there a higher chance of error (than in explicit signaling). For example, packet loss could be due to channel bit error rate (BER) and not congestion, misleading protocols to cut down their rate when it's unnecessary.

Explicit signaling: i- advantage: there is no 'guessing' (or inference) involved at the edges (unlike implicit signaling). Rather, the edges get a clear signal from the network, based on in-network measurements, that congestion is occurring. Sometimes the signal also indicates the level of congestion and suggests a rate for the senders.
ii- disadvantage: it requires deployment in the network (which is a major undertaking, unlikely to occur in reasonable time in the Internet). It also increases the processing complexity in the routers to estimate the onset of the congestion and to send signals to sources who may be the cause of such congestion.

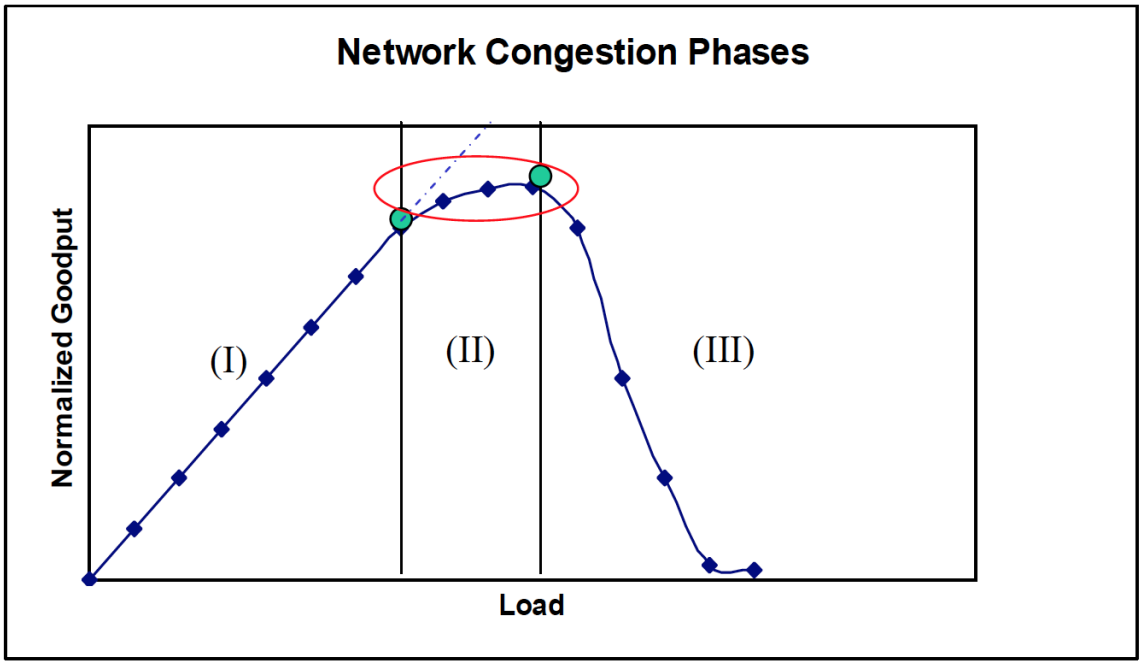
C. TCP uses implicit congestion signaling at the edges. It uses sequence numbers for packets and ack's, along with the timer mechanisms (for time out) based on estimated round trip times (RTT). The sender looks for gaps in sequence numbers (or duplication) in the acks to infer packet loss, and perform retransmission and window adjustments. [Extra: note that TCP Tahoe, Reno, NewReno, and SACK, do not use the RTT estimates to adjust the window size directly. TCP Vegas attempts to estimate congestion levels based on RTT estimates/variations, and adjust the window size accordingly (not discussed in class)]

Q11- Network congestion phases:

A. Describe (with the aid of a graph) the different phases of network load/overload outlining the degrees of congestion with increase of load. Indicate the point of congestion collapse and explain why it occurs.

B. Where does TCP operate on that graph? Explain for the various phases of TCP; slow start, congestion avoidance (due to timeout), fast retransmit-fast recovery triggered by duplicate ACKs.

Answer: A.



(I) No Congestion
(II) Moderate Congestion
(III) Severe Congestion (Collapse)

- no congestion --> near ideal performance
- overall moderate congestion:
  - severe congestion in some nodes
  - dynamics of the network/routing and overhead of protocol adaptation decreases the network Tput
- severe congestion:
  - loss of packets and increased discards
  - extended delays leading to timeouts
  - both factors trigger re-transmissions
  - leads to chain-reaction bringing the Tput down

B. For TCP: in slow start, the load starts from *cwnd*=1 (at the beginning of phase I), then ramps up quickly (exponential growth of *cwnd*) until a loss is experienced (in phase II or beginning of phase III).
After the loss, if a timeout occurs, TCP goes down to *cwnd*=1 (at the beginning of phase I) then ramps up to roughly half the load the led to the loss (i.e., half way in phase I).
In congestion avoidance *cwnd* increases linearly, which means the load increases slowly towards the end of phase I and into phase II, until another loss occurs.

In fast retransmit fast recovery (due to duplicate acks), the load is cut in half (half way into phase I), then slow (linear) increase towards phase II (as in congestion avoidance).

Q12- Argue for or against this statement (reason using examples as necessary): “Packets are lost only when network failures occur (e.g., a link goes down). But when the network heals (e.g., the failed link comes back up again), packets do not get lost.”

Answer: discussed in class and relates to the animation videos shown in class.
When the network fails (e.g., a link goes down), a number of packets that cross that link may be lost. When the network heals, packets/flows may cross relatively shorter paths to get to the destination. Shorter paths have a “*delay-bandwidth* product” less than longer paths, and hence can hold fewer bytes in the pipe (i.e., fewer segments can be in flight). Having a TCP connection with a high CongWin (*cwnd*) go over a shorter path may also cause packet loss, since the shorter paths will get congested, and buffers may overflow causing multiple (sometimes severe) packet losses.

Q13- Compare and contrast *AIMD* vs *MIMD*. Focus on network stability.

Answer: AIMD is ‘additive increase multiplicative decrease’, and is the current algorithm used by TCP congestion control to adjust its window size. When there is no congestion, and there is capacity to increase the window, this is done cautiously and slowly using an ‘additive’ equation *cwnd*+=1 for every RTT. When there is congestion (as indicated by a packet loss and timeout) then the window size is cut down aggressively using the multiplicative equation *cwnd=cwnd*/2.
This *stabilizes* the network and provides enough time for network recovery after congestion.

MIMD is ‘multiplicative increase multiplicative decrease’, where aggressive adjustment of the window size is adopted in both cases (of congestion and lack-of-congestion). Example multiplicative increase equation could be *cwnd=cwnd**2. This leads to *oscillations*, as the TCP sources increase their window size aggressively leading to congestion, then they cut down their window sizes aggressively in the face of congestion, which causes congestion relief, only to increase the window size quickly again multiplicatively. This *destabilizes* the network and causes performance degradation.

[A *lab* part will be added to hwk2 later on using tracing tools as relates to this homework.
Instructions will be posted separately for the lab part.]