EE122 Fall 2013 HW3

Instructions

Record your answers in a file called *hw3.pdf*. Make sure to write your name and SID at the top of your assignment. For each problem, clearly indicate your **final answer**, bold and underlined. You do not need to explain your answer unless explicitly told to do so. Submit your answers via your instructional account using submit hw3. This assignment is due on **Dec 2nd, 5pm**.

Questions? Post on piazza, or email Radhika (radhika *at* eecs.berkeley) or Sameer (sa *at* berkeley)!

Q1. DNS Lookup

Your EECS instructional machine *(@fuji.cs.berkeley.edu)* has a utility called dig that allows you to query Domain Name Service (DNS) servers around the Internet. You can simply run dig as follows:

```
$> dig @dns.server.name[optional] record-type domain-name
```

Here dns.server.name is the hostname of the DNS server you wish to query. The record-type is the type of DNS record you wish to retrieve (such as ANY, MX, A, CNAME etc.) and domain-name is the name of the host or domain you seek information on. For more information on how to use dig, please consult the man page and/or RFC1035.

As we discussed in class, there can be 2 types of DNS queries—recursive or iterative. For a recursive DNS query, the name server attempts to completely resolve the name by following the naming hierarchy all the way to the authoritative name server. On the other hand, for an iterative query, the name server simply gives a referral to another name server in the hierarchy that should be contacted next in order to resolve the name. By default, dig sets the RD (recursion desired) bit in the DNS query packet to indicate that it would like to have the query resolved recursively. Not all servers support recursive queries from arbitrary resolvers.

a) **Basic Usage:** Resolve the IP address of www.google.com using your default name server. Attach the output of the dig query and explain each field of the ANSWER SECTION in the output

   (Actual IP addresses may be different)

dig query output

```
; <<>> DiG 9.8.3-P1 <<>> www.google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4254
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.google.com.                IN      A

;; ANSWER SECTION:
www.google.com.         54      IN      A       74.125.236.80
www.google.com.         54      IN      A       74.125.236.83
www.google.com.         54      IN      A       74.125.236.82
www.google.com.         54      IN      A       74.125.236.84
www.google.com.         54      IN      A       74.125.236.81

;; Query time: 13 msec
;; SERVER: 192.168.0.1#53(192.168.0.1)
;; WHEN: Fri Dec  6 09:01:47 2013
;; MSG SIZE  rcvd: 112
```

Google's IP address: 74.125.236.80

Fields in the answer section:

- First (www.google.com): is the domain name being returned (NAME).
- Second (54): is the TTL.
- Third (IN): is the CLASS and stands for Internet.
- Fourth (A): is the TYPE and stands for a mapping a domain name to an IPv4 address.
- Fifth (74.125.236.80): is the the IP Address.

b) **Iterative DNS Queries:** Now, instead of using your default name server, we would like you to resolve www.google.com using one of the root DNS servers (*e.g.,* a.root-servers.net). Note that this server doesn't accept a recursive query from you, but it gives you the reference to the next name server that is to be queried in www.google.com's hierarchy. Use dig to execute the sequence of iterative queries by following the chain of referrals to obtain www.google.com's address[1]. Attach the dig output and write down the sequence of name servers you queried in the process and specify the domain each server was responsible for.

$> dig @a.root-servers.net www.google.com (domain: .)

[1] If a query returns multiple servers, just choose one amongst them at random. If a particular server times out,

$> dig @m.gtld-servers.net www.google.com (domain: .com)
$> dig @ns2.google.com www.google.com (domain: google.com)

c) **Recursive DNS Queries:** ns1.iitkgp.ac.in and nsl.fujitsu.fr are two name servers in India and France that answer recursive DNS queries. Use these DNS servers to resolve www.google.com. Attach the dig output. Now compare the end-to-end latencies between your machine and the sets of IPs for google.com that were returned by these two DNS servers and by your default DNS server in part (a). Why do we see a difference in latency?

$> dig @ns1.iitkgp.ac.in www.google.com (IP1)
$> dig @nsl.fujitsu.fr www.google.com (IP2)

$> ping IP1
$> ping IP2
RTT of India's IP (IP1) > RTT of France's IP (IP2) > RTT of IP in part (a)

DNS supports IP anycast. The name servers will always return the IP address of the nearest server, which in this case might be located in different parts of the world.

d) **DNS Spoofing:** As the evil CTO of www.evilsearch.com, Garry realizes that the easiest way to defeat Google would be to redirect all the users to his website by simply messing with the DNS servers. Having taken EE122, he recalls that servers cache A and NS records from DNS replies and referrals and he can configure his own DNS server to return incorrect results for arbitrary domains. If the resolver caches Garry's malicious results, it will return bad results to future DNS queries. Help him complete his master plan to hijack Google's domain name by writing down exactly what would Garry's name server returns upon a DNS query. What must a robust DNS server implementation do to counter this attack?

When Garry's name server receives a query for www.evilsearch.com, it returns the following malicious results:

www.evilsearch.com long-TTL in NS ns1.google.com
google.com long-TTL in NS ns1.google.com
ns1.google.com long-TTL in A w.x.y.z (Garry's DNS server)

If a DNS server blindly caches everything, it will redirect all future queries for www.google.com to Garry's nameserver (w.x.y.z).

A robust DNS server implementation should be less trustful of results returned by other DNS servers and only cache information that's directly relevant to the queried domain. In the above example, since google.com is not a subdomain of evilsearch.com, a correct DNS server implementation should ignore all information related to google.com in the results.

Another option is to use something like DNSSec, which allows the query response to be authenticated.

Q2. Content Delivery with HTTP

We would like to download an article on the web. This involves two steps:
Step 1: Download a master index page of size B. This page contains links to four images.
Step 2: Download the four images. Each image is also of size B

Assumptions:

- Ignore all transmission delays
- Assume that HTTP responses fit in a single TCP packet. i.e., the HTTP response including the B bytes and the size of the HTTP header is smaller than the MSS of a single TCP packet
- We don't need to wait for the HTTP responses to be acknowledged nor for the TCP connections to terminate

Calculate the time taken to download the article under each of the following scenarios

a) All files are stored at a single origin server S. Our RTT to S is 'R'. We use sequential requests with non-persistent TCP connections. Give your answer in terms of multiples of 'R'.
(2R + 4(2R)) = 10R

b) All files are stored at a single origin server S. Our RTT to S is 'R'. We use sequential requests with *persistent* TCP connections. Give your answer in terms of multiples of R.
(2R + 4R) = 6R

c) All files are stored at a single origin server S. Our RTT to S is 'R'. We use concurrent requests with non-persistent TCP connections. (Note that the master index page is downloaded first, followed by concurrent download of the images). Give your answer in terms of multiples of R.
(2R + 2R) = 4R

d) The content provider now signs up with a CDN for the delivery of its images. The master index page is still served from the origin server S with RTT equal to 'R'. Our RTT to the closest CDN server is (R/2). We use sequential requests with non-persistent TCP connections. Give your answer in terms of multiples of R.
(2R + 4(2R/2)) = 6R

e) Repeat part (d) assuming sequential persistent connections
(2R + R/2 + 4R/2) = 4.5R

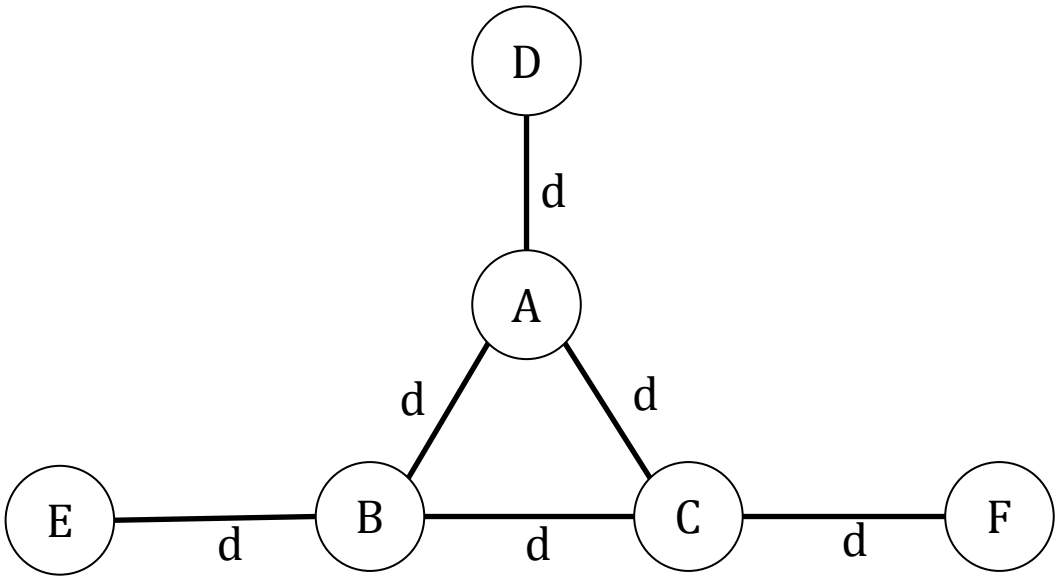
f) Repeat part (d) assuming concurrent non-persistent connections
(2R + 2R/2) = 3R

g) The CDN tries to do some load balancing and different images are served by different CDN servers, at distances 3R/4, R/2, R/3 and R/4. The master index page is still served from the origin server S. We use sequential requests with non-persistent TCP connections. Give your answer in terms of multiples of R.
2R + 2(3R/4) + 2(R/2) + 2(R/3) + 2(R/4) = 17R/3 = 5.66R

h) Repeat part (g) assuming sequential persistent connections
2R + 2(3R/4) + 2(R/2) + 2(R/3) + 2(R/4) = 17R/3 = 5.66R

i) Repeat part (g) assuming concurrent non-persistent connections
2R + 2(3R/4) = 3.5R

Q3. Wireless



Consider the above wireless network in which each node has a radio range of distance *d*. In the figure, two nodes are in each-other's range, if there is an edge between them. Consider two collision resolution schemes:

- CS: This is a pure carrier sense scheme in which a node does not send when it hears someone else transmitting, but otherwise can send whenever it wants.
- MACA: No carrier sense. Nodes wishing to communicate use an RTS-CTS-Data exchange. Nodes overhearing an RTS wait to allow the CTS to be sent. If no CTS is heard, the node can transmit. If a CTS is heard (even if no earlier RTS is heard), the node is quiet for the entire duration of the data transmission.

Note: Treat each of the five parts of this question as independent scenarios

a) Assume **node E is transmitting data to node B** when a node, say *X* decides to transmit data to another node, say *Y*. You have the following three cases:

Case 1: *X* = A, *Y* = B
Case 2: *X* = F, *Y* = C
Case 3: *X* = C, *Y* = A

For each case, answer the following questions:

i. If all nodes use CS, can *X* transmit to *Y*?
If NO: explain why.
If YES: Is the transmission successful (i.e., will *Y* correctly receive the data from *X*)? Is the original transmission (from E to B) impacted?

ii. If all nodes use MACA, can *X* transmit to *Y*?
If NO: explain why.
If YES: is the transmission successful (i.e., will node Y receive the data)? Is the original transmission (from E to B) impacted?

Case1: (i) Yes, Not successful, E to B impacted
(ii) No, as A will hear B's CTS and won't send an RTS.
Case2: (i) Yes, Successful, E to B not impacted
(ii) No, as C will hear B's CTS and won't send back a CTS.
Case3: (i) Yes, Successful, E to B impacted
(ii) No, as C will hear B's CTS and won't send an RTS.

b) Now assume **node B is transmitting data to node E** when a node, say *X* decides to transmit data to another node, say *Y*. You have the same three cases:

Case 1: *X* = A, *Y* = B
Case 2: *X* = F, *Y* = C
Case 3: *X* = C, *Y* = A

For each case, answer the following questions:

i. If all nodes use CS, can *X* transmit to *Y*?
If NO: explain why.
If YES: Is the transmission successful (i.e., will *Y* correctly receive the data from *X*)? Is the original transmission (from B to E) impacted?

ii. If all nodes use MACA, can *X* transmit to *Y*?
If NO: explain why.
If YES: is the transmission successful (i.e., will node B receive the data)? Is the original transmission (from B to E) impacted?

Case1: (i) No, A will hear B
(ii) No, A sends an RTS but B doesn't respond
Case2: (i) Yes, Not successful, B to E not impacted
(ii) No, C can't hear F's RTS
Case3: (i) No, as C will hear B
(ii) No, A can't hear C's RTS

c) Assume **A is transmitting to B** at the beginning of time. List the pair of nodes (if any) that can successfully communicate:
i. if all nodes use CS
ii. if all nodes use MACA
For this, consider each direction separately. i.e., list the pair *(X, Y)* if *X* can successfully

send data to *Y*, irrespective of whether *Y* can successfully send to *X*. We say that the data is sent successfully if the sender is able to transmit it and the receiver is able to successfully receive it, without any interference.

(i) No other pair can have successful communication with CS

(ii) No other pair can have successful communication with MACA

d) Node D would like to transmit to A, node E to B and node F to C. Can nodes D, E and F transmit simultaneously using CS? What about with MACA (clearly list your assumptions with respect to the ordering and timing of RTS/CTS)?

They can do so using CS.

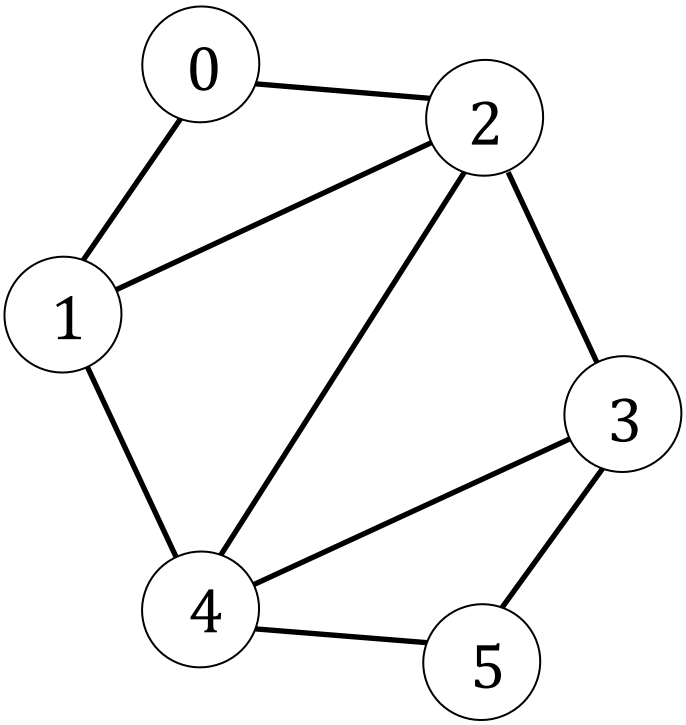
They can do so using MACA **only if** D, E and F send RTS at the same time and A, B and C then send CTS at the same time. If there is any time difference, it won't be possible.

e) Now assume *bidirectional communication* between node pairs (D, A), (E, B) and (F, C), i.e., for a node pair *(x,y)*, *x* has data to send *y* and *y* has data to send to *x*. In an ideal scenario, which pairs of nodes can simultaneously communicate? Will CS allow this ideal scenario? What about MACA (clearly list your assumptions with respect to the ordering and timing of RTS/CTS)?

In the ideal scenario, bidirectional/back-and-forth exchanges can happen in two "phases" that repeat:
In the first phase, the outside nodes can send i.e. (D->A), (E->B), (F->C) can communicate simultaneously.
In the second phase, the inner nodes can send i.e (A->D), (B->E), (C->F) can communicate simultaneously.

Using CS:
(D->A), (E->B), (F->C) can communicate simultaneously. But the bidirectional communication cannot happen.
Using MACA:
(D->A), (E->B), (F->C) can communicate simultaneously **only if** the three RTSes are sent at the same time and the three CTSes are sent at the same time.
(A->D), (B->E), (C->F) can communicate simultaneously **only if** the three RTSes are sent at the same time and the three CTSes are sent at the same time.

Q4. Spanning Tree Protocol and Learning Switches



Consider the above topology, with each node representing an Ethernet switch. For simplicity assume that the MAC address of a switch is represented by a single number as shown.

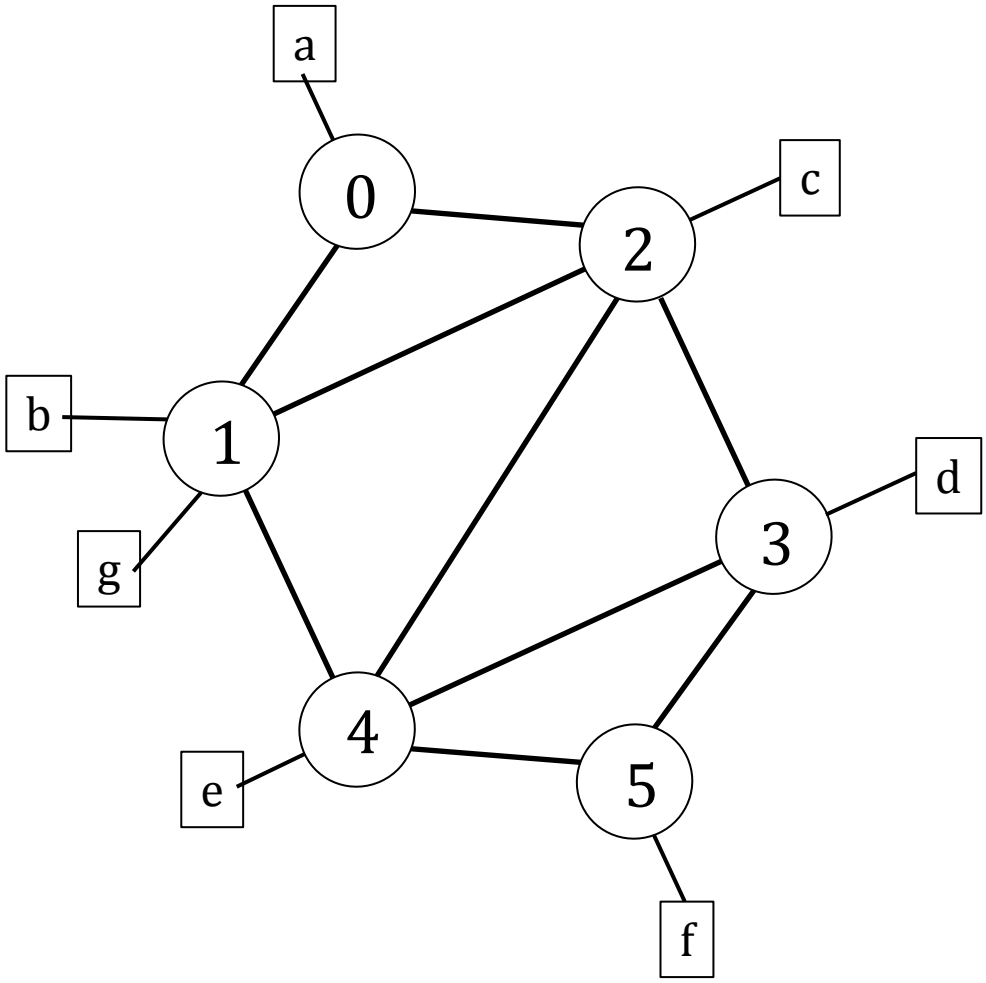
a) Spanning Tree Protocol

i. List the root and the edges of the spanning tree obtained after running the Spanning Tree Protocol for Switched Ethernet on the above network. (You don't have to show how the algorithm proceeds, simply write the final result.). Ties are broken based on smaller id if multiple shortest paths to the root exist.
Root: 0
Edges: 0-1, 0-2, 1-4, 2-3, 3-5

ii. Suppose node 0 fails. Identify the new root and the edges of the new Spanning Tree that is constructed after the failure.
Root: 1
Edges: 1-2, 1-4, 2-3, 4-5

b) Learning Switches

Consider the original topology (before node 0 fails) with end-hosts attached to them as shown below. Again, for simplicity, assume the MAC addresses of the end-hosts are represented by a single alphabet.
The circular nodes in the topology are "Learning Switches".



Consider the following packet transmissions occurring between a source-destination pair, in the given order, one after another. **For each packet transmission, list the switches and the end-hosts that receive the packet.**
Assume that you start with no entries in the forwarding table of any switch.
Note: If a switch floods the packet, all the adjacent switches and end-host will receive it and may further flood or unicast it. If a switch unicasts a packet, then only the corresponding end host or switch would receive the packet, which may further flood or unicast the packet.

1. b to c
2. c to b
3. d to c
4. a to b
5. a to g

1. (b), 1, g, 4, e, 0, a, 2, c, 3, d, 5, f
2. (c), 2, 0, 1, b
3. (d), 3, 5, f, 2, c
4. (a), 0, 1, b
5. (a), 0, 1, b, g, 4, e, 2, c, 3, d, 5, f

c) Learning Switches with Limited Memory

Consider the case that the switches have a small memory cache with enough space to store only a single entry in the forwarding table and therefore, they store only the most recent entry.
Assume that you start with no entries in the forwarding table of any switch.

For each of the packet transmissions, in the given order, write down whether **switch 0** floods the packet or unicasts it.

1. b to c flood
2. a to b unicast
3. c to b flood
4. b to c unicast
5. a to b unicast
6. c to b flood
7. b to c unicast
8. a to b unicast
9. c to b flood
10. b to c unicast
11. a to b unicast
12. c to b flood

i. List for each pair amongst (a to b), (b to c) and (c to b), the fraction of packets that are flooded by the switch and the fraction of packets that are unicast out of the 4 packets transmitted for each pair.
(a to b): 4/4 unicast; 0/4 flood
(b to c): ¾ unicast; ¼ flood
(c to b): 0/4 unicast; 4/4 flood

ii. How can you re-order these packet transmissions, such that for all three source-destination pairs, half of the packets are flooded and half are unicast? *(Hint: you need to make at most 2 swaps in the sequence of packet transmissions).*

1. b to c
2. a to b
3. c to b
4. b to c
5. ***c to b***

6. a to b
7. b to c
8. a to b
9. c to b
10. b to c
11. c to b
12. a to b

or

1. b to c
2. c to b
3. a to b
4. b to c
5. a to b
6. c to b
7. b to c
8. a to b
9. c to b
10. b to c
11. c to b
12. a to b

or

1. b to c
2. a to b
3. c to b
4. b to c
5. a to b
6. c to b
7. b to c
8. c to b
9. a to b
10. b to c
11. c to b
12. a to b