Problem Set #4

Due Tuesday, December 10th

Problem 1 Hashing (20 pts)

The hash table has 13 slots, and integer keys are hashed into the table with the following hash function H

```
int H (int key)
{
     x = ( key + 5 ) * ( key - 3 );
     x = int( x / 7 ) + key;
     x =   x % 13;
     return x;
}
```

(a) Fill in the final hash table with the following keys: 17, 22, 73, 56, 310, 100, 230, 12, 42, 18, 19, 24, 49.

Okay, so I wrote a little C program to do this.

```
jalu7098@elra-01:~/CSCI4273-NetworkSystems/JasonLubrano_PS4$ ./ps4_test
indx: 0 | arrs: 17 | hash: 9
indx: 1 | arrs: 22 | hash: 4
indx: 2 | arrs: 73 | hash: 8
indx: 3 | arrs: 56 | hash: 10
indx: 4 | arrs: 310 | hash: 7
indx: 5 | arrs: 100 | hash: 8
indx: 6 | arrs: 230 | hash: 11
indx: 7 | arrs: 12 | hash: 7
indx: 8 | arrs: 42 | hash: 4
indx: 9 | arrs: 18 | hash: 2
indx: 10 | arrs: 19 | hash: 8
indx: 11 | arrs: 24 | hash: 7
indx: 12 | arrs: 49 | hash: 0
```

Slot	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Contents	**49**		**18**		**22** **42**			**310** **12** **24**	**73** **100** **19**	**17**	**56**	**230**	

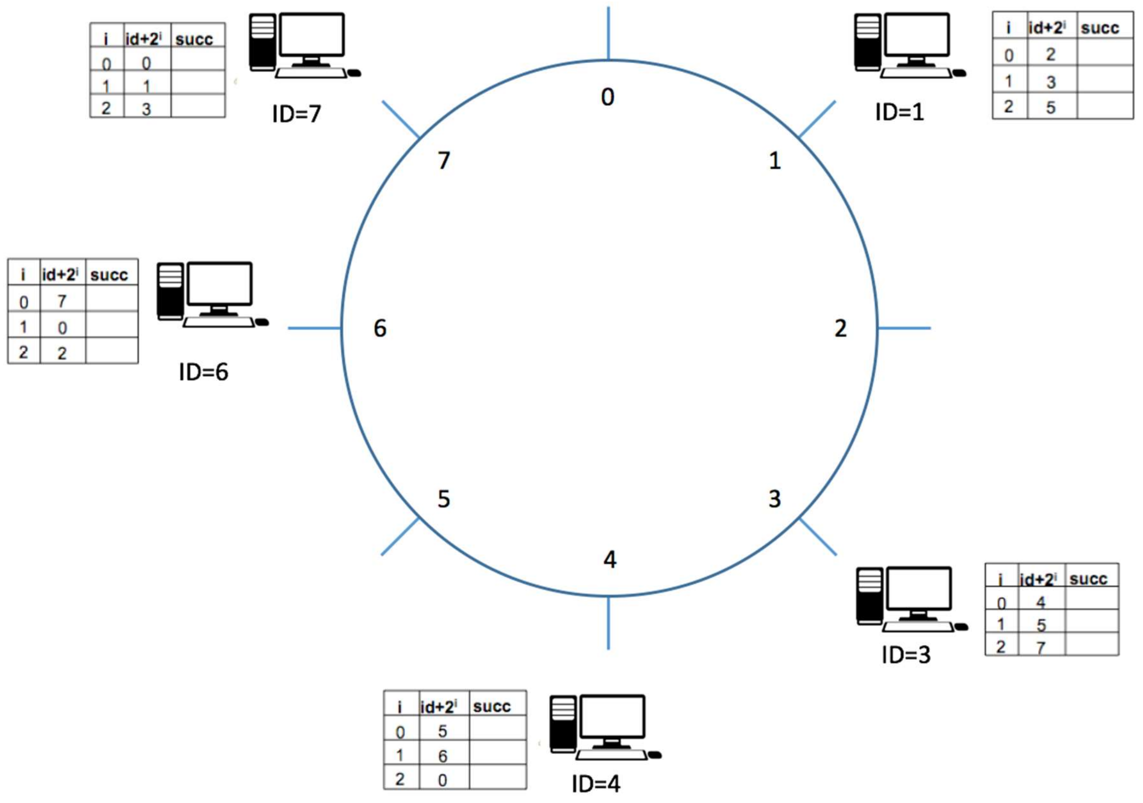
(b) List one or two methods that can handle collision in hashing.

Suppose H(int K) is our hash function that takes in some integer K. A hash collision occurs when H(K1) = H(K2). In our problem before we had several locations where H(K1) = H(K2) and = H(K3). One way to handle this is by **tabling**. What we do is initialize some length of table size, then overall K mod table_size and it creates a table where the values are linked list. This could work but in cases where a bucket in the table has a large hash linked-list, it may negatively affect performance to traverse this linked list. A second way to handle is **open addressing**. Here, the size of the table is greater than the number of keys. A set of functions probe the table to search for an empty slot. Lets say that we have H(K1) != H(K2) in our already stabled hash table. Suppose we insert H(K3) such that H(K3) = H(K2). What happens is that we run in our function: K3 mod table_size = K2 mod table_size. To resolve, we say K3 + 1 mod table_size. If K3 + 1 mod table_size = K1 mod table_size, then the algorithm moves on to K3 + 2 mod table_size. This finds an empty space for the new hash to arrive to. For large hash tables that are filled up, the worst case scenario is for the hash table to iterate through every value until the last bucket is empty.

Problem 2 Distributed Hash Tables (20 pts)

There is a Chord DHT in Figure 1. with 5 nodes. The finger tables are listed beside the nodes. Each node may be storing some items according to the Chord rules (Chord assigns keys to nodes in the same way as consistent hashing)

Figure 1. Chord DHT for Problem 2



(a) Fill in the table for node id=1 and 7

	ID＋2^i	successor
0	2	3
1	3	3
2	5	6

Table for ID=1

	ID＋2^i	successor
0	0	1
1	1	1
2	3	3

Table for ID=7

(b) List the node(s) that will receive a query from node 1 for item 5 (item named by key 5)

The query starts off on Node1. Node1 does not have node5 in its table, so it goes to Node3. It doesn't go to Node6 because that overshoots item 5.
Node1
The query, now at Node3, uses Node3's lookup table. Node3 has to go to Node4. Node3 HAS Node6 in its table, but again, this overshoots item 5.
Node3
Now at Node 4, the next item in the table is Node6.
Node6
Query traveling:
Node1 > Node 3> Node4> Node6

Problem 3 Bloom Filters (10 pts)

Derive the probability of false positive rate after 10 keys (or elements) are inserted into a table of size 100. Assume that 5 hash functions are used to setup bit positions in the table for the keys (elements).

5 Hashing functions means: For each key (element) there will be 5 bits to be set to 1 in the table (it can also be less than 5 if the hash functions generate same outputs).

Hint:
Assume **m** is the number of bits in the filter, **n** is the number of elements and k is the number of hash functions used.
After inserting one key, the probability of a particular bit being 0 is $(1-\frac{1}{m})^k$. This is because k hash functions are independent, and each hash function will have $(1-\frac{1}{m})$ probability for a particular bit remain 0. Then after inserting n keys, the probability for a particular bit remain 0 is $(1-\frac{1}{m})^{kn}$S
Now you have to apply this to the case of false positive.

P(bit=1 | H1) = (1 – 1/m)^(kn)
= (1 - 1/100)^(5*10)
= 0.605…
But that isn't for a false positive where the bit is set to one…
P(bit=1 | FP -> H1..K) = (1 - 0.605)^k
= (1 – 0.605)^5
= 0.00962
So the probability that a false positive occurs between all bits is 0.00962

Problem 4 P2P system (10 pts)

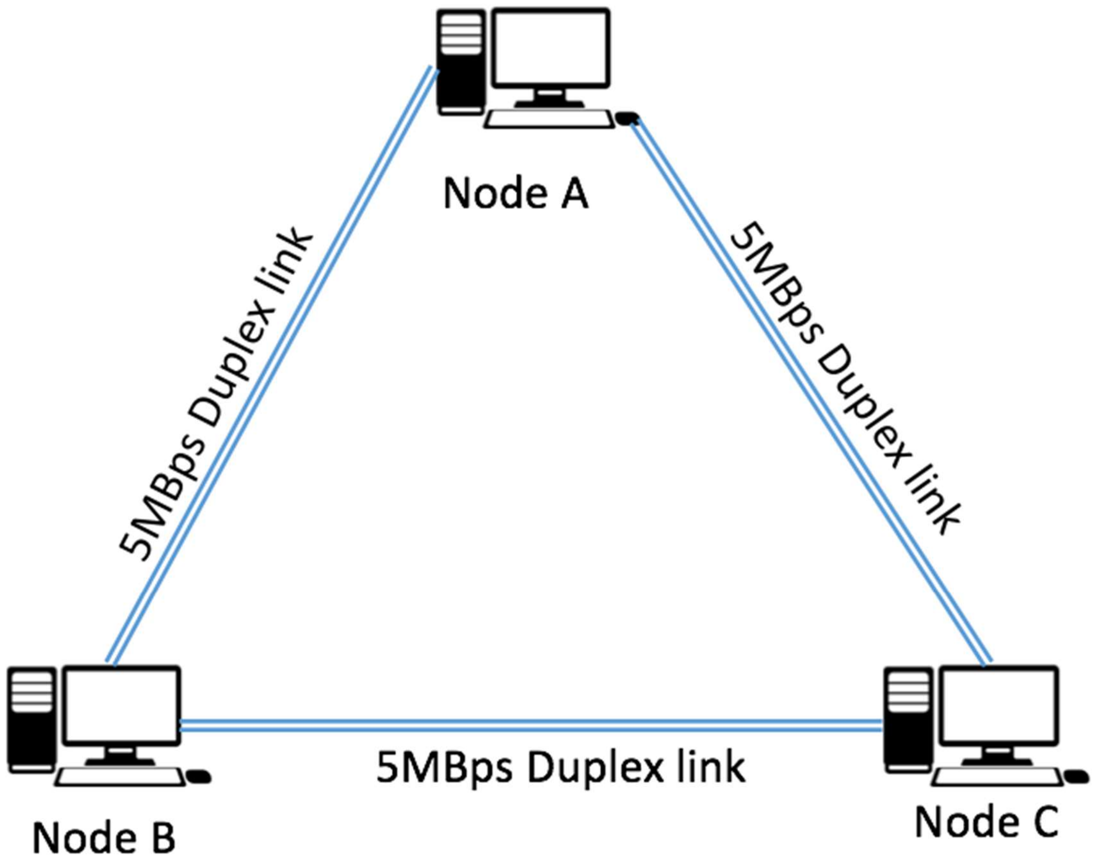


Figure 2. Network topology for Problem 4

3 nodes A, B and C are connected with each other via 5MBps duplex link as is now in Figure 2. Node A want to share a 2500MB file to Node B and C. During the actual transmission 2.5MB piece of the file can be send on the links each time. In the problem we ignore the RTT delay.

(a) What is the time of the sharing process using centralized approach? (The process ends when B and C all received the file, and the centralized approach means B and C are communicating with A independently and there is no communication between B and C)

Delay = R / S = 2500 MB / 5 MBps = 500 - seconds

(b) What is the ideal minimum time of the sharing process using P2P approach? (P2P approach means after B and C received a piece from A, they immediately share the their piece to other party)

Delay = 1/2 * R / S = 1/2 * 2500 MB / 5 MBps = 250 – seconds
A sends a packet to B and C
A -> B and A -> C
Then, B and C send a packet to each other
B -> C and C ->B

Problem 5 File Distribution (10 pts)

Consider distributing a file of F = 20 Gbits to N peers. The server has an upload rate of u_s = 30 Mbps, and each peer has a download rate of d_i = 2 Mbps and an upload rate of u. For N = 10 and 100 and u = 300 Kbps and 2 Mbps, prepare a chart giving the minimum distribution time for each of the combination of N and u for both client-server distribution and P2P distribution.

Client Server

U	N= 10	N= 100

300 Kbps	max{(10*20Gbit)/30Mbps, (20Gbit)/300Kbps}
	
	max {6667, 66667}
	
	66667
		max{(100*20Gbit)/30Mbps, (20Gbit)/300Kbps}
		
		max {66667, 66667}
		
		66667

2 Mbps	max{(10*20Gbit)/30Mbps, (20Gbit)/2Mbps}
	
	max {6667, 10000}
	
	10000
		max{(100*20Gbit)/30Mbps, (20Gbit)/2Mbps}
		
		max {66667, 10000}
		
		66667

Peer to Peer

U	N= 10	N= 100

300 Kbps	max{20Gbps/30Mbps, 20Gbps/300Kbps, (10*20Gbps)/(30Mbps + 10*300Kbps)}
	
	max{666.7, 66667, 6061}
	
	66667
		max{20Gbps/30Mbps, 20Gbps/300Kbps, (100*20Gbps)/(30Mbps + 100*300Kbps)}
		
		max{666.7, 66667, 33333}
		
		66667

2 Mbps	max{20Gbps/30Mbps, 20Gbps/2Mbps, (10*20Gbps)/(30Mbps + 10*2Mbps)}
	
	max{666.7, 10000, 4000}
	
	10000
		max{20Gbps/30Mbps, 20Gbps/2Mbps, (100*20Gbps)/(30Mbps + 100*2Mbps)}
		
		max{666.7, 10000, 8696}
		
		10000

Problem 6 BGP (20 pts)

1. Give the types of business relationships in BGP peering and mention who pays whom. What conditions make the Internet stable? Explain each condition in a line or two.

   Provider-Customer
   In a Provider-Customer relationship, the customer needs to be reachable from everyone, so the provider tells all its neighbors how to reach the customer. The customer does not want to provide transit service, so they do not let the providers route to it. The customer pays the provider for access.

   Peer-To-Peer
   In a peer-to-peer network, peers agree to exchange traffic for free. Peers exchange traffic between customers. AS exports only customer routes to a peer and a peer's routes only to its customers. As stated, normally no money is exchanged.

2. Please identify which of the following paths are valid, which of them are invalid based on the network topology of Figure 3.

Path 1 3 d	- Invalid
Path 1 4 d	- Valid
Path 8 d	- Valid
Path 6 d	- Valid
Path 4 d	- Valid
Path 7 5 d	- Valid
Path 7 5 3 d	- Valid
Path 2 1 3 d	- Invalid
Path 1 4 6 d	- Valid

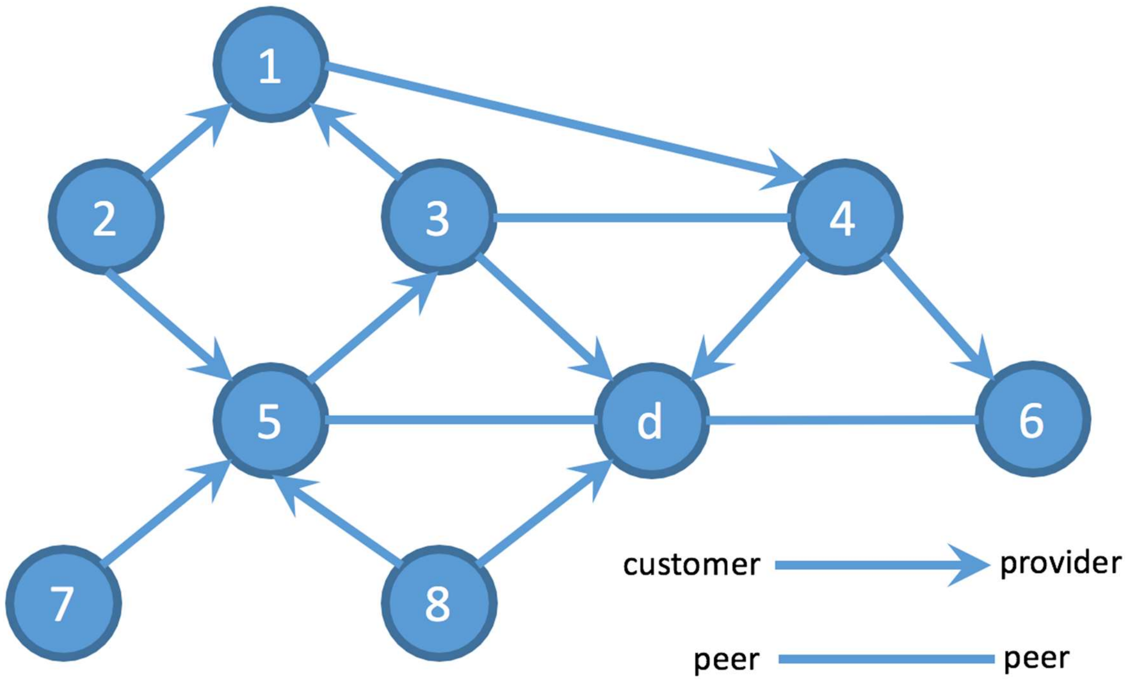


Figure 3. Network topology for Problem 6

Problem 7 Security (10 pts)

Diffie-Hellman Symmetric Key Exchange solves key the distribution issue of symmetric keys. Fill in the brackets below. Assume that Alice and Bob know p-ordered group G and a generator g, and Alice's and Bob's random seeds are ***a*** and ***b*** and their public keys are

A and ***B***, respectively, and computes a shared key **s**. Please use ***p*** = 23, ***g*** = 11, ***a***= 13, and ***b*** = 8. Note that Eve is an eavesdropper and can see any communication between Alice and Bob.

Alice		**Bob**		**Eve**	
Known	Unknown	Known	Unknown	Known	Unknown
p = 23 ***g*** = 11 ***a*** = 13	***b***	***p*** = 23 ***g*** = 11 ***b*** = 8	***a***	***p*** = 23 ***g*** = 11	***a, b***
1. Alice chooses a private key ***a*** and sends its public key ***A*** to Bob					
A = [g^a mod p] ***A*** = [11^13 mod 23] ***A*** = [17]					
2. Bob chooses a private key ***b*** and sends its public key ***B*** to Alice					
		B = [g^b mod p] ***B*** = [11^8 mod 23] ***B*** = [8]			
3. Eve knows Alice's public key ***A*** and Bob's public key ***B***					
A, B		***A, B***		***A, B***	
4. Alice and Bob calculate its shared key ***s***					
s = [B^a mod p] ***s*** = [8^13 mod 23] ***s*** = [18]		***s*** = [A^b mod p] ***s*** = [17^8 mod 23] ***s*** = [18]			***s***