ECE 358 Assignment 2

Jason Shao, Lihao Luo, Minghao Lu

May 21, 2016

1. Suppose we are given some integer $m > 1$, which parameterizes the m-bit ID's of our Chord DHT. Then, consider an instance of the DHT that is the following:

   - A peer at 0
   - A peer at 2^{m-1}
   - A peer at $2^m - 1$

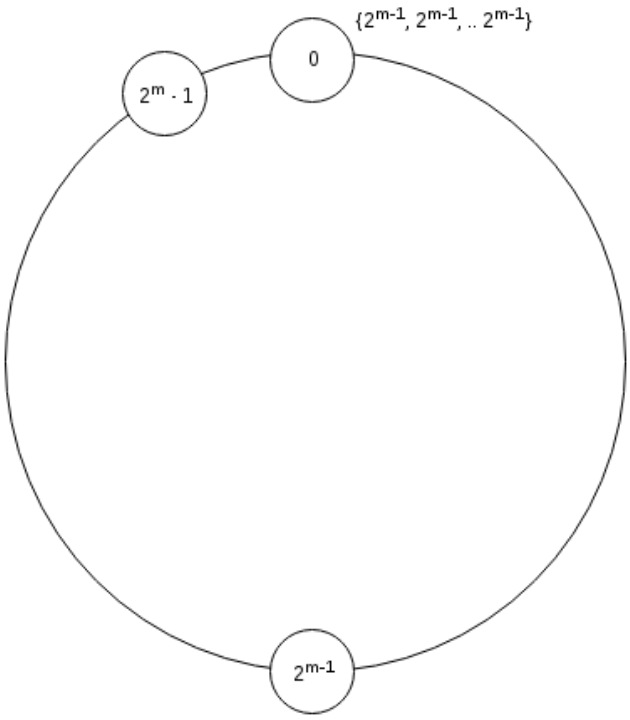

Figure 1: Configuration of Chord for Question 1

As we can see from Figure 1, the finger table for peer 0 contains all 2^{m-1}. This is because $FT_p[i] = succ(p + 2^{i-1}), i \in [1, m]$ so

$$FT_0[i] = succ(0 + 2^{i-1}), i \in [1, m]$$

This ranges from $succ(0+2^0)$ to $succ(0+2^{m-1})$ which both have values of peer 2^{m-1} since there are no peers between peer 0 and peer 2^{m-1}. Hence, the finger table values of peer 0 are all 2^{m-1}.

Now, consider $lookup(2^m-1)$ at peer 0. We know that the next hop will always be peer 2^{m-1}. In this situation, $p=0$, $q=2^{m-1}$, $k=2^m-1$. We can see that there are no peers between p and q. However, there is one peer between p and k. Therefore, we can draw the following contradiction from the original claim:

$$q-p \geq (k-p)/2$$
$$0 \geq \frac{1}{2}$$

By the principle of contradiction, it is shown for all $m>1$, there exists some configuration that disproves $q-p \geq (k-p)/2$.

2. (a) The worst case number of finger table need to be updated is 2^{m-1}. If $n-1 \leq 2^{m-1}$, the number of updates is all $n-1$ existing peers.

Proof there exists an arrangement of peers that costs 2^{m-1}

Let there be 2^{m-1} existing peers. Let the existing n-1 peers have keys 0 to n-2. Let the new peer have key $z=2^m-1$. We can observe this isn't the key of an existing entry since the max existing peer's key is $n-2$ and $n-2<n-1\leq 2^{m-1}\leq 2^{m-1}+(2^{m-1}-1)=2^m-1$. Let's consider an existing peer with the key value of p:

$$\begin{aligned} & p \leq n-2 && \text{since p is an existing peer} \\ \Rightarrow\ & p+2^{m-1} \leq n-2+2^{m-1} && \\ & \leq n-1+2^{m-1}-1 && \\ & \leq 2^{m-1}+2^{m-1}-1 && \text{since } n-1\leq 2^{m-1} \\ & \leq 2^m-1 && \leq z \end{aligned}$$

Since $FT_p[m]=succ(p+2^{m-1})$. If we can prove there is no existing peer with key q such that $p+2^{m-1}\leq q<2^m-1$, then z have to be the m^{th} entry of p's finger table. Since the z wasn't in p's table before, p's table must have updated.

Since p is non-negative, $p+2^{m-1}\geq 2^{m-1}\geq n-1>n-2$, since the max key value of existing key is $n-2$, $p+2^{m-1}$ is greater than any of the existing peer's key, thus there can't be an existing key q such that $p+2^{m-1}\leq q<2^m-1$. Thus we've prove p's table updated.

Since we put no restrictions on p other than its the key of an existing peer, we've proven all of the n-1 peers will have to have their finger table update. Thus this arrangement update 2^{m-1} nodes. This can be extended to any $n-1<2^{m-1}$ where all the existing $n-1$ nodes would be updated.

Proof the upper bound is is 2^{m-1}

First we should observe the absolute value of the inserted key does not matter, only its relative value matters. Thus, let the key of the new peer be $z = 2^m - 1$. From the proof above, we can observe if we place 2^{m-1} peers between 0 and $2^{m-1} - 1$, then the insert of z will cause 2^{m-1} updates. We prove from this particular arrangement onwards, "adding" peers between 2^{m-1} and $2^m - 2$ can not increase the number of updates beyond what we started with. We then prove from any arrangement, "removing" peers between 0 and $2^{m-1} - 1$ can not increase number of updates.

Proof that, given 2^{m-1} peers arranged between 0 and $2^{m-1} - 1$, "adding" peers with key between 2^{m-1} to $2^m - 2$ can not increase number of updates beyond what we started with. We consider all the peers we "added" at once. For the peers between 0 and $2^{m-1} - 1$, the new peer z only updates the m^{th} entry. Each of the peer we are "adding" is exactly 2^{m-1} off from one of the peers between 0 and $2^{m-1} - 2$. Thus, each peer we can will be the best fit m^{th} entry of some peer between 0 and $2^{m-1} - 2$, and the insertion of z can no longer update this finger table. "Adding" a peer obviously can not enable another peer's finger table to be updated by z's insertion if it wasn't updated before. Thus at best, each peer q "added" will enable itself to be update by z's insertion, but definitely will disable $p = q - 2^{m-1}$'s finger table from been updated. Thus, under this specific starting configuration "adding" new peers between 0 and $2^{m-1} - 1$ will not increase number of updates beyond what we started with.

Proof that "removing" peers with key between 0 and $2^{m-1} - 1$ can not increase number of updates. For any peer q between 2^{m-1} and $2^m - 2$, the new peer $z = 2^m - 1$ will have a "smaller" value than any peer p between 0 and $2^{m-1} - 1$(smaller as in how the q's finger table intepret $<$). Thus it can not be the case where without p, the z would have taken displaced some entry in the finger table of q, but with p, the z can't displace that same entry. This is because p can never compete with the z(where z would have originally displaced) since z is "smaller"(from q's perspective) than p and successor takes the smallest. Thus "removing" any entry between 0 and $2^{m-1} - 1$ does not enable z to update the table of any entry between 2^{m-1} and $2^m - 2$ that z didn't previously update.

For any peer p,r between 0 and $2^{m-1} - 1$ and $p \neq r$. If r is greater than p, than from r's perspective, the new peer z is "smaller" than p, so by the same logic as above, "removing" p could not have enabled update in r if r didn't updated before. If r is smaller than p, observe p does not qualify for the m^{th} entry of r's table since $r \geq 0 \Rightarrow r + 2^{m-1} \geq 2^{m-1} > p$. Thus "removing" p could not have stopped z from displacing the m^{th} entry of r. If z didn't cause r to update with p in the network, then "removing" p does not enable z to cause r to update.

We didn't assume any arrangement of existing peers before "removing" a peer p between 0 and $2^{m-1}-1$, so "removing" any number of such peer at any time can not increase the number of peers updated.

The combination of what we prove above imply an upper bound of 2^{m-1} for any arrangement. To see this, for any arrangement, start by arranging 2^{m-1} peers between 0 and $2^{m-1}-1$, this gives the familiar upper bound of 2^{m-1} updates. Then we "add" the necessary peers between 2^{m-1} to 2^m-2, no increase in number of updates. Then we can "remove" all the unnecessary peers between 0 and $2^{m-1}-1$, no increase in number of updates. So the final arrangement is still upper bounded by 2^{m-1} updates. Although when the circle is full, it is interesting to note that the number of updates is m, but nonetheless, this is not the worst case which is bounded by 2^{m-1}.

(b) Only the peer immediately preceding the new peer is updated, because that's the only peer that cares about the new peer. The other peers in the circle have an unchanged "next" peer. There can only be one immediate predecessor of the new peer, so at worst only one existing update is needed (not including the initialization of the inserted peer).

3. (a) For a given $m \geq 1$, we want to show that the worst case number of hops is bounded by a linear function $f(m)$. This is sufficient to show that the number of hops of *any* lookup is bounded by $f(m)$.

Consider $lookup(k)$ at peer p and r is the peer immediately before $succ(k)$. Suppose q is the hop after p. We showed in class that $p-q > (r-p)/2$ in terms of arclength. This tells us that the arclength distance from between hops are *at least* half of the distance to r.

Now let's consider how many times the query can cut the distance to r in half. Assume the worst case where the initial distance from p to r is maximal: for a m-bit Chord DHT, the arclength distance between two peers is bounded by 2^m as that's the total number of peers. This distance can be cut in half $\log_2 2^m = m$ times before the distance to r is equal to 1. To actually reach the $succ(k)$, there can be one hop needed to actually reach r and one additional hop needed to reach from r to $succ(k)$.

Overall, the worst case number of hops for is bounded by $f(m) = m+1+1 = m+2$ which is a linear function in m.

(b) Since I have to show a lower bound for the worst case number of hops, it is sufficient to show *any* configuration that takes $g(m)$ hops, because the worst case configuration must take at least $g(m)$ hops since it's the worst.

For a given $m \geq 1$, consider the following Chord DHT configuration: There exists a peer at every ID, so there are 2^m peers in total. Now consider $lookup(2^m - 1)$ at peer 0.

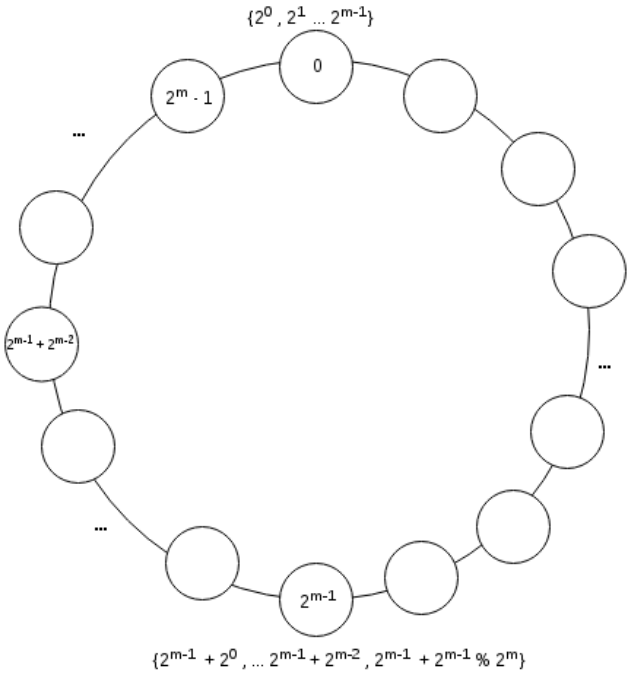

Figure 2: Configuration of Chord for Question 2b

Since we know every peer exists, each ID is responsible by exactly the peer associated with that ID.

First hop
Now consider $lookup(2^m - 1)$ at peer 0. Clearly, the next hop will be peer 2^{m-1} which is the last entry from the finger table. This has to be the case as we're looking up the furthest ID from peer 0.

Second hop
From peer 2^{m-1}, the values in its finger table are $\{2^{m-1} + 1, ..., 2^{m-1} + 2^{m-2}, 0\}$. We know the last entry is $2^{m-1} + 2^{m-1}$ which is peer 0 after modulus 2^m. We can't hop to the last entry because it overshoots the target, so the hop has to go to the peer $2^{m-1} + 2^{m-2}$.

Third hop
From peer $2^{m-1} + 2^{m-2}$, similarly, we can't overshoot the target by going to $2^{m-1} + 2^{m-2} + 2^{m-2}$, so we have to go to the peer $2^{m-1} + 2^{m-2} + 2^{m-3}$.

Similarly, we continue these hops until we reach peer $2^{m-1}+2^{m-2}+...+2^0 = 2^m-1$ which is our target lookup. The number of hops is equivalent to the number of power terms in $2^{m-1} + 2^{m-2} + ... + 2^0$ which is m. Therefore, we need m hops to perform $lookup(2^m - 1)$ from peer 0 under this configuration. Since I found a configuration and lookup that takes $g(m) = m$ hops, the worst case number of hops for a lookup must be at least $g(m) = m$ hops.

4. (a) $\underline{N_2}$

	Binary Form	Dot Decimal Notation
IP Address	00000001.00000010.00000011.00000100	1.2.3.4
Subnet Mask	11111111.11111111.11111111.11110000	255.255.255.240
After Mask	00000001.00000010.00000011.00000000	1.2.3.0

We see that address 1.2.3.4 does not lie in N_1 as the post-mask value is not 1.2.3.160.

	Binary Form	Dot Decimal Notation
IP Address	00000001.00000010.00000011.00000100	1.2.3.4
Subnet Mask	11111111.11111111.11111111.00000000	255.255.255.0
After Mask	00000001.00000010.00000011.00000000	1.2.3.0

We see that address 1.2.3.4 does lie in N_2 as the post-mask value is 1.2.3.0.

(b) $\underline{N_2}$

	Binary Form	Dot Decimal Notation
IP Address	00000001.00000010.00000011.11000011	1.2.3.195
Subnet Mask	11111111.11111111.11111111.11110000	255.255.255.240
After Mask	00000001.00000010.00000011.11000000	1.2.3.192

We see that address 1.2.3.195 does not lie in N_1 as the post-mask value is not 1.2.3.160.

	Binary Form	Dot Decimal Notation
IP Address	00000001.00000010.00000011.11000011	1.2.3.195
Subnet Mask	11111111.11111111.11111111.00000000	255.255.255.0
After Mask	00000001.00000010.00000011.00000000	1.2.3.0

We see that address 1.2.3.195 does lie in N_2 as the post-mask value is 1.2.3.0.

(c) Both

	Binary Form	Dot Decimal Notation
IP Address	00000001.00000010.00000011.10101011	1.2.3.171
Subnet Mask	11111111.11111111.11111111.11110000	255.255.255.240
After Mask	00000001.00000010.00000011.10100000	1.2.3.160

We see that address 1.2.3.171 does does lie in N_1 as the post-mask value is 1.2.3.160.

	Binary Form	Dot Decimal Notation
IP Address	00000001.00000010.00000011.10101011	1.2.3.171
Subnet Mask	11111111.11111111.11111111.00000000	255.255.255.0
After Mask	00000001.00000010.00000011.00000000	1.2.3.0

We see that address 1.2.3.171 does lie in N_2 as the post-mask value is 1.2.3.0.