

# ECE 358 Assignment

Jason Shao, Lihao Luo, Minghao Lu

June 5, 2016

1. I am answering this question under the assumption that each edge corresponds to a unique router.

The tight lower bound for the largest-sized routing table across all the  $n$  routers is 1. I will prove this by first showing that for any  $n > 0$ , I can construct such a network where the routing table does not need to be greater than 1.

Consider a network where the nodes (subnetworks) can form a Hamiltonian cycle (a cycle that includes all nodes in the graph). We can then take that cycle and create a routing table with exactly one entry that points to the next router (edge) in the cycle as the next hop for all destination addresses. Because we can create this cycle of all subnetworks, it is guaranteed that a packet forwarded in this cycle will be guaranteed to arrive at its intended subnetwork. For every other edge in the graph but not in the cycle, assign a single entry to any router that is directly in the cycle. This way, that router will forward the packet to a router in the Hamiltonian cycle which is then guaranteed to arrive at its destination. Here, we see that the lower bound is 1 entry if there exists a Hamiltonian cycle.

If it is the case that there is no Hamiltonian cycle, the minimum number of entries is 2. For any router (edge) in the configuration, partition the remaining subnetworks (vertices) into two sets: one set for the subnetworks on the left of the edge and one set for the subnetworks on the right. We will assign IP addresses using hierarchical addressing so that the addressing is sensitive to this topology. That means a router only needs to keep two entries in its table, one for each router in its incident subnetworks.

2. Yes, there exists a polynomial time algorithm for the decision problem. I will prove it by construction:
  - Look at every pair of prefixes
  - Check if any pair can be merged
    - If so, remove the pair of prefixes and insert the aggregated prefix. Then re-start at step 1 by checking all pairs of prefixes again
    - If not, check the final size of the list of prefixes, and compare it to  $k$ .

Checking if two prefixes can be merged can be done in  $O(1)$  time as the IP address is bounded. Merging the two prefixes can also be done in  $O(1)$  time by removing two existing 32-bit prefixes and inserting a new 32-bit prefix. There are  $O(n^2)$  pairs to check where  $n$  is the initial number of prefixes. After we do a merge, we reduce the size of the set of prefixes and check all pairs again. Therefore, we can do at most  $O(n)$  merges since it reduces the size of the set by 1 each time. In all, the algorithm has a worst case run-time of  $O(n)O(n^2) = O(n^3)$ . This is polynomial-time in the input size since there are  $n$  prefixes in the prefix list.

Assuming prefixes are stored as an array of bits, here is the pseudo-code:

```
def mergeable(prefix1, prefix2):
    if prefix1.prefix_length != prefix2.prefix_length:
        return False

    // Check that every bit is same except last bit
    for i <- 0 to prefix1.prefix_length - 2:
        if prefix1[i] != prefix2[i]:
            return False
    return prefix1[prefix_length - 1] != prefix2[prefix_length - 1]

def merge(prefix1, prefix2):
    // Decrement prefix length by 1
    output = prefix1
    output[prefix1.prefix_length - 1] = 0
    output.prefix_length = prefix1.prefix_length - 1

def decision(P, k):
    P' = P
    for i <- 0 to P'.size() - 1:
        for j <- i + 1 to P'.size() - 1:
            if mergeable(P'[i], P'[j]):
                P'.add(merge(P'[i], P'[j]))
                P'.remove(P'[i])
                P'.remove(P'[j])
                i <- 0
                j <- 0
    return P'.size() <= k
```

3. In the site depicted on slide 20 of lecture 1-4, all traffic destined for 12.46.129.5 may arrive at interface  $D$ . This is because  $S$  is a multi-homed site with both provider-aggregatable (PA) and provider-independent (PI) IPv4 addresses. If the site uses the

PA address, it advertises 12.46.129.0/25 at points  $C$  and  $D$  to  $P1$  and  $P2$  respectively. While  $P1$  can aggregate this prefix into 12/8,  $P2$  cannot since the prefix is not numerically adjacent to its own. For destinations matching 12.46.129.0/25, the prefix is more specific when going through  $P2$  than  $P1$ . Therefore, traffic with destination 12.46.129.5 tends to go through  $P2$  due to the longest-matching-prefix algorithm used in internet routing.

To address this, the site  $S$  can use the PI prefix instead of PA. Then, 198.134.135.0/24 is advertised to  $P1$  and  $P2$ . Since neither of the ISPs are able to aggregate this prefix, the full 198.134.135.0/24 is advertised to both points  $A$  and  $B$ . Therefore, hosts would send datagrams to 198.134.135.0/24 and  $S$  can be reached naturally through whichever ISP is closer to the sender.

4. Yes, I concur with Alice.

Let  $T$  be the initial minimum spanning tree of  $G$ . Let  $u$  be the node that leaves  $G$ , and let  $v$  be the node such that  $uv$  is the only edge incident to  $u$  in  $T$ . Let  $T'$  be  $T$  without  $u$  and  $uv$ . I want to show that  $T'$  is the minimum spanning tree of the new graph.

### Proof By Contradiction

Assume for the sake of contradiction that  $T'$  is not the most minimum spanning tree of the new graph. That means there must exist some tree  $R$  that spans the new graph, and the total weight of its edges is less than that of  $T'$ .

Now consider  $R$  with  $uv$  added to the tree. We can show that  $R$  with  $uv$  is a spanning tree of  $G$ , because  $R$  is a tree that spans every node in  $G$  except  $u$ . However, we know  $v$  is in  $R$  so  $uv$  can be added to  $R$  to span all nodes in  $G$  while maintaining the tree structure.

Moreover, observe that  $R$  with  $uv$  has total weight less than that of  $T$  because:

$$\begin{aligned} \text{weight}(R) &< \text{weight}(T') \\ \text{weight}(R) + \text{weight}(uv) &< \text{weight}(T') + \text{weight}(uv) \\ \text{weight}(R) + \text{weight}(uv) &< \text{weight}(T) \end{aligned}$$

Now we have shown that  $R$  with  $uv$  is a spanning tree of  $G$  and its weight is less than that of the  $T$ , the minimum spanning tree. This is clearly a contradiction, so we can declare that there does not exist another spanning tree that has total weight less than  $T'$ . Therefore, we do not need to recompute the MST as when a node leaves the network and it only has one edge incident on it, the new MST is merely the original tree with the node and its incident edge removed.

5. According to the book, for a request there are two "receiver" MAC address in the frame: DST address in the header that is 48 bits of 1 or Target address in the message

that is just 0. I'm going to assume you want the latter.

part a)

(i) request (ii) 1.2.3.4 (iii) 1.2.3.4's MAC address (iv) 1.2.3.10 (v) 0  
(i) response (ii) 1.2.3.10 (iii) 1.2.3.10's MAC address (iv) 1.2.3.4 (v) 1.2.3.4's MAC address

part b)

(i) request (ii) 1.2.3.4 (iii) 1.2.3.4's MAC address (iv) 1.2.1.1 (v) 0  
(i) response (ii) 1.2.1.1 (iii) 1.2.1.1's MAC address (iv) 1.2.3.4 (v) 1.2.3.4's MAC address  
(i) request (ii) 10.11.12.1 (iii) 10.11.12.1's MAC address (iv) 10.11.12.25 (v) 0  
(i) response (ii) 10.11.12.25 (iii) 10.11.12.25's MAC address (iv) 10.11.12.1 (v) 10.11.12.1's MAC address  
(i) request (ii) 15.16.17.25 (iii) 15.16.17.25's MAC address (iv) 15.16.17.18 (v) 0  
(i) response (ii) 15.16.17.18 (iii) 15.16.17.18's MAC address (iv) 15.16.17.25 (v) 15.16.17.25's MAC address