Homework

Zhijiia Chen

October 24, 2018

1 HW1

5. Hash table:

Can I use hash key as an id? Specify the reason.

No. A hash function is a multiple-to-one mapping which means for a certain item, its hash value is not unique, while an id by its definition should be unique.

How to find the difference between two list of strings?

For each pair of corresponding strings in the two list, first compare their hash values, if their hash values are different, then they are different. Otherwise, since having the same hash value does not guarantee the two strings are identical, we still need to compare the two strings character by character.

How to handle the duplication of the hash code?

We can do rehash by using new hash functions. We can also use open hash addressing such as chaining, linear probing.

How to deal with an shared hash table being used on the web server?

1. When updating the harsh table, lock the table. Each time, there are only one request can update the table. Save space because we only need to keep one copy of the table. But waste time, other requests need to wait.

2. Each request works on its own copy of the table, as long as they are not updating the same entry, the changes distributed among those copies can be merged. But if there multiple requests are updating the same entry, a lock is still needed. It saves time but wastes space.

2 HW2

1. What are the transport layer protocols used for live video, file transfer, DNS and email, respectively?

answer: UDP, TCP, UDP, TCP

2. Consider a TCP implementation with the Additive Increase (linear) and Multiplicative Decrease (AIMD) algorithm, ignoring the first phase where the ssthreshold is detected, assume the window size at the start of the slow start phase is 1 MSS and the ssthreshold at the start of the first transmission is 8 MSS.

Assume that a timeout occurs during the 4th transmission. Find the congestion window size at the end of the 8th transmission.

answer:

Suppose we are using the TCP reno implementation.

Phase 1: slow start. Double the window size after each successful transmission. Window size of 1st transmission: 1 MSS. Window size of 2nd transmission: 2MSS. Window size of 3rd transmission: 4 MSS. Window size of 4th transmission: 8 MSS. A timeout occurs, set ssthreshold to half of current window size, i.e., 4 MSS, and set window size to 1. Window size of 5th transmission: 1 MSS. Window size of 6th transmission: 2 MSS. Window size of 7th transmission: 4 MSS. The window size reaches the ssthreshold, leaving Phase 1 and entering phase 2.

Phase 2: congestion avoidance. Increase windwo size by one after each successful transmission. Window size of 8th transmission: 5 MSS.

So by the end of 8th transmission, the congestion windows size is 5 MSS.

3. UDP: Look into the following IP packet enclosing a UDP datagram. UDP checksum includes three sections: a pseudoheader, the UDP header, and payload – the data coming from the application layer. Pseudoheader and padding are only used to calculate checksum. After checksum is calculated, they are dropped. The figure. 1 is not a complete IP packet. It has the major info of the IP packet that has the UDP datagram and these info are what UDP checksum need to calculate. Now, assume we have

- Source IP: 153.18.8.105
- Destination IP: 171.2.14.10
- The protocol is ”UDP” (value is 17)
- Source port: 1087
- Destination port: 13
- Payload: ”TESTING”.

answer:

Source IP field: 1001100100010010 0000100001101001
Destination IP field: 1010101100000010 0000111000001010
Protocol field: 00010001
The length of the pseudoheader is 12 bytes.
The length of the head is 8 bytes.
The length of the payload after padding is 8 bytes.
UDP total length (28 bytes) field: 0000000000011100
Source port field: 0000010000111111
Destination port field: 0000000000001101
Data field: 0101010001000101 0101001101010100 0100100101001110 0100011100000000

Add up every 16 bits of the above fields (pad zeros if the length of a field is not the multiple of 16) with overflow bit wrapped around, we get 1001011011101001. Then we take the 1's complement of the sum to get the checksum 0110100100010110.

4. I run the nmap command "nmap -sP 192.168.56.1-128" in my linux virtual machine and got the following output:

Starting Nmap 7.60 (https://nmap.org) at 2018-10-24 12:01 EDT
Nmap scan report for 192.168.56.1
Host is up (0.00051s latency).
Nmap scan report for zhijia-VirtualBox (192.168.56.101)
Host is up (0.000086s latency).
Nmap scan report for 192.168.56.102
Host is up (0.00077s latency).
Nmap done: 128 IP addresses (3 hosts up) scanned in 17.16 seconds

I have two linux VMs running on my Mac and they are attached to a virtual subnet. So the command probes the available hosts in the network. In the above output, host 192.168.56.1 is my Mac, and host 192.168.56.101 and host 192.168.56.102 are the VMs.

3 HW3

1. Use Link-State and distance vector algorithms to find the short paths from the "source" node to other nodes in fig. 2. Describe each of your steps of the algorithms. Coding is not required.

Define the following notation:

- $d(v)$: cost of the least-cost path from the source node to destination v as of this iteration of the algorithm.
- $d_x(y)$ current minimum cost from node x to node y.
- $c(x, y)$ cost from x to directly attached neighbor y.
- $p(v)$: previous node (neighbor of v) along the current least-cost path from the source to v.
- N: all the nodes.
- N': subset of nodes; v is in N' if the least-cost path from the source to v is definitively known.

Link-State:

```
N' = {u}
for all nodes v
   if v is a neighbor of u then
```

```
        d(v) = c(u, v)
    else
        d(v) = ∞
loop
    find w not in N' such that d(w) is a minimum
    add w to N'
    update d(V) for each neighbor v of w and not in N':
        d(v) = min(d(v), d(w) + c(w, v))
until N' = N
```

Distance-Vector:
At each node x:

```
for all destinations y in N:
    d_x(y) = c(x, y) /* if y is not a neighbor, then c(x, y) =
        ↪ ∞ */
for each neighbor w
    d_w(y) = ∞ for all destinations y in N
for each neighbor w
    send distance vector D_x = [d_x(y): y in N] to w
loop
    wait (until sense a link cost change to some neighbor w or
        ↪  until receive a a distance vector from some
        ↪ neighbor w)
    for each y in N:
        d_x(y) = min_v{c(x, v) + d_v(y)}
    if d_x(y) changed for any destination y
        send distance vector D_x = [d_x(y): y in N] to all
            ↪ neighbors
forever
```

2. Assume if you are a user connecting to the node "source", and you want to browse a web page of a server connecting to one node in this network fig. 2. Describe each step with what protocols you use and what those protocols are doing for your browsing, like we did in class.

1. DNS lookup using UDP.
2. Initiate a TCP connection with the server.
3. Request the webpage using HTTP/HTTTPS protocol.

3. Assume you would like to set a VPN to connect to another node in this network fig. 2, how do you do it? Again describe each step.