

Network System Design

CS6100

Tutorial 05

MAC Address Lookup using Address Folding and Double Hashing

Student Details

Name: R Abinav
Roll No: ME23B1004
Date: February 13, 2026

1 Source Code

Listing 1: MAC address lookup implementation

```
1 use std::fmt;
2
3 //mac address structure (48 bits)
4 #[derive(Clone, Copy, Debug, PartialEq)]
5 struct MacAddress {
6     bytes: [u8; 6],
7 }
8
9 impl MacAddress {
10     fn new(bytes: [u8; 6]) -> Self {
11         MacAddress { bytes }
12     }
13
14     //fold 48-bit mac to 32-bit value
15     fn fold_to_32bit(&self) -> u32 {
16         //first 32 bits (bytes 0-3)
17         let upper = ((self.bytes[0] as u32) << 24)
18             | ((self.bytes[1] as u32) << 16)
19             | ((self.bytes[2] as u32) << 8)
20             | (self.bytes[3] as u32);
21
22         //last 16 bits (bytes 4-5)
23         let lower = ((self.bytes[4] as u32) << 8) | (self.bytes
24             [5] as u32);
25
26         //xor to get 32-bit folded address
27         upper ^ lower
28     }
29 }
```

```

27     }
28
29     fn from_string(s: &str) -> Result<Self, String> {
30         let parts: Vec<&str> = s.split(':').collect();
31         if parts.len() != 6 {
32             return Err("invalid mac address format".to_string());
33         }
34
35         let mut bytes = [0u8; 6];
36         for (i, part) in parts.iter().enumerate() {
37             bytes[i] = u8::from_str_radix(part, 16)
38                 .map_err(|_| "invalid hex in mac address".
39                         to_string())?;
40         }
41
42         Ok(MacAddress::new(bytes))
43     }
44
45 impl fmt::Display for MacAddress {
46     fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
47         write!(
48             f,
49             "{:02x}:{:02x}:{:02x}:{:02x}:{:02x}:{:02x}",
50             self.bytes[0],
51             self.bytes[1],
52             self.bytes[2],
53             self.bytes[3],
54             self.bytes[4],
55             self.bytes[5]
56         )
57     }
58 }
59
60 //hash table entry
61 #[derive(Clone, Debug)]
62 struct Entry {
63     mac: MacAddress,
64     port: u32,
65     occupied: bool,
66 }
67
68 impl Entry {
69     fn new() -> Self {
70         Entry {
71             mac: MacAddress::new([0; 6]),
72             port: 0,
73             occupied: false,
74         }
75     }
76 }

```

```

77 //hash table with open double hashing
78 struct HashTable {
79     table: Vec<Entry>,
80     size: usize,
81     count: usize,
82 }
83
84 impl HashTable {
85     fn new(size: usize) -> Self {
86         let mut table = Vec::with_capacity(size);
87         for _ in 0..size {
88             table.push(Entry::new());
89         }
90     }
91     HashTable {
92         table,
93         size,
94         count: 0,
95     }
96 }
97
98 //primary hash function h1
99 fn hash1(&self, folded: u32) -> usize {
100     (folded as usize) % self.size
101 }
102
103 //secondary hash function h2 (must be relatively prime to
104 //table size)
105 fn hash2(&self, folded: u32) -> usize {
106     let h2 = 1 + ((folded as usize) % (self.size - 1));
107     h2
108 }
109
110 //insert mac address with port mapping
111 fn insert(&mut self, mac: MacAddress, port: u32) -> Result<()
112     , String> {
113     if self.count >= self.size {
114         return Err("hash table is full".to_string());
115     }
116
117     let folded = mac.fold_to_32bit();
118     let h1 = self.hash1(folded);
119     let h2 = self.hash2(folded);
120
121     let mut index = h1;
122     let mut probes = 0;
123
124     //double hashing: h(k, i) = (h1(k) + i * h2(k)) mod size
125     while self.table[index].occupied && probes < self.size {
126         if self.table[index].mac == mac {
127             //update existing entry

```

```

126         self.table[index].port = port;
127         return Ok(());
128     }
129     probes += 1;
130     index = (h1 + probes * h2) % self.size;
131 }
132
133 if probes >= self.size {
134     return Err("could not find empty slot".to_string());
135 }
136
137 self.table[index].mac = mac;
138 self.table[index].port = port;
139 self.table[index].occupied = true;
140 self.count += 1;
141
142 Ok(())
143 }
144
145 //lookup mac address
146 fn lookup(&self, mac: MacAddress) -> Option<u32> {
147     let folded = mac.fold_to_32bit();
148     let h1 = self.hash1(folded);
149     let h2 = self.hash2(folded);
150
151     let mut index = h1;
152     let mut probes = 0;
153
154     while probes < self.size {
155         if !self.table[index].occupied {
156             return None;
157         }
158
159         if self.table[index].mac == mac {
160             return Some(self.table[index].port);
161         }
162
163         probes += 1;
164         index = (h1 + probes * h2) % self.size;
165     }
166
167     None
168 }
169
170 //delete mac address
171 fn delete(&mut self, mac: MacAddress) -> bool {
172     let folded = mac.fold_to_32bit();
173     let h1 = self.hash1(folded);
174     let h2 = self.hash2(folded);
175
176     let mut index = h1;

```

```

177     let mut probes = 0;
178
179     while probes < self.size {
180         if !self.table[index].occupied {
181             return false;
182         }
183
184         if self.table[index].mac == mac {
185             self.table[index].occupied = false;
186             self.count -= 1;
187             return true;
188         }
189
190         probes += 1;
191         index = (h1 + probes * h2) % self.size;
192     }
193
194     false
195 }
196
197 fn display_stats(&self) {
198     println!("hash table statistics:");
199     println!("    size: {}", self.size);
200     println!("    entries: {}", self.count);
201     println!("    load factor: {:.2}", self.count as f64 / self
202             .size as f64);
203 }
204
205 fn main() {
206     //create hash table with 1024 locations
207     let mut table = HashTable::new(1024);
208
209     println!("mac address lookup using address folding and open
210             double hashing");
211     println!("table size: 1024\n");
212
213     //test mac addresses
214     let test_macs = vec![
215         ("00:1a:2b:3c:4d:5e", 1),
216         ("ff:ee:dd:cc:bb:aa", 2),
217         ("12:34:56:78:9a:bc", 3),
218         ("aa:bb:cc:dd:ee:ff", 4),
219         ("00:00:00:00:00:01", 5),
220         ("ff:ff:ff:ff:ff:fe", 6),
221     ];
222
223     //insert mac addresses
224     println!("inserting mac addresses:");
225     for (mac_str, port) in &test_macs {
226         let mac = MacAddress::from_string(mac_str).unwrap();

```

```

226     let folded = mac.fold_to_32bit();
227     match table.insert(mac, *port) {
228         Ok(_) => println!("  {} -> port {} (folded: 0x{:08x})",
229                         " ", mac, port, folded),
230         Err(e) => println!(" failed to insert {}: {}", mac,
231                           e),
232     }
233
234     println!();
235
236     //lookup mac addresses
237     println!("looking up mac addresses:");
238     for (mac_str, _) in &test_macs {
239         let mac = MacAddress::from_string(mac_str).unwrap();
240         match table.lookup(mac) {
241             Some(port) => println!("  {} found on port {}", mac,
242                                     port),
243             None => println!("  {} not found", mac),
244         }
245     }
246
247     println!();
248
249     //test lookup of non-existent mac
250     let unknown_mac = MacAddress::from_string("de:ad:be:ef:ca:fe")
251         ".unwrap()";
252     match table.lookup(unknown_mac) {
253         Some(port) => println!("unknown mac {} found on port {}", unknown_mac,
254                               port),
255         None => println!("unknown mac {} not found (expected)", unknown_mac),
256     }
257
258     println!();
259
260     //delete a mac address
261     let delete_mac = MacAddress::from_string("00:1a:2b:3c:4d:5e")
262         .unwrap();
263     if table.delete(delete_mac) {
264         println!("deleted {}", delete_mac);
265     }
266
267     //verify deletion
268     match table.lookup(delete_mac) {
269         Some(port) => println!("{} still found on port {}", delete_mac,
270                               port),
271         None => println!("{} not found after deletion (expected)", delete_mac),
272     }

```

```
268     println!();
269     table.display_stats();
270 }
```

2 Output

```
?~?~/Desktop/course-work/nsd/NSD/tutorial-05 ?main ?1 .....  
› cargo run  
Compiling tutorial-05 v0.1.0 (/Users/abinav/Desktop/course-work/nsd/NSD/tutorial-05)  
  Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.77s  
    Running `target/debug/tutorial-05'  
mac address lookup using address folding and open double hashing  
table size: 1024  
  
inserting mac addresses:  
00:1a:2b:3c:4d:5e -> port 1 (folded: 0x001a6662)  
ff:ee:dd:cc:bb:aa -> port 2 (folded: 0xffee6666)  
12:34:56:78:9a:bc -> port 3 (folded: 0x1234ccc4)  
aa:bb:cc:dd:ee:ff -> port 4 (folded: 0xaabb2222)  
00:00:00:00:00:01 -> port 5 (folded: 0x00000001)  
ff:ff:ff:ff:ff:fe -> port 6 (folded: 0xfffff0001)  
  
looking up mac addresses:  
00:1a:2b:3c:4d:5e found on port 1  
ff:ee:dd:cc:bb:aa found on port 2  
12:34:56:78:9a:bc found on port 3  
aa:bb:cc:dd:ee:ff found on port 4  
00:00:00:00:00:01 found on port 5  
ff:ff:ff:ff:ff:fe found on port 6  
  
unknown mac de:ad:be:ef:ca:fe not found (expected)  
  
deleted 00:1a:2b:3c:4d:5e  
00:1a:2b:3c:4d:5e not found after deletion (expected)  
  
hash table statistics:  
size: 1024  
entries: 5  
load factor: 0.00
```

Figure 1: program output showing mac address operations