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

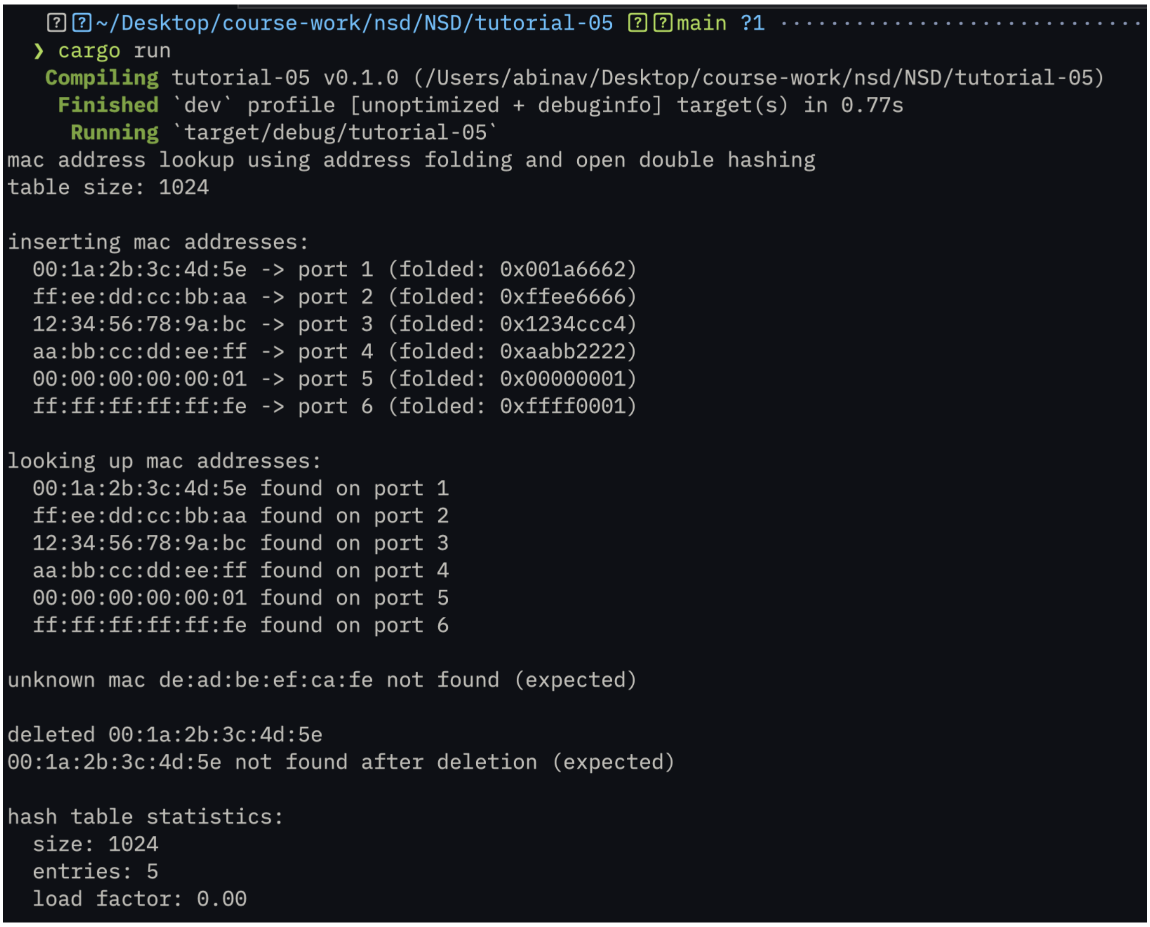
1 Source Code

Listing 1: MAC address lookup implementation

```
use std::fmt;

//mac address structure (48 bits)
#[derive(Clone, Copy, Debug, PartialEq)]
struct MacAddress {
    bytes: [u8; 6],
}

impl MacAddress {
    fn new(bytes: [u8; 6]) -> Self {
        MacAddress { bytes }
    }

    //fold 48-bit mac to 32-bit value
    fn fold_to_32bit(&self) -> u32 {
        //first 32 bits (bytes 0-3)
        let upper = ((self.bytes[0] as u32) << 24)
            | ((self.bytes[1] as u32) << 16)
            | ((self.bytes[2] as u32) << 8)
            | (self.bytes[3] as u32);

        //last 16 bits (bytes 4-5)
        let lower = ((self.bytes[4] as u32) << 8) | (self.bytes[5] as u32);

        //xor to get 32-bit folded address
        upper ^ lower
    }

    fn from_string(s: &str) -> Result<Self, String> {
        let parts: Vec<&str> = s.split(’:’).collect();
        if parts.len() != 6 {
            return Err("invalid mac address format".to_string());
        }

        let mut bytes = [0u8; 6];
        for (i, part) in parts.iter().enumerate() {
            bytes[i] = u8::from_str_radix(part, 16)
                .map_err(|_| "invalid hex in mac address".to_string())?;
        }

        Ok(MacAddress::new(bytes))
    }
}

impl fmt::Display for MacAddress {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        write!(
            f,
            "{:02x}:{:02x}:{:02x}:{:02x}:{:02x}:{:02x}",
            self.bytes[0],
            self.bytes[1],
            self.bytes[2],
            self.bytes[3],
            self.bytes[4],
            self.bytes[5]
        )
    }
}

//hash table entry
#[derive(Clone, Debug)]
struct Entry {
    mac: MacAddress,
    port: u32,
    occupied: bool,
}

impl Entry {
    fn new() -> Self {
        Entry {
            mac: MacAddress::new([0; 6]),
            port: 0,
            occupied: false,
        }
    }
}
```

```

//hash table with open double hashing
struct HashTable {
    table: Vec<Entry>,
    size: usize,
    count: usize,
}

impl HashTable {
    fn new(size: usize) -> Self {
        let mut table = Vec::with_capacity(size);
        for _ in 0..size {
            table.push(Entry::new());
        }
        HashTable {
            table,
            size,
            count: 0,
        }
    }

    //primary hash function h1
    fn hash1(&self, folded: u32) -> usize {
        (folded as usize) % self.size
    }

    //secondary hash function h2 (must be relatively prime to table size)
    fn hash2(&self, folded: u32) -> usize {
        let h2 = 1 + ((folded as usize) % (self.size - 1));
        h2
    }

    //insert mac address with port mapping
    fn insert(&mut self, mac: MacAddress, port: u32) -> Result<(), String> {
        if self.count >= self.size {
            return Err("hash table is full".to_string());
        }

        let folded = mac.fold_to_32bit();
        let h1 = self.hash1(folded);
        let h2 = self.hash2(folded);

        let mut index = h1;
        let mut probes = 0;

        //double hashing: h(k, i) = (h1(k) + i * h2(k)) mod size
        while self.table[index].occupied && probes < self.size {
            if self.table[index].mac == mac {
                //update existing entry
                self.table[index].port = port;
                return Ok(());
            }
            probes += 1;
            index = (h1 + probes * h2) % self.size;
        }

        if probes >= self.size {
            return Err("could not find empty slot".to_string());
        }

        self.table[index].mac = mac;
        self.table[index].port = port;
        self.table[index].occupied = true;
        self.count += 1;

        Ok(())
    }

    //lookup mac address
    fn lookup(&self, mac: MacAddress) -> Option<u32> {
        let folded = mac.fold_to_32bit();
        let h1 = self.hash1(folded);
        let h2 = self.hash2(folded);

        let mut index = h1;
        let mut probes = 0;

        while probes < self.size {
            if !self.table[index].occupied {
                return None;
            }

            if self.table[index].mac == mac {
                return Some(self.table[index].port);
            }

            probes += 1;
            index = (h1 + probes * h2) % self.size;
        }

        None
    }

    //delete mac address
    fn delete(&mut self, mac: MacAddress) -> bool {
        let folded = mac.fold_to_32bit();
        let h1 = self.hash1(folded);
        let h2 = self.hash2(folded);

        let mut index = h1;
        let mut probes = 0;

        while probes < self.size {
            if !self.table[index].occupied {
                return false;
            }

            if self.table[index].mac == mac {
                self.table[index].occupied = false;
                self.count -= 1;
                return true;
            }

            probes += 1;
            index = (h1 + probes * h2) % self.size;
        }

        false
    }

    fn display_stats(&self) {
        println!("hash table statistics:");
        println!("  size: {}", self.size);
        println!("  entries: {}", self.count);
        println!("  load factor: {:.2}", self.count as f64 / self.size as f64);
    }
}

fn main() {
    //create hash table with 1024 locations
    let mut table = HashTable::new(1024);

    println!("mac address lookup using address folding and open double hashing");
    println!("table size: 1024\n");

    //test mac addresses
    let test_macs = vec![
        ("00:1a:2b:3c:4d:5e", 1),
        ("ff:ee:dd:cc:bb:aa", 2),
        ("12:34:56:78:9a:bc", 3),
        ("aa:bb:cc:dd:ee:ff", 4),
        ("00:00:00:00:00:01", 5),
        ("ff:ff:ff:ff:ff:fe", 6),
    ];

    //insert mac addresses
    println!("inserting mac addresses:");
    for (mac_str, port) in &test_macs {
        let mac = MacAddress::from_string(mac_str).unwrap();
        let folded = mac.fold_to_32bit();
        match table.insert(mac, *port) {
            Ok(_) => println!("  {} -> port {} (folded: 0x{:08x})", mac, port, folded),
            Err(e) => println!("  failed to insert {}: {}", mac, e),
        }
    }

    println!();

    //lookup mac addresses
    println!("looking up mac addresses:");
    for (mac_str, _) in &test_macs {
        let mac = MacAddress::from_string(mac_str).unwrap();
        match table.lookup(mac) {
            Some(port) => println!("  {} found on port {}", mac, port),
            None => println!("  {} not found", mac),
        }
    }

    println!();

    //test lookup of non-existent mac
    let unknown_mac = MacAddress::from_string("de:ad:be:ef:ca:fe").unwrap();
    match table.lookup(unknown_mac) {
        Some(port) => println!("unknown mac {} found on port {}", unknown_mac, port),
        None => println!("unknown mac {} not found (expected)", unknown_mac),
    }

    println!();

    //delete a mac address
    let delete_mac = MacAddress::from_string("00:1a:2b:3c:4d:5e").unwrap();
    if table.delete(delete_mac) {
        println!("deleted {}", delete_mac);
    }

    //verify deletion
    match table.lookup(delete_mac) {
        Some(port) => println!("{} still found on port {}", delete_mac, port),
        None => println!("{} not found after deletion (expected)", delete_mac),
    }

    println!();
    table.display_stats();
}
```

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: 0xffff0001)

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