Politechnika Śląska

Wydział Automatyki, Elektroniki i Informatyki

Inżynieria Internetu

Temat 3

Prosta tablica routingu

Maciej Maciejewski Natalia Stręk

Gliwice 2025

1 Wstęp

Celem projektu było stworzenie programu konsolowego implementującego prostą tablicę routingu. Program ten przyjmuje jako pierwszy argument z linii poleceń plik tekstowy, zawierający linie w postaci: < IP podsieci> <maska sieci> <akcja>. Jako kolejne argumenty linii poleceń podawane są adresy IPv4. Program analizuje te adresy i dla każdego z nich sprawdza, do jakiej podsieci należy, porównując je z zapisanymi w pliku adresami podsieci za pomocą maski i operacji and. Jeśli dla podanego przez użytkownika adresu IPv4 zostanie odnaleziona pasująca podsieć, program zwraca odpowiadającą jej akcję. W przypadku braku dopasowania, program domyślnie przypisuje akcję „DROP". Program został wykonany w języku RUST.

2 Kod programu

```
use std::env;
use std::fs::File;
use std::io::{self, BufRead};

#[derive(Debug)]
struct Route {
    network: u32,
    mask: u32,
    action: String,
}

fn parse_ipv4(s: &str) -> u32 {
    let ip: Vec<&str>=s.split(".").collect();
    if ip.len() != 4
    {
        panic!("Invalid IPv4 address");
    }
    let mut ip_8: Vec<u32> = Vec::new();
    for i in ip.iter()
    {
        ip_8.push(i.parse::<u32>().expect("Parsing error"));
    }
    let ip_32: u32 = (ip_8[0] << 24) + (ip_8[1] << 16) + (ip_8[2] << 8) + (
   ip_8[3]);
    return ip_32;
}

fn ip_matches(ip: u32, route: &Route) -> bool {
    (ip & route.mask) == (route.network & route.mask)
}

fn read_routes(filename: &str) -> Vec<Route> {
    let file = File::open(filename).expect("Unable to open file");
    let reader = io::BufReader::new(file);

    reader.lines()
        .map(|line| {
            let line = line.expect("Unable to read line");
            let parts: Vec<&str> = line.split_whitespace().collect();
            if parts.len() != 3 {
                panic!("Invalid route line format");
            }
            Route {
```

```
                network: parse_ipv4(parts[0]),
                mask: parse_ipv4(parts[1]),
                action: parts[2].to_string(),
            }
        })
        .collect()
}

fn main() {
    let args: Vec<String> = env::args().collect();

    if args.len() < 3 {
        eprintln!("Usage: {} <routes_file> <ip1> <ip2> ...", args[0]);
        return;
    }

    let routes = read_routes(&args[1]);

    for ip_str in &args[2..] {
        let ip = parse_ipv4(ip_str);
        let mut action = "DROP".to_string();

        for route in &routes {
            if ip_matches(ip, route) {
                action = route.action.clone();
                break;
            }
        }

        println!("{} {}", ip_str, action);
    }
}
```

Opis działania programu

Poniżej przedstawiono kluczowe funkcje programu oraz ich role w działaniu aplikacji:

- **Route** – struktura reprezentująca pojedynczy wpis w tablicy routingu. Zawiera adres sieci (`network`), maskę (`mask`) oraz przypisaną akcję (`action`). Adresy IP i maski są przechowywane jako liczby 32-bitowe (`u32`).
- **parse_ipv4** – funkcja konwertująca adres IP w formacie tekstowym (np. `"192.168.0.1"`) na liczbę typu `u32` poprzez przesunięcia bitowe. Umożliwia późniejsze operacje logiczne na adresach IP.
- **ip_matches** – sprawdza, czy dany adres IP pasuje do danego wpisu w tablicy routingu. Porównanie odbywa się poprzez zastosowanie operacji AND z maską i porównanie z adresem sieci.
- **read_routes** – wczytuje zawartość pliku zawierającego trasę routingu. Każda linia pliku jest parsowana do struktury `Route`, a wynikowe wpisy są zwracane jako wektor.
- **main** – funkcja główna programu. Odczytuje argumenty z wiersza poleceń (plik + lista adresów IP), wczytuje tablicę routingu, a następnie dla każdego adresu IP wyszukuje pierwsze dopasowanie i wypisuje przypisaną akcję lub `DROP`, jeśli dopasowanie nie zostanie znalezione.

3 Przykładowe działanie programu

3.1 Plik tekstowy

192.168.0.0 255.255.255.0 akcja1
192.168.1.0 255.255.255.0 akcja2
0.0.0.0 255.255.0.0 akcja3
1.0.0.0 255.255.0.0 akcja4
192.168.4.1 255.255.255.252 akcja5
192.168.4.0 255.255.255.0 akcja6

3.2 Polecenie

cargo run routes.txt 192.168.1.1 10.0.0.1 192.168.4.2 192.168.4.118 1.1.0.0 1.0.0.1

3.3 Wynik

192.168.1.1 akcja2
10.0.0.1 DROP
192.168.4.2 akcja5
192.168.4.118 akcja6
1.1.0.0 DROP
1.0.0.1 akcja4

4 Podsumowanie

Program został zrealizowany zgodnie z podanymi założeniami. Podczas realizacji projektu zarówno zaznajomiono się, ale także pogłębiono znajomość języka RUST. Możliwym rozszerzeniem funkcjonalności realizowanego projektu byłoby wprowadzenie bardziej zaawansowanego mechanizmu wyboru najlepszego dopasowania np. poprzez najdłuższe dopasowanie prefiksu.