УНИВЕРЗИТЕТ У НОВОМ САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Nikola Aničić PR 24/2018

TRACE

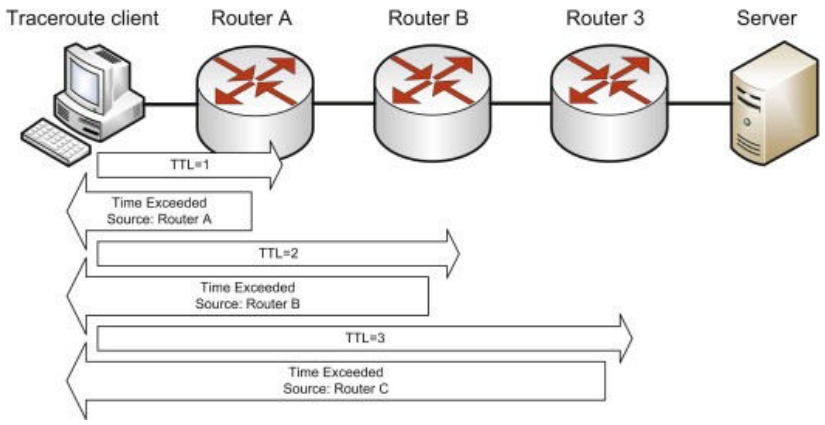
ПРОЈЕКАТ

- Примењено софтверско инжењерство (ОАС) -

SADRŽAJ

OPIS REŠAVANOG PROBLEMA

Potrebno je implementirati program koji skok po skok ispisuje rutu kojom mrežni paketi putuju do odredišne IP adrese. Svaki mrežni paket pod idealnim uslovima stiže do odredišta kroz niz rutera koji ga upućuju sve bliže odredištu. Svaki od rutera duž rute proverava TTL polje i na osnovu vrednosti odbacuje paket ili ga prosleđuje dalje na osnovu tabele rutiranja uz umanjivanje TTL vrednosti unutar zaglavlja za 1. Kao što je već navedeno cilj ovog projekta je implementirati program koji vraća svaki skok kroz koji paketi prolaze. Da bi program manipulisao TTL poljem potrebno je da ima pristup transportnom sloju mreže, odnosno da koristi IP protokol. Korišćenjem IP protokola program ima mogućnost da postavlja različite TTL vrednosti, pa paketi mogu duže ostati na mreži ali korišćenje IP protokola ne rešava problem dobijanja adrese svakog skoka. Ovaj problem se može rešiti korišćenjem još jednog protokola TCP/IP steka tojest ICMP protokola. ICMP je protokol koji je namenjen upravaljanju greškama na mreži kao i kontroli određenih mehanizama na mreži [1]. Slanjem ICMP paketa program ima mogućnost da dobija informacije o mreži, a da bi bila vraćena IP adresa rutera koji je umanjio TTL polje paketa potrebno je slati *echo request* poruke ka odredištu. Odredišni ruter će u slučaju *echo request* poruke ka klijentu vratiti paket koji je primio sa malom izmenom koja se ogleda u kopiranju dela IP paketa u deo za podatke ICMP paketa [1]. Program sada ima mogučnost dobavljanja adrese rutera kao kontrolu nad brojem skokova koje paket može da izvrši, pa se implementacija svodi na uzastopno slanje ICMP *echo request* paketa sa uvećavanjem TTL polja pri svakom zahtevu do dobijanja odgovora od rutera koji je zadužen za željenu IP adresu. Takođe je potrebno obraditi odgovor rutera i meriti proteklo vreme od trenutka slanja paketa do prijema odgovora. Srž rešenja problema može se jasno videti na slici 1.1.



slika 1.1

Isti problem rešava *tracert* komanda na *Windows* operativnom sistemu. *Tracert* takođe funkcioniše na principu slanja *echo request* paketa i uvećavanju TTL-a pri svakom sledećem slanju. Implementacija *tracert* komande takođe podržava određen skup flegova čijim korišćenjem je moguće modifikovati poziv komande npr. */d* fleg onemogućava da komanda vrši *reverse DNS lookup* za IP adrese skokova, *tracert* takođe podržava rad sa IPv6 adresama. Implementacija u ovom projektu koristi skup od samo dva flega, jedan obavezan za prosleđivanje IP adrese ili domenskog imena i jedan opcioni za prosleđivanje maksimalne TTL vrednosti.

Fokus problema je dakle pravilno korišćenje protkola TCP/IP steka i upotreba njihovih mehanizama kako bi se dobio željeni rezultat, a to su IP adrese svih skokova koje će paketi poslati sa klijenta izvršiti da bi stigli do odredišta.

OPIS KORIŠĆENIH TEHNOLOGIJA I ALATA

Kako je predmet u okviru kog se izvodi ovaj projekat predavan na programskom jeziku C na *Windows* operativnom sistemu i sam projekat je implementiran u programskom jeziku C uz korišćenje određenih biblioteka potrebnih za mrežnu komunikaciju. Razvojno okruženje korišćeno je *Visual Studio 2019* i *MSVC* kompajler koji dolazi u sklopu *Visual Studio 2019* razvojnog okruženja. Kako je implementacija namenjena za *Windows* platformu biblioteka korišćena za komunikaciju preko mreže je *winsock2. Winsock2* je zaglavlje u jeziku C koje u kombinaciji sa ws2_32.lib import bibliotekom uspostavlja interfejs ka mreznim funkcijama u ws2_32.dll dinamickoj biblioteci. Pored mrezne biblioteke takođe je korišćena i biblioteka za merenje vremena, odnosno *time.h* zaglavlje iz standardne biblioteke kao i *stdio.h* za rad sa standardnim ulazom i izlazom, *stdint.h* za korišćenje int tipova specifične širine.

OPIS REŠENJA PROBLEMA

Rešenje problema predstavljenog u uvodu se može razložiti na nekoliko bitnih delova. Pre svega ostalog potrebno je primiti unos od korisnika u vidu adrese ili domenskog imena i opciono maksimalnu ttl vrednost. Kada je unos uspešno prihvaćen, unete podatke je potrebno obraditi i validirati. Nakon validacije unetih podataka sledeći korak je dobavljanje IP adrese ako je korisnik uneo domensko ime. Ako je adresa uspešno razrešena program zauzima soket kroz koji će slati pakete ka odredištu i započinje ciklus slanja paketa. Ceo proces koji je iznad opisan u implementaciji je izdeljen na nekoliko logičkih celina odnosno zaglavlja u jeziku C. Svako od zaglavlja daje određeni deo funkcionalnosti i svako zaglavlje sadrži funkcije koje su vezane za tačno jednu funkcionalnost programa. Pregled zaglavlja, implementacija i funkcionalnost mogu se videti u tabeli 1.

ZAGLAVLJE	**IMPLEMENTACIJA**	**FUNKCIONALNOST**
address.h	address.c	Rad sa mrežnim adresama
bytes.h	bytes.c	Rad sa memorijom na heap-u
headers.h	/	Sadrži format IP i ICMP zaglavlja
icmp_codes.h	icmp_codes.c	Razrešavanje značenja ICMP poruke
stopwatch.h	stopwatch.c	Mernje proteklog vremena
user_input.h	user_input.c	Rad sa koriničkim unosom
winsock_wrapper.h	winsock_wrapper.c	Rad sa mrežom

tabela 1.

- *address.h*

Funkcije ovog zaglavlja omogućavaju rad sa mrežnim adresama, odnosno razrešavanje domenskih imena, konvertovanje između različite predstave IP adresa i popunjavanje struktura potrebnih za rad sa mrežom. Implementacija se oslanja na funkcije iz *winsock2* zaglavlja, kao i na *bytes* zaglavlje za rad sa memorijom. Implementacija funkcija definisanih u *address.h* zaglavlju nalazi se u *address.c* fajlu kao i dodatne funkcije vezane za implementaciju koje ne bi trebalo da budu dostupne van modula. Glavne funkcije modula su *populate_address(short port, char* address)* i *parse_from_hostname(char* host), populate_address()* funkcija popunjava *SOCKADDR_IN* strkturu i vraća pokazivač na ispravno popunjenu strukturu ili NULL ako parametri poslati u funkciju nisu ispravni ili nije uspešno zauzet prostor na heap-u za potrebnu strukturu, *parse_from_hostname()* koristi funkcije *winsock2* biblioteke *getaddrinfo()* [2] i *getnameinfo()* [2].

- *bytes.h*

bytes zaglavlje sadrži prototipove za samo dve funkcije, funkciju koja zauzima memoriju i vraća pokazivač na zauzetu memoriju i funkciju koja oslobađa memoriju. Implementacija korisiti *calloc()* [3]

funkciju koja zauzima memoriju i inicijalizuje svaki bajt zauzete memorije na 0 i *free()* [3] funkciju za oslobađanje memorije.

- *headers.h*

headers zaglavlje sadrži definicije struktura ICMP i IP zaglavlja potrebnih za slanje i prijem ICMP paketa.

- *icmp_codes.h*

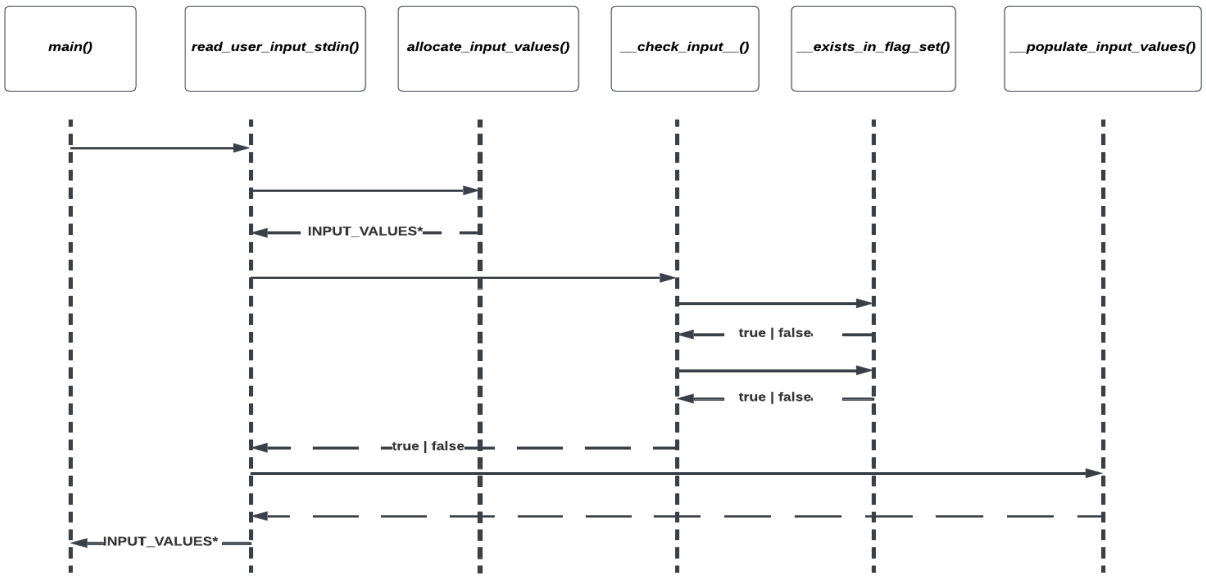
icmp_codes zaglavlje sadrži prototip za funkciju koja vraća konstantan pokazivač na char tip, odnosno razrešava ICMP poruku u odnosu na *type* i *code* polje u zaglavlju ICMP paketa. Implementacija zaglavlja sadrži *lookup* tabele koje sadrže tekst ICMP poruke izedeljene po *code* polju ICMP zaglavlja. Funkcija koja razrešava poruke na osnovu koda i tipa određuje iz koje tabele dobavlja poruku.

- *stopwatch.h*

stopwatch zaglavlje sadrži prototipove za funkcije koje predstavljaju bazičnu funkcionalnost štoperice odnosno *start(), stop()* i *get().* Implementacija štoperice sadrži globalne promenljive *start* i *stop* koje čuvaju vreme uključivanja odnosno isključivanja štoperice, *get()* funkcija vraća razliku između *stop* i *start* promenljive podeljenu sa brojem otkucaja sistemskog sata u sekundi i podešenu na milisekunde.

- *user_input.h*

user_input zaglavlje sadrži funkcije koje vrše rad sa korisnikom odnosno formatiraju i validiraju unete podatke, kao i definiciju *INPUT_VALUES* strukture u kojoj će biti čuvani uneti podaci. Implementacija modula koji radi sa korisnikom zasniva se na tačno određenom načinu pozivanja programa sa komandne linije sa pozicionim flegovima. Flegovi koji su dozvoljeni su *(a, l),* fleg *a* je obavezan dok je fleg *l* opcioni fleg. Flegom *a* se prosledjuje IP adresa ili domensko ime, a flegom *l* se prosleđuje maksimalni broj skokova u kom je potrebno kompletirati ispis rute. Funkcija koja vrši rad sa korisnikom i delegira delove funkcionalnosti ka implementacionim funkcijama je *read_user_input_stdin(),* ova funkcija vraća pozivaocu popunjenu *INPUT_VALUES* strukturu formatiranim podacima primljenim sa standardnog ulaza u slučaju da su prosleđeni podaci validni. Pravila za validaciju primljenih podataka sadržana su unutar *__check_input__()* funkcije i zasnivaju se na skupovima flegova, odnosno flegovi su pozicioni i na određenoj poziciji može se naći samo fleg iz skupa koji sadrži flegove za poziciju. Proveru pripadnosti skupu flegova vrši *__exists_in_flag_set()* funkcija. Dakle *main()* funkcija od operativnog sistema preuzima parametre koje korisnik prosleđuje pri pozivu programa, *main()* poziva *read_user_input_stdin()*, a ova funkcija kroz pomoćne funkcije zauzima prostor, proverava unos, popunjava *INPUT_VALUES* strukturu i vraća je korisniku. Dijagram opisane sekvence može se videti na slici 3.1.



slika 3.1.

- *winsock_wrapper.h*

winsock_wrapper zaglavlje proglašava funkcije koje u implementaciji pružaju pojednostavljen interfejs ka mrežnim funkcijama *winsock2* biblioteke i dodaje *ping()* funkciju. Mrežne funkcije koje su obuhvaćene *winsock_wrapper* zaglavljem vezane su za dobavljanje i podešavanje soketa, pokretanje i zaustavljanje *winsock2* biblioteke kao i funkcije za slanje podataka ka mreži i prijem podataka sa mreže. Implementacija sadrži tri globalne promenljive, broj sekvence paketa, vrednost tajmauta i skup fajl deskriptora. Inicijalizacija *winsock2* biblioteke izvršena je *winsock_startup()* funkcijom koja pored pokretanja *winsock2* biblioteke inicijalizuje tajmaut i skup fajl deskriptora. Zaglavlje takođe sadrži i funkciju *get_raw_icmp_socket()* koja dobavlja soket podešen za korišćenje uz ICMP protokol , funkcije koje podešavaju opcije na soketu su *set_non_blocking_mode()* ,koja je iskorišćena da označi soket kao neblokirajući, i *set_socket_ttl()* koja podešava time-to-live vrednost na soketu. Ključne funkcije u ovoj celini su funkcije koje se bave slanjem i prijemom podataka, odnosno *pack_ping_packet(), send_ping_packet()* i *receive_ping_reply(). pack_ping_packet()* u prosleđen *IcmpHeader** bafer unosi potrebne vrednosti za ICMP *echo request* poruku, postavlja vremensku odrednicu, broj sekvence, inicijalizuje deo za podatke u paketu na nula i računa kontrolnu sumu ICMP zaglavlja (slika 3.2.). Kontrolna suma se računa po algoritmu opisanom u RFC 792 [4].

```
void pack_ping_packet(IcmpHeader* icmp_buffer, int packet_size)
{
    if(!icmp_buffer) return;

    icmp_buffer->baseData.type = 8;
    icmp_buffer->baseData.code = 0;

    icmp_buffer->echo.id = (uint16_t)getpid();
    icmp_buffer->echo.sequence = sequence_number++;
    icmp_buffer->timestamp = GetTickCount();
    icmp_buffer->baseData.checksum = 0;
    char* packet_data_pointer = (char*)icmp_buffer + sizeof(IcmpHeader);
    int bytes_left = packet_size - sizeof(IcmpHeader);
    memset(packet_data_pointer, 0, bytes_left);
    icmp_buffer->baseData.checksum = icmp_checksum((uint16_t*)icmp_buffer, packet_size);
}
```

slika 3.2.

Funkcije *send_ping_packet()* i *receive_ping_reply()* vrše slanje paketa na određenu adresu odnosno prijem paketa sa određenog soketa korišćenjem *sendto()* i *recvfrom()* [2] funkcija iz *winsock2* biblioteke. *send_ping_packet()* poziva *sendto()* nad prosleđenim parametrima i na osnovu broja poslatih bajtova vraća povratnu vrednost o uspešnosti slanja paketa, *receive_ping_reply()* zauzima prostor na lokalnom steku za bafer od 256 B, taj bafer prosleđuje *recvfrom()* funkciji i ako je prijem podataka uspešan iz bafera preuzima IP i ICMP zaglavlja. Sve prethodno opisane funkcije služe kao pomoćne da bi se ostvarila glavna funkcionalnost ove celine a to je dobavljanje informacija vezanih za jedan skok i ispis na standardni izlaz odnosno *trace_step()* funkcija. Ova funkcija upravlja izvršavanjem prethodno opisanih pomoćnih funkcija i kroz njih omogućava funkcionalnost ove celine odnosno ona priprema paket za slanje, šalje određen broj istih paketa ka odredištu, meri proteklo vreme do odgovora za svaki od paketa i vrši ispis proteklog vremena i IP adrese odnosno domenskog imena ako ga je moguće dobaviti. Slika 3.3. prikazuje prethodno opisanu sekvencu uz izostavljanje pokretanja i zaustavljanja štoperice i pozivanja funkcija koje vrše ispis. *trace_step()* dobavlja adresu jednog skoka i ispisuje informacije na standardni izlaz. Sama sekvenca biće ponavljana do dosezanja maksimalne ttl vrednosti ili dobijanja odgovora sa unete odredišne adrese.

slika 3.3.

Program dakle po pokretanju iz konzolnog prozora preuzima parametre prosleđene u pozivu komande, validira adresu i time to live vrednost ako je prosleđena. Nakon validacije prosleđenih parametara ako je prosleđeno domensko ime ono biva razrešeno na IP adresu, zatim se od operativnog sistema dobavlja soket i počinje sa slanjem paketa ka odredištu. Slanje paketa traje dok se ne dosegne maksimalna ttl vrednost ili dok se ne dobije odgovor od željene adrese, izvedba u kodu je prikazana na listingu 3.1.

```c
INPUT_VALUES* user_input = read_user_input_stdin(argv, argc);
printf("\nTRACING ROUTE TO %s [MAX %d HOPS]\n", user_input->address,user_input->ttl);

user_input->address = parse_from_hostname(user_input->address);
SOCKADDR_IN* destination = populate_address(PING_PORT, user_input->address);

SOCKET trace_socket = get_raw_icmp_socket();
if (!set_non_blocking_mode(trace_socket)) exit(EXIT_FAILURE);

for (int ttl = 1; ttl <= user_input->ttl && !is_same_addr(&responder, &(destination->sin_addr)); ttl++)
{
    set_socket_ttl(trace_socket, &ttl);
    trace_step(trace_socket, destination, &responder);
}
```

listing 3.1.

PREDLOZI ZA DALJA USAVRŠAVANJA

Najveci nedostatak sistema je neadekvatan sistem prihvatanja flegova, odnosno pozicioni sistem koji u trenutnom obliku ima lošu održivost jer za svaku novu opciju potrebno je dodati novi skup flegova ili napraviti isključiv uslov sa flegovima jednog od postojećih skupova. Optimalniji način rada sa flegovima bio bi putem *lookup* tabela, tj. u nizu na normalizovanom indeksu vrednosti flega čuvati strukturu koja bi sadržala sve što je potrebno da se obradi unos određenog flega kao u primeru na listingu 4.1.

```c
#define chartoindex(index)(index - 32 - '0')

typedef struct __flag_table_values__
{
    void (*flag_handler)(char*);
}FlagTableNode;


void handle_a_flag(char* passed_value)
{
    // neki kod koji obrađuje ulazne vrednosti flega a
}

FlagTableNode __flags_table__[30] = { [chartoindex('a')] = {handle_a_flag}};
```

listing 4.1.

Ovakvom obradom flegova dobio bi se održiviji i pregledniji kod kao i to da bi se izgubila poziciona priroda flegova što bi olakšalo korišćenje programa. Pored promene načina obrade flegova, trebalo bi povećati broj opcija koje korisnik može proslediti programu od kojih bi najvažnija bila opcija koja bi program sprečavala da vrši *reverse dns lookup* što u većini slučajeva znatno usporava rad programa. Takođe bi trebalo dodati opcije za prosleđivanje željenog tajmauta i prosleđivanje broja paketa koji će biti poslat na odredište. Još jedna ispravka rešenja mogla bi doći u vidu centralizovanja zauzimanja i oslobađanja memorijskih bafera, odnosno prosleđivanje delova zauzete memorije iz *main()* funkcije ili dodatnog modula koji bi upravljao memorijom programa nasuprot lokalnom pozivanju funkcija za zauzimanje i oslobađanje memorije. Centralizovanje rada sa memorijom dovelo bi do manjih šansi za curenje memorije kao i do toga da bi kod bio održiviji.

LITERATURA

[1] Internet Core Protocols The Definitive Guide - Eric Hall

[2] *ws2tcpip.h - https://docs.microsoft.com/en-us/windows/win32/api/ws2tcpip/*

[3] *Microsoft C runtime library reference* - https://docs.microsoft.com/en-us/cpp/c-runtime-library/c-run-time-library-reference?view=msvc-170

[4] RFC 792 - https://www.rfc-editor.org/rfc/rfc792