Protocoale de Comunicatie № 1

Universitatea Politehnica din Bucuresti 28/03/2021

1 Prezentare generala

In cadrul temei vom implementa procesul de dirijare a pachetelor dintr-un router (forwading in engleza). Un router are doua parti, astfel:

- Data-plane - care implementeaza procesul de dirijare. Tema voastra este implementarea acestei componente. In cele ce urmeaza, in lipsa altor precizari, toate referintele la router din textul temei se refera la partea de dirijare a pachetelor.
- Control plane - componenta care implementeaza algoritmii de rutare e.g. RIP, OSPF, BGP; acesti algoritmi distribuiti calculeaza routele pentru fiecare retea destinatie si le insereaza in dataplane. NU este nevoie sa implementati acesti algoritmi pentru aceasta tema. Routerul vostru va functiona cu o tabela de rutare statica, data intr-un fisier de intrare, si care nu se schimba in timpul rularii routerului.

Un router are mai multe interfete si poate receptiona datagrame pe oricare dintre acestea. Routerul trebuie sa transmita pachetul mai departe, catre un calculator sau alt router direct conectat, in functie de regulile din tabela de rutare. Se cere implementarea procesului de dirijare a routerului.

2 Procesul de dirijare

Procesul de dirijare constă în primirea unui pachet, investigarea tabelei de rutare, descoperirea rutei corespunzătoare și dirijarea pachetului. Dirijarea este un proces care are loc la nivelul 3 (Rețea) din stiva OSI, lucrând cu adresa IP destinatie din pachetele dirijate.

Rutele (in numar arbitrar de mare) se găsesc în tabela de rutare și constau din două elemente:

- partea de match: adresa de rețea destinație (adresă și mască de rețea).
- partea de acțiune: următorul dispozitiv de rutare (next hop) (sau interfața de ieșire)

Tabela de routare din cadrul temei va fi structurata ca in exemplul de mai jos.

Listing 1: Tabela de rutare

```
Prefix        Next hop     Mask           Interface
192.168.0.0 192.168.0.2 255.255.255.0 0
192.168.1.0 192.168.1.2 255.255.255.0 1
```

```
192.168.2.0 192.168.2.2 255.255.255.0 2
192.168.3.0 192.168.3.2 255.255.255.0 3
```

Nota: tabela de mai sus este data cu scop informativ. Tabelele folosite pentru testarea temelor pot contine oricate intrari, oricate interfete, si adrese next hop si masti arbitrare.

În momentul în care un dispozitiv care rutează primește un pachet, extrage adresa IP destinație, localizează cea mai specifica intrare din tabela de rutare care se potriveste cu pachetul si dirijează (retransmite) pachetul către următorul ruter (next hop).

Functionalitatile routerului, executate intr-o bucla infinita, sunt:

1. Primeste un pachet de la oricare din interfetele adiacente.
2. Daca este un pachet IP destinat routerului, raspunde doar in cazul in care acesta este un pachet ICMP ECHO request. Arunca pachetul original.
3. Dace este un pachet ARP Request catre un IP al routerului, raspunde cu ARP Reply cu adresa MAC potrivita.
4. Daca este un pachet ARP Reply, updateaza tabela ARP; daca exista pachete ce trebuie dirijate catre acel router, transmite-le acum.
5. Daca este un pachet cu TTL <= 1 trimite un mesaj ICMP corect sursei (vezi mai jos); arunca pachetul.
6. Daca este un pachet cu checksum gresit, arunca pachetul.
7. Decrementeaza TTL, updateaza checksum.
8. Cauta intrarea cea mai specifica din tabela de rutare (numita f) astfel incat ($iph- > daddr\&f.mask == f.prefix$). Odata identificata, aceasta specifica next hop pentru pachet. In cazul in care nu se gaseste o ruta, se trimite un mesaj ICMP sursei; arunca pachetul
9. Modifica adresele source si destination MAC. Daca adresa MAC nu este cunoscuta local, genereaza un ARP request si transmite pe interfata destinatie. Salveaza pachetul in coada pentru transmitere. atunci cand adresa MAC este cunoscuta (pasul 4).
10. Trimite pachetul mai departe folosind functia send_packet(...).

Procesul se reia pe următorul router până când pachetul ajunge la destinație. In cadrul temei, vor exista două routere care sunt conectate direct la câte două calculatoare gazdă emulate (host-uri).

Implementarea routerului se va face in router.c. Datagramele primite/trimise de router vor avea urmatorul format:

Listing 2: Functionalitati router.

```
typedef struct {
```

```
        int len;
        char payload[MAX_LEN];
        int interface;
} packet;
```

In scheletul routerului se gasesc doua functii get_packet care primeste o datagrama si send_packet care trimite o datagrama pe o interfata specificata.

Listing 3: Functionalitati router.

```
/*
  send_packet nu va elibera memoria pachetului. intoarce zero in caz
  de eroare.
*/
int send_packet(int interface, packet *m);

/*
   receive_packet scrie un pachet la adresa primita
   intoarce 0 in caz de eroare.
*/
int get_packet(packet *m);

/* get the IP of one of the router's interfaces */
char *get_interface_ip(int interface);

/* get the MAC of one of the router's interface */
int get_interface_mac(int interface, uint8_t *mac);
```

3 Protocoale utilizate

In schelet apar urmatoarele structuri pentru headere protocoalelor utilizate. Puteti sa le folositi sau sa va definiti propria voastra structura.

3.1 Ethernet

In cadrul temei o sa lucrati cu frame-uri de Ethernet. Atunci cand un packet este trimis catre next hop schimbam adresa sursa si adresa destinatie a pachetului.

În headerul de sistem `netinet/if_ether.h`, puteți găsi o descriere C a antetului de Ethernet sub forma structurii `struct ether_header`.

3.2 IP

Un pachet IP poate avea ca destinatie una din adresele IP ale routerului sau poate fi destinat in alta retea. Pachetele cu destinatia diferita de interfetele routerului urmaresc procesul normal

de forwarding.

În headerul de sistem `netinet/ip.h`, puteți găsi o descriere C a antetului de Ethernet sub forma structurii `struct iphdr`.

3.3 ARP

Protocolul ARP este utilizat pentru a determina adresa MAC a next hop-ului. Astfel, router-ul va trebui sa genereze mesaje de tip ARP REQUEST si ARP RESPONSE. Pentru a evita poluarea retelei cu cereri ARP va trebui sa stocati datele intr-o tabela ARP. Structura tabelei este la latitudinea voastra.

Cererile ARP vor fi trimise ca broadcast(ff:ff:ff:ff:ff:ff). Raspunsurile ARP sunt trimise catre adresa MAC a host-ului ce a facut cererea.

În headerul de sistem `net/if_arp.h`, puteți găsi o descriere C a antetului de ARP sub forma structurii `struct arphdr`.

În headerul de sistem `netinet/if_ether.h`, puteți găsi o descriere C a antetului necesar pentru un reply/request, sub forma structurii `struct ether_arp` (care conține un `struct arphdr` menționat mai sus).

Pentru a veni in ajutorul vostru, va oferim doua functii ajutatoare care pot genera pachete ARP, si care pot extrage headerul ARP dintr-un packet existent.

3.4 ICMP

Un pachet IP poate contine un mesaj ICMP. Routerul va trebui sa genereze urmatoarele mesaje ICMP ca raspuns:

- **Echo reply** (type 0) - Trimis ca raspuns la un echo request(ping). Trebuie sa raspundeti doar mesajelor ICMP care sunt adresate routerului.
- **Destination unreachable** (type 3, code 0) - Trimis in cazul in care nu exista ruta pana la destinatie, atunci cand pachetul nu este destinat routerului.
- **Time exceeded** - Trimis daca TTL-ul este 0.

În headerul de sistem `netinet/ip_icmp.h`, puteți găsi o descriere C a antetului de Ethernet sub forma structurii `struct icmphdr`.

Pentru a veni in ajutorul vostru, va oferim implementarea a doua functii ajutatoare: una care genereaza un pachet ICMP, si una care parseaza un pachet pentru a extrage headerul ICMP.

4 API suplimentar

Vă punem la dispoziție în schelet un API pentru folosirea unei cozi. Urmăriți comentariile din fișierul `include/queue.h`.

5 Cerinte tema

Descarcati arhiva temei si creati reteaua virtuala folosind comanda de mai jos. Aceasta va porni 7 terminale: 4 pentru hosts, 2 pentru routere si 1 pentru controller(nu vom interactiona cu el). Pe hosts se pot rula comenzi precum ping, wget, nc etc. Pe routere vom rula programul rezultat in urma rularii comenzii make. In cadrul temei vom lucra in fisierul router.c.

- Scrieti cod care parseaza tabela de rutare (10p). Daca folositi functia de parsare din laboratorul 4 (parser.o + parser.h), pierdeti cele 10p pentru aceasta cerinta.
- Implementati protocolul ARP (30p). Se puna la dispozitie urmatorul API optional(Va puteti folosi si implementarea propie). API-ul il regasiti in skel.h:

Listing 4: API ARP.

```
void send_arp(uint32_t daddr, uint32_t saddr, struct ether_header ↩
    *eth_hdr, int interface, uint16_t arp_op);
struct arp_header* parse_arp(void *buffer);
```

  Functia parse_arp presupune ca primeste un pachet ethernet valid (e.g. ii puteti transmite m.payload ca parametru, daca m este de tip packet). Daca packetul este de tip ARP, va intoarce un pointer catre structura arp_header continuta in pachet. Daca pachetul nu este de tip ARP, va intoarce NULL.

  Tema se poate preda si cu o tabela de ARP statica, cu pierderea punctajului aferent.

  În arhivă aveți inclusă o tabelă ARP statică. Puteți folosi funcția de parsare din laboratorul 4.

  Prezența acestui fișier în arhiva va opri checkerul din a rula teste de ARP, deci nu veți primi puncte pe ele. Dacă ați implementat protocolul ARP, nu adăugați fișierul `arp_table.txt`.

- Implementati procesul de dirijare (25p).

  Algoritmul de cautare in tabela de rutare trebuie sa aiba o complexitate mai buna decat O(N)(ne intereseaza sa nu implementati o solutie similara cu cea de la laborator, in timp linear), unde n este dimensiunea tabelei de rutare (15p). Implementarea unei cautari liniare duce la pierderea celor 15p.

  Atentie: In cazul in care un pachet trebuie trimis mai departe dar inainte de dirijare se face un ARP Request, atunci pachetul va fi pus intr-o coada si va fi dirijat dupa ce s-a primit ARP Reply-ul. Utilizarea unei cozi este obligatorie, in cazul in care aceasta nu este folosita, acest subpucnt va fi punctat cu 0.

- Implementati suport pentru protocolul ICMP (20p). Se puna la dispozitie urmatorul API optional(Va puteti folosi si implementarea propie). API-ul il regasiti in skel.h

Listing 5: API ICMP.

```
void send_icmp(uint32_t daddr, uint32_t saddr, uint8_t *sha, ↩
    uint8_t *dha, u_int8_t type, u_int8_t code, int interface, int↩
     id, int seq);
void send_icmp_error(uint32_t daddr, uint32_t saddr, uint8_t *sha,↩
     uint8_t *dha, u_int8_t type, u_int8_t code, int interface);
struct icmp_header * parse_icmp(void *buffer)
```

Functia parse_icmp presupune ca primeste un pachet ethernet valid (e.g. ii puteti transmite m.payload ca parametru, daca m este de tip packet). Daca pachetul este de tip ICMP, va intoarce un pointer catre structure ICMP header din pachet; altfel, va intoarce NULL.

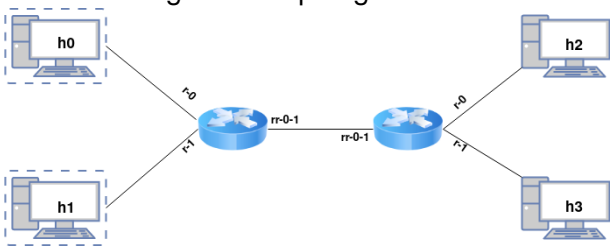
- BONUS: Implementati modificarea checksumului dupa decrementarea TTL folosind algoritmul incremental din RFC 1624. (10p)

Notă: Puteți scrie implementarea în C sau C++. Pe mașina virtuală de pe vmchecker, există gcc 7.5.0

6 Testare

Vom folosi mininet pentru a simula o rețea cu următoarea topologie:

Figure 1: Topologie Tema



Aveți la dispoziție un script de Python3, `topo.py`, pe care îl puteți rula pentru a realiza setupul de testare. Acesta trebuie rulat ca root:

```
$ sudo python3 topo.py
```

Astfel, se va inițializa topologia virtuală și se va deschide câte un terminal pentru fiecare host, câte un terminal pentru fiecare router și unul pentru controller (cu care nu vom interacționa); terminalele pot fi identificate după titlu. Din terminalele cu nume de forma "h-X", puteți folosi comenzi care generează trafic în IP pentru a testa funcționalitatea routerului implementat. Vă recomandăm arping, ping și netcat.

Pentru a rula routerele folosim urmatoarele comenzi, prima pe router 0 si a doua pe router 1:

```
./router rtable0.txt rr-0-1 r-0 r-1
```

```
./router rtable1.txt rr-0-1 r-0 r-1
```

Deasemenea, vă punem la dispoziție și o suită de teste, pe care o puteți rula cu argumentul "tests":

```
$ sudo python3 topo.py tests
```

În urma rulării testelor, va fi generat un folder `host_outputs` care conține, pentru fiecare test, un folder cu outputul tuturor hosților (ce au scris la `stdout` și `stderr`). În cazul unui test picat, s-ar putea să găsiți utilă informația de aici, mai ales cea din fișierele de `stderr`.

Notă: Scopul testelor este de a ajuta cu dezvoltarea și evaluarea temei. Mai presus de rezultatul acestora, important este să implementați *cerința*. Astfel, punctajul final poate diferi de cel tentativ acordat de teste, în situațiile în care cerința temei nu este respectată (un caz extrem ar fi hardcodarea outputului pentru fiecare test în parte). Vă încurajăm să utilizați modul interactiv al topologiei pentru a explora și alte moduri de testare a temei (e.g. `ping`, `arping`).

7 Trimitere

Tema trebuie trimisă pe vmchecker v2, la această adresă. Pentru trimitere este necesară o arhivă zip cu numele Nume_Prenume_Grupa.zip care să conțină, in radacina urmatoarele fisiere:

- `README`: în care să explicați pe larg abordarea implementării și problemele întâmpinate.
- Codul sursă al routerului
- `Makefile`: checkerul va rula `make` fără niciun argument

Nu este nevoie să includeți fișierele din checker sau tabela de rutare.

Notă: Nu folositi Vmchecker pentru a face debugging la tema. Tema trebuie dezvoltata si testata temeinic pe calculatorul vostru, cu trafic serios (e.g. iperf), cu multe intrari in tabela de routare (de exemplu 100K), si incarcata doar atunci cand sunteti convinsi ca este corecta.

Notă: Supraincarcarea Vmchecker prin submisii succesive ale aceeasi versiuni ale temei, poate duce la depunctare.

Notă: Orice probleme de punctare a temelor de catre Vmchecker, daca exista, vor fi solutionate de catre echipa de la PC pana punctajele finale vor fi anuntate. Nu vor fi acordate extensii ale termenului daca unele teste din Vmchecker sunt gresite; daca gasiti astfel de teste, va rugam sa le semnalati. Ele vor fi rezolvate pana la anuntarea notelor finale pe tema.

Deadline

Tema trebuie trimisa pana pe data de 18 aprilie 2021, ora 23:59, pentru obtinerea punctajului total de 100p. Pentru fiecare zi intarziere dupa acest termen, punctajul maxim pentru tema

scade cu 10p. Numarul maxim de zile de intarziere este 7.

Termenul este hard: nu vor fi acordate nici un fel de extensii.

Dependinte tema

Pentru a simula o retea virtuala in care sa testam ruterul nostru, vom folosi emulatorul de retea Mininet. Vom avea nevoie de urmatoarele pachete: mininet si openvswitch-testcontroller. Tema va fi rulata pe Linux (recomandam Ubuntu 18.04). Aici puteti gasi o masina virtuala cu Ubuntu 18.04.

Listing 6: Instalare software necesar pentru tema

```
sudo apt install mininet openvswitch-testcontroller xterm python3-pip
sudo cp /usr/bin/ovs-testcontroller /usr/bin/ovs-controller
sudo pip3 install scapy
sudo pip3 install pathlib
sudo pip3 install git+https://github.com/mininet/mininet.git
```

8 FAQ

- Nu imi apar terminalele cand rulez mininet.

```
sudo apt install xterm
```

- Cum pornesc mai multe terminale?

```
xterm &
```

- In cazul in care portul este ocupat rulati sudo fuser -k 6653/tcp

```
Exception: Please shut down the controller which is running on ↩
    port 6653
```

- Daca aveti eroarea de mai jos trebuie sa instalati openvswitch-testcontroller si creat fisierul /usr/bin/ovs-controller

```
raise Exception( 'Could not find a default OpenFlow controller' )
```

- Cum extrag header-ul IP dintr-un pachet de tip msg?

```
struct iphdr *ip_hdr = (struct iphdr *)(packet.payload + sizeof(struct ether_header));
```

- Cum pot vedea traficul de pe interfata X?

```
tcpdump -n -i X
```

- Cum pot reprezenta o adresa IP in memorie?

```
uint32_t
```

- Cum afisez interfetele unui host?

```
ip a s
```

- Valorile primite pe retea nu coincid cu valorile la care ma asteptam, ce fac?

  Pachetele primite sunt in format big endian, folositi urmatoarele functii pentru a face trecerea intre Little Endian si Big Endian.

```
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

- Cum initializez o coada?

```
queue q;
q = queue_create();
```

- Cum adaug un element in coada?

```
queue_enq(q, packet);
```

- Am implementat totul si nu imi trec testele.

  In cazul in care aplicati diferite operatii asupra tabelei de rutare la etapa de preprocesare, aveti grija ca acestea sa aiba o complexitate O(n log n) deoarece testarea temei incepe dupa 2 sec de initializare.

- Cum determin adresa IP si MAC-ul unei interfete a routerului?

```
/* Intoarce adresa in format zecimal, e.g. "192.168.2.1" */
char *get_interface_ip(int interface);
/* Adresa e intoarsa prin parametrul de iesire mac; sunt intorsi ↩
   octetii. */
void get_interface_mac(int interface, uint8_t *mac);
```

- Nu văd output de la router (`router_output.txt` este gol)

```
/* fist instruction in main from router.c */
setvbuf(stdout, NULL, _IONBF, 0);
```

- Când rulez `make` primesc o eroare ce conține "cannot open output file router: No such file or directory".

```
$ make distclean
$ make
```

- In WireShark primesc "Internet Protocol, bogus version" Aceasta eroare inseamna ca completarea header-ului IP/ICMP nu este corecta.
- Putem folosi C++? Da.
- Ce biblioteci sunt permise? Este permisa folosirea oricarei functii din biblioteca standard de C++. Mai mult, puteti folosi orice header standard de Linux, cu precizarea că trebuie în continuare să rezolvi manual cerința. De exemplu, Linux știe să răspundă singur la ICMP; pe router această funcționalitate este dezactivată, ca s-o implementați voi. Reactivarea ei cu apeluri din C nu reprezintă o soluție validă și nu va fi punctată.
- Bonsul este verificat de checker? Nu. Pentru bonus trebuie specificat in README.
- Primesc eroarea "[14]: ioctl SIOCGIFINDEX No such device" cand incerc sa rulez pe masina virtuala de Linux.

  Routerul caută anumite interfețe după nume; acestea probabil nu există pe sistemul vostru, sunt create de mininet. Asigurați-vă că rulați binarul de outer dintr-un terminal de router după pornirea topologiei.
- Cand incerc sa accesez informatia din headerul ICMP, in timp ce dau ping de la un host la altul, toate informatiile din header sunt 0. Spre deosebire de laborator, in tema trebuie implementat si protocolul ARP. In cazul de fata, se incearca extragerea header-ul ICMP dintr-un pachet ce contine doar Ethernet + ARP.

- Pentru protocolul ARP, cu ce trebuie initializate sender_mac, sender_ip, target_mac si sender_ip? Sau cum as putea sa le aflu? Nu am inteles principiul de functionare. Ati putea de asemenea sa explicati flow-ul acestui protocol si ce ar trebui sa facem mai exact?

```
h0 vrea sa dea ping la h1
h0 trimite ARP request la router, routerul ii trimite ARP reply cu adresa sa MAC
h0 completeaza headerul de Ethernet si trimite pachetul la router
routerul cauta in tabela IP-ul destinatie si vede ca trebuie trimis lui h1
routerul trimite ARP request la h1, h1 ii trimite ARP reply cu adresa sa MAC
routerul completeaza headerul de Ethernet si trimite pachetul la h1
```

- Cum pot determina din cod adresa MAC a unei interfețe?

  Aveți o funcție ajutătoare `get_interface_mac`, care primește o interfață și-i întoarce adresa MAC. Interfața pe care a venit pachetul trebuie extrasă din structura `msg`.

- La timeout is unreachable primesc urmatoarea eroare:

```
File "./checker.py", line 38, in passive
status = fn(testname, packets)
File "/media/sf_Shared_Folder/PC/tema1/tests.py", line 351, in icmp_timeout_p
assert ICMP in packets[1], "no ICMP packet from router"
```

  Nu trimiți înapoi un pachet de tip ICMP (probabil nu setezi corect câmpul protocol din headerul IP).

- Pe local am mai multe puncte decat pe vmchecker

  Aceasta problema poate aprea atunci cand implementarea voastra are o performanta scazuta(e.g. faceti sortare la fiecare apel de LPM). Pentru a rezolva aceasta problema, asigurati-va ca aveti o impementare decenta din punct de vedere al performantei.

- Imi pica ultimele 2 teste. Ultimele doua teste o sa pice, in general, daca faceti LPM ineficient.

- Cum folosesc API-ul de ARP si ICMP?

```
// De exemplu, trimitem un ICMP Destination unreachable(3, 0) vom face urmatorul apel
```

```
send_icmp(ip_sursa, ip_destinatie, mac_destinatie, mac_sursa, 3, ↩
    0, interface, id, sequence);

// Pentru a extrage un header ICMP dintr-un frame primit:
packet m;
rc = get_packet(&m);
struct icmphdr *icmp_hdr =  parse_icmp(m.payload);
// Check if the header has been extracted succesfully
if (icmp_hdr != NULL) {
    ...
}
```

```
// De exemplu, pentru a trimite un ARPOP_REPLY(sau ARPOP_REQUEST)
void send_arp(ip_sursa, ip_destinatie, header_ethernet, interface,↩
    ARPOP_REPLY)

// Pentru a extrage un header ICMP dintr-un frame primit:
packet m;
rc = get_packet(&m);
struct arp_header* arp_hdr = parse_arp(m.payload)
// Check if the header has been extracted succesfully
if (arp_hdr != NULL) {
    ...
}
```

References

[1] RFC 791 - Internet Protocol

[2] RFC 826 - An Ethernet Address Resolution Protocol

[3] RFC 894 - A Standard for the Transmission of IP Datagrams over Ethernet Networks

[4] RFC 792 - Internet Control Message Protocol

[5] RFC 1624 - Computation of the Internet Checksum via Incremental Update