Prezentare generală

În cadrul temei vom implementa dataplane-ul unui router. Un router are două părți:

- Dataplane - partea care implementează procesul de dirijare propriu-zis, pe baza intrărilor din tabela de routare.
- Control plane - partea care implementează algoritmii de rutare (e.g. RIP, OSPF, BGP); acești algoritmi distribuiți calculează intrările din tabela de rutare.

Tema constă în implementarea componentei de dataplane. În cele ce urmează, în lipsa altor precizări, toate referințele la router din textul temei se referă la dataplane.

NU va fi nevoie să implementați acești algoritmi în cadrul temei; vom lucra strict din perspectiva dataplane-ului, pentru care tabelele de rutare sunt deja populate. Routerul nostru va funcționa cu o tabelă de rutare statică, primită într-un fișier de intrare, și care nu se va schimba pe parcursul rulării.

Un router are mai multe interfețe și poate recepționa pachete pe oricare dintre acestea. Routerul trebuie să transmită pachetul mai departe, către un calculator sau către alt router direct conectat, în funcție de regulile din tabela de rutare.

Tema poate fi implementată în C sau C++.

Lectură tema

- The Internet Protocol (9 min)
- ARP: Mapping between IP and Ethernet (10 min)
- Looking at ARP and ping packets (8 min)
- Hop-by-hop routing (13 min)
- Capitolul 4.3 (What's Inside a Router) din Computer Networking: A Top-Down Approach (6th Edition)
- Capitolul 4.4.3 (Internet Control Message Protocol (ICMP)) din Computer Networking: A Top-Down Approach (6th Edition).
- Capitolul 5.4.1 (Link-Layer Addressing and ARP) din Computer Networking: A Top-Down Approach (6th Edition)

Deadline

Deadline-ul temei este specificat pe moodle.

Vă reamintim părțile relevante din regulamentul cursului de PCom:

- După expirarea acestui termen limită se mai pot trimite teme un interval de maxim 3 zile, cu următoarele depunctări: 10p în prima zi, 20p în a două zi și 30p în a treia zi.
- După cele 3 zile tema nu se mai poate trimite.
- Oferim posibilitatea fiecărui student de a avea un număr de maxim **5 zile** numite "sleep days".
- Aceste zile pot fi folosite pentru a amâna termenul de predare al temei de casă (fără penalizări).
- Nu se pot folosi mai mult de **două** sleep days pentru o temă de casă.
- Pentru a utiliza aceste zile completați formularul de pe Moodle
- Temele de casă sunt *individuale*.

Setup

Pentru a simula o rețea virtuală vom folosi Mininet. Mininet este un simulator de rețele ce folosește în simulare implementari reale de kernel, switch și cod de aplicații.

```
sudo apt update
sudo apt install mininet openvswitch-testcontroller tshark python3-click python3-
scapy xterm
sudo pip3 install mininet
```

După ce am instalat Mininet, vom folosi următoarea comandă pentru a crește dimensiunea fontului în terminalele pe care le vom deschide.

```
echo "xterm*font: *-fixed-*-*-*-18-*" >> ~/.Xresources
xrdb -merge ~/.Xresources
```

Procesul de dirijare (forwarding)

Dirijarea este un proces care are loc la nivelul 3 ("Rețea") din stiva OSI. În momentul în care un pachet ajunge la router, acesta trebuie să efectueze următoarele acțiuni:

1. **Parsarea pachetului:** din fluxul de octeți ce reprezintă pachetul, routerul trebuie să afle ce antete sunt prezente și ce valori conțin câmpurile acestora, construind și inițializând structuri interne corespunzătoare. La acest pas, dacă routerul descoperă că un pachet primit e malformat (e.g. prea scurt), îl aruncă.
2. **Validarea L2:** pachetele conțin un antet de `Ethernet`; routerul nostru trebuie să considere doar pachetele trimise către el însuși (i.e. câmpul `MAC destination` este același cu adresa MAC a interfeței pe care a fost primit pachetul) sau către toată lumea (i.e. câmpul `MAC destination` este adresa de *broadcast*, `FF:FF:FF:FF:FF:FF`). Orice alt pachet trebuie aruncat.
3. **Inspectarea următorului header:** routerul inspectează câmpul `Ether Type` din header-ul Ethernet pentru a descoperi următorul antet prezent. În cadrul temei, ne vor interesa doar două posibilități: IPv4 și ARP. Următoarele acțiuni ale routerului vor depinde de tipul acestui header. Orice alt fel de pachet trebuie ignorat.

În continuare, vom studia protocoalele implicate în procesul de dirijare și vom prezenta cum acestea sunt folosite de către router.

Ethernet

Ethernet este echivalentul protocolului de DataLink pe care l-am implementat în primele laboratoare. Noi vom lucra doar cu **cadre Ethernet** ce sunt transmise ca payload peste implementarea protocolului de nivel fizic Ethernet. Cum CRC-ul este calculat în hardware, nu o să îl regăsim în header. În acest caz, header-ul pe care îl vom folosi este următorul:

```
Ethernet L2 Frame Header
+-----------------+------------+-------------+
|     Bytes 0-5   | Bytes 6-11 | Bytes 12-13 |
+-----------------------------+-------------+
| Destination MAC | Source MAC |  EtherType  |
+-----------------+------------+-------------+
```

Adresa MAC Destinație reprezintă identificatorul dispozitivului de nivel 2 către care a fost trimis acest cadru.

În cadrul laboratorului puteți folosi următoarea structura pentru un cadru Ethernet.

```
struct  ether_header {
    uint8_t  ether_dhost[6];    // MAC destinație
    uint8_t  ether_shost[6];    // MAC sursă
    uint16_t ether_type;        // folosit pentru a specifica protocolul de nivel
                                // superior care este encapsulat în Ethernet
};
```

> În RFC 5342 sunt definite valorile pe care ethertype le poate lua. În cazul nostru, ne interesează IPv4 (0x0800) și ARP (0x0806).

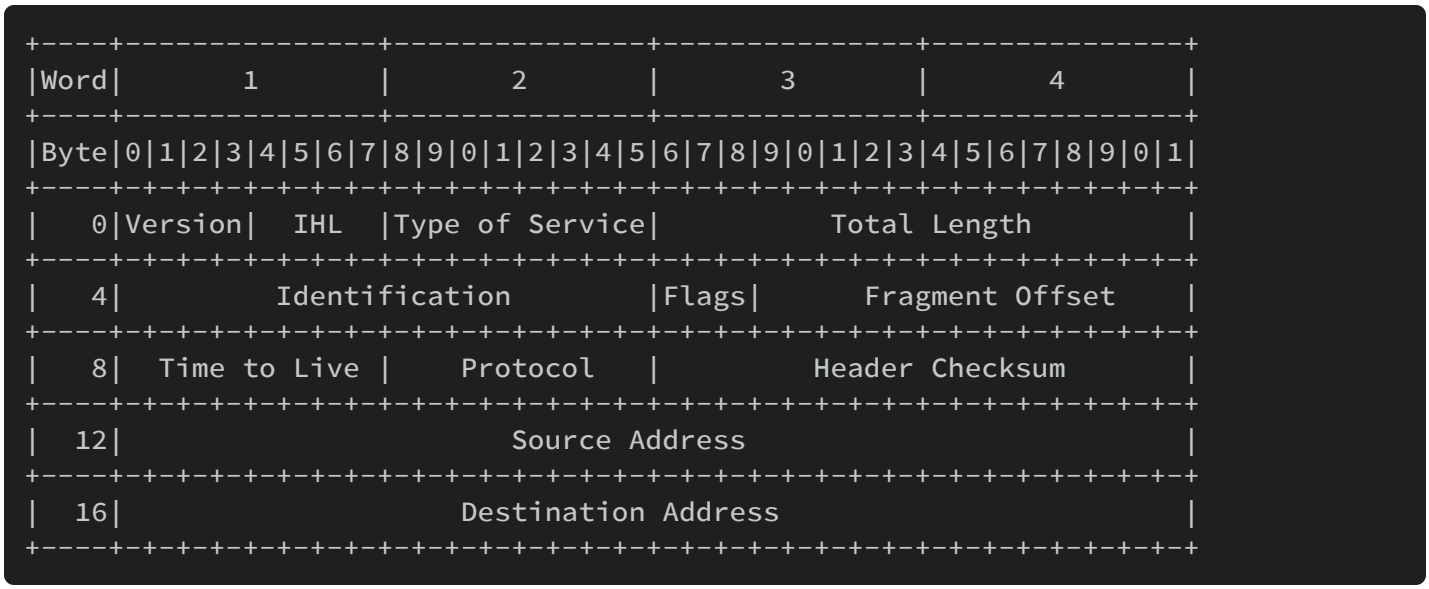
Noi vom folosi API-ul de nivel 2 pentru a trimite cadre Ethernet L2 peste protocolul de L1 Ethernet ce se ocupa cu framing, checksums etc. Aici găsiți o descriere completă a structurii Ethernet. În payload-ul cadrului de Ethernet vom avea encapsulat fie protocolul ARP, fie protocolul IP.

În figura de mai jos, găsim o captură Wireshark ce surprinde un cadru ethernet de L2 în care avem encapsulat IP și ICMP peste IP.

```
▸ Frame 130: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface enp37s0, id 0
▾ Ethernet II, Src: ASRockIn_7d:ee:86 (70:85:c2:7d:ee:86), Dst: Microsof_14:02:63 (00:15:5d:14:02:63)
    ▸ Destination: Microsof_14:02:63 (00:15:5d:14:02:63)
    ▸ Source: ASRockIn_7d:ee:86 (70:85:c2:7d:ee:86)
      Type: IPv4 (0x0800)
▸ Internet Protocol Version 4, Src: 172.19.2.160, Dst: 142.250.186.78
▸ Internet Control Message Protocol
```

Protocolul IPv4

Protocolul IP este utilizat pentru a permite dispozitivelor conectate în rețele diferite să schimbe informații prin intermediul unui dispozitiv intermediar numit router. O descriere completă a header-ului de IPv4 o găsiți aici. Header-ul unui pachet (packet) IP este următorul:



Următoarea strucură poate fi folosită pentru a reprezenta un pachet IPV4.

```
struct iphdr {
    // version este pe 4 biți, ihl este pe 4 biți
    uint8_t    version:4,
               ihl:4;     // Nu vom implementa protocolul IP cu opțiuni,
                          // așa că 5 × 32 bits = 160 bits = 20 bytes
                          // (internet header length)
    uint8_t    tos;         // Nu este relevant pentru temă (set pe 0)
    uint16_t   tot_len;     // dimensiunea totală a header + date
    uint16_t   id;          // Nu este relevant pentru temă, (set pe 1)
    uint16_t   frag_off;    // Nu este relevant pentru temă, (set pe 0)
    uint8_t    ttl;         // Time to live
    uint8_t    protocol;    // Identificator al protocolului encapsulat (e.g.
ICMP)
    uint16_t   check;       // checksum of the iphdr, we checksum = 0 when
computing
    uint32_t   saddr;       // Adresa IP sursă
    uint32_t   daddr;       // Adresa IP destinație
};
```

Nu suntem interesați de anumite funcționalități din IP precum fragmentarea. Astfel, în cazul în care construiți un pachet IP de la 0, vom seta următoarele:

```
tos = 0
frag_off = 0
version = 4
ihl = 5
id = 1
```

În RFC 990 pagina 24 găsiți o listă de identificatori pentru diversele protocoale ce pot fi encapsulate în IP. Vom folosi aceste valori pentru a completa câmpul `protocol`.

Când un router primește un pachet de tip IPv4, trebuie să realizeze următoarele acțiuni:

1. **Verifică dacă el este destinația:** deși un intermediar, routerul este de asemenea o entitate cu interfețe de rețea și adrese IP asociate acestora, deci poate fi destinatarul unui pachet. În acest caz, routerul nu trebuie să trimită mai departe pachetul, ci să îi înțeleagă conținutul pentru a putea acționa în consecință. În cadrul acestei teme, routerul va răspunde doar la mesaje de tip **ICMP**.

2. **Verifică checksum:** routerul trebuie să recalculeze suma de control a pachetului, și să o compare cu cea primită în antetul de IP; dacă sumele diferă, pachetul a fost corupt și trebuie aruncat.

3. **Verificare și actualizare `TTL`:** pachetele cu câmpul `TTL` având valoarea 1 sau 0 trebuiesc aruncate. Routerul va trimite înapoi, către emițătorul pachetului un mesaj ICMP de tip "Time exceeded" (mai multe detalii în secțiunea ICMP). Altfel, câmpul `TTL` e decrementat.

4. **Căutare în tabela de rutare:** routerul caută adresa IP destinație a pachetului în tabela de rutare pentru a determina adresa următorului hop, precum și interfața pe care va trebui scos pachetul. În caz că nu găsește nimic, pachetul este aruncat. Routerul va trimite înapoi, către emițătorul pachetului un mesaj ICMP de tip "Destination unreachable" (mai multe detalii în secțiunea ICMP).

5. **Actualizare checksum:** routerul recalculează suma de control a pachetului (această trebuie recalculată din cauza schimbării câmpului `TTL`) și o surpascrie în antetul IP al pachetului.

6. **Rescriere adrese L2:** pentru a forma un cadru corect care să fie transmis la următorul hop, routerul are nevoie să rescrie adresele de L2: adresa sursă va fi adresa interfeței routerului pe care pachetul e trimis mai departe, iar adresa destinație va fi adresa MAC a următorului hop. Pentru a determina adresa următorului hop, routerul folosește protocolul ARP.

7. **Trimiterea noului pachet pe interfața corespunzătoare următorului hop.**

Tabela de rutare

Fiecare router dispune de o "tabelă de rutare" -- o structură pe baza căreia alege portul pe care să emită un pachet. În mod normal, aceste tabele sunt populate de către control plane, în urma rulării unui algoritm de rutare (e.g. OSPF); în cadrul temei, vom folosi o tabelă statică.

O intrare în tabel are patru coloane:

- `prefix` și `mask`: împreună aceste două câmpuri formează o adresă de rețea (e.g. `192.168.1.0/24`)
- `next hop`: adresa IP a mașinii către care va fi trimis pachetul; dacă routerul nu e conectat direct la destinatar, aceasta va fi adresa unui router intermediar.
- `interface`: identificatorul interfeței pe care routerul va trebui să trimită pachetul respectiv către `next hop`.

```
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
```

În contextul temei, este suficient să considerați că numărul maxim de intrări din tabela de rutare este de 100 000 de intrări.

Longest Prefix Match

Pentru adresa IPv4 destinație din pachetul primit, routerul trebuie să caute intrările din tabela de routare care descriu o rețea ce cuprinde adresa respectivă. Este posibil ca mai multe intrări să se potrivească; în acest caz, routerul trebuie s-o aleagă pe cea mai specifică, i.e. cea cu masca cea mai mare. Acest criteriu de căutare se numește "Longest Prefix Match" (LPM).

Programatic, routerul poate verifică apartenența unei adrese la o rețea, făcând operația de `AND` pe biți și verificând că este egală cu prefixul:

```
ip.destination & entry.mask == entry.prefix
```

Odată ce routerul găsește toate prefixele care se potrivesc adresei destinație din pachet, va alege intrarea din tabela de rutare corespunzătoare celui mai lung prefix (masca cea mai mare).

Protocolul ARP

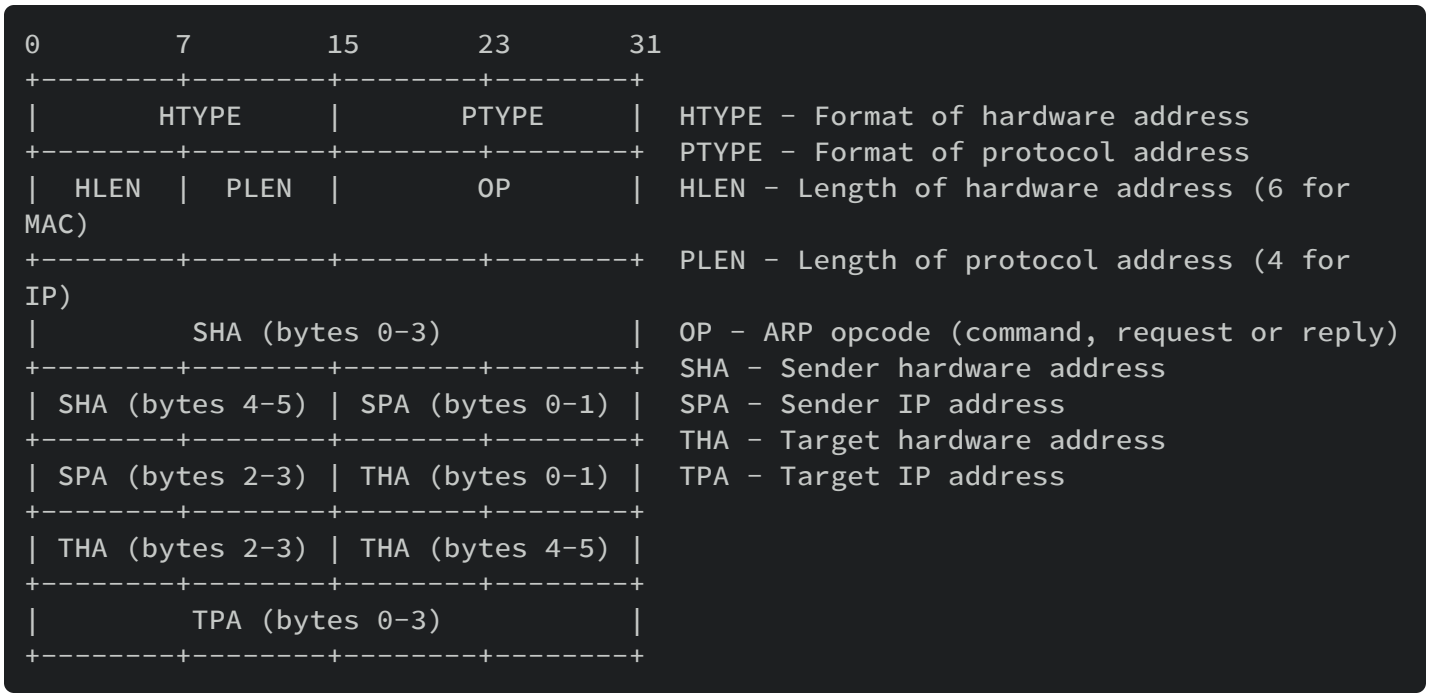
După ce un router a determinat următorul hop pentru un pachet folosind LPM, trebuie să-l trimită mai departe, încapsulându-l într-un cadru cu adresa MAC destinație setată ca cea a următorului hop. Cum știe routerul această adresă?

ta opțiune este ca această să fie reținută static, într-un tabel.

În realitate, însă, din mai multe motive (e.g. flexibilitate), routerul *nu știe* aceste adrese, ci trebuie să le determine folosind protocolul ARP (RFC 826). Având adresa IP a următorului hop precum și identificatorul interfeței care duce la acesta, routerul generează un mesaj de broadcast în rețeua respectivă, întrebând care e adresa MAC asociată acelei adrese IP. Mașina respectivă observă că este vorba de propriul său IP și îi trimite routerului un mesaj cu propria adresă MAC. Acum routerul poate forma cadrul corespunzător; deasemena, păstrează într-un cache adresa MAC primită, pentru un interval limitat de timp.

Protocolul ARP este deci relevant în două puncte: atunci când primim un pachet cu antet ARP și atunci când trebuie să rescriem pentru forwardare un pachet IPv4.

Antetul protocolului ARP este următorul:



Următoarea structura poate fi folosită pentru a reprezenta acest header în C:

```c
/* Ethernet ARP packet from RFC 826 */
struct arp_header {
    uint16_t htype;   /* Format of hardware address. */
    uint16_t ptype;   /* Format of protocol address. */
    uint8_t hlen;    /* Dimensiunea adrese hardware 6 bytes pentru MAC */
    uint8_t plen;    /* Dimensiunea adresei protocolului (IPV4). 4 bytes petnru IP
*/
    uint16_t op;    /* ARP opcode */
    uint8_t sha[6];  /* Adresa hardware (MAC) sender */
    uint32_t spa;   /* Adresa IP sender */
    uint8_t tha[6];  /* Adresa hardware target */
    uint32_t tpa;   /* Adresa IP target */
} __attribute__((packed));
```

> Noi vom folosi ARP pentru a determina adresa MAC a unui host având adresa IPv4 a acestuia. Parametrii folosiți de ARP precum `op` sau `htype` îi găsiți documentati de IANA. Cum facem translația din adrese IPv4, în `ptype` vom folosi identificatorul IPv4 conform RFC 5342. `hlen` reprezintă dimensiunea adresei hardware, in acest caz adresa MAC are `6 bytes`. `plen` reprezintă adresa protocolului de nivel network, în acest caz `IPv4`.

Exemplu de pachet de tip ARP request capturat cu Wireshark:

```
Ethernet II, Src: Micro-St_57:ff:f6 (d8:bb:c1:57:ff:f6), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: Micro-St_57:ff:f6 (d8:bb:c1:57:ff:f6)
    Sender IP address: 192.168.0.150
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 169.254.255.255
```

Pașii relevanți pentru dirijarea unui pachet IPv4 sunt următorii:

1. **Căutare în cache:** routerul se uită în cache-ul său ARP pentru a vedea dacă există o intrare curentă pentru adresa IPv4 a următorului hop. Dacă aceasta există, routerul o ia din cache, rescrie antetul L2 al pachetului și îl trimite mai departe.

2. **Salvare pachet pentru mai târziu:** dacă adresa necesară nu se găsește în cache-ul ARP, routerul va trebui să facă o interogare generând un pachet ARP și așteptând un răspuns. Pachetul original care trebuia dirijat este adăugat într-o coadă, pentru a putea fi trimis mai târziu, după sosirea răspunsului ARP.

3. **Generare ARP request:** routerul generează un pachet de tip ARP pentru a interoga despre adresa MAC a mașinii cu adresa IPv4 a următorului hop. Pentru asta are nevoie să genereze un pachet cu un antet Ethernet, urmat de un antet ARP.

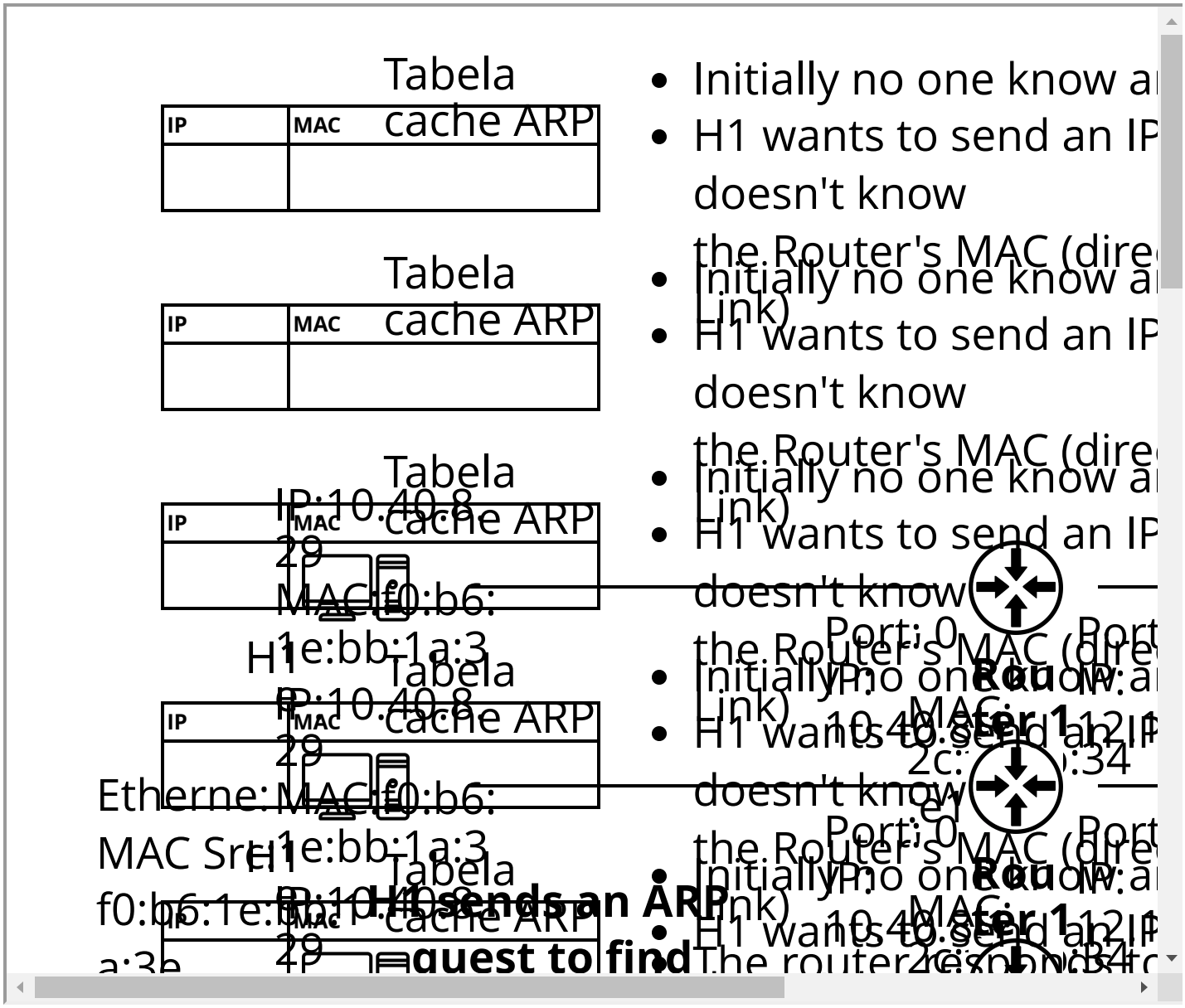
- **Antetul Ethernet:** trebuie să conțină un `ethertype` care să identifice un pachet de tip ARP (`0x806`). Adresa MAC sursă va fi adresa interfeței routerului către next hop; adresa MAC destinație va fi cea de broadcast (`FF:FF:FF:FF:FF:FF`).

- **Antetul ARP:** trebuie să conțină tipul de adresă folosit în căutare (IPv4) împreună cu adresa în sine, precum și tipul de adresă căutată (MAC) împreună cu adresa în sine. Pentru mai multe detalii, consultați [@arprfc]

4. **Parsează ARP reply.** Atunci când routerul primește un pachet de tip ARP reply, îl va adăuga în cache-ul ARP local. În plus, routerul va parcurge lista de pachete care așteaptă răspunsuri ARP și le va trimite pe cele pentru care adresa următorului hop este cunoscută.

Un `ARP Request` către router poate avea ca destinație broadcast sau adresa MAC a interfeței router-ului.

În slideshow-ul de mai jos găsiți un exemplu de utilizare a protocolului ARP pentru a determina adresa MAC a unui dispozitiv la care suntem direct conectați.



În mod normal, hoștii sunt conectați la un switch, astfel ARP Request-ul este trimis la mai mai multe dispozitive, nu doar la router.

Protocolul ICMP

Protocolul ICMP descris în RFC 792 este folosit de routere în procesul de dirijare pentru a transmite informații legate de conectivitatea la nivel IP și transport gazdelor care au trimis datagrama în cauza. Protocolul ICMP se situează deasupra IP în stiva de protocoale, dar nu este un protocol de transport, ci un protocol de control (i.e. de debugging) pentru IP.

De exemplu, în anumite situații în care un pachet este aruncat de router, un mesaj ICMP este generat de router și trimis către expeditorul pachetului. În cadrul temei, ne vor interesa doar următoarele situații și mesaje:

- **Destination unreachable** (type 3, code 0) - Trimis în cazul în care nu există rută până la destinație, atunci când pachetul nu este destinat routerului.
- **Time exceeded** (type 11, code 0) - Trimis dacă pachetul este aruncat din cauza expirării câmpului `TTL`.

Pentru ambele tipuri de mesaj de eroare, pachetul emis de router trebuie să conțină, deasupra headerului ICMP, headerul de IPv4 al pachetului dropped, precum și primii 64 de biți din **payload-ul** pachetului original (adică doar ce se află deasupra antetului IPv4).

De asemenea, routerul fiind și el o entitate în rețea, poate primi mesaje ICMP de tip `"Echo request"` (type 8, code 0) destinate lui însuși. Acesta trebuie să răspundă cu un mesaj ICMP de tip `"Echo reply"` (type 0, code 0). În cazul acestor mesaje, octeții 4-5 din header capată semnificația unui "număr de identificare", iar octeții 6-7 ai unui "număr de secvență". Interpretarea acestora este de datoria hostului care a emis pachetele ICMP, routerul trebuie doar să se asigure că păstreze aceleași valori în pachetul de tip "Echo reply". Routerul trebuie să trimită înapoi și orice date care se aflau deasupra antetului ICMP în pachetul original.

Header-ul ICMP este următorul:

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Depends on type and code                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

De exemplu, din RFC 792 pagina 6, aflăm că răspunsul de tip `Time Exceeded Message` o să conțină la final, header-ul IP al pachetului aruncat și 64 de biți din datele cărate de IP, în acest caz un ICMP echo. Aveți mai jos o reprezentare a unui astfel de răspuns.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |     Type      |     Code      |          Checksum             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           unused                              |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |      Internet Header + 64 bits of Original Data Datagram      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

> Pentru a înțelege mai bine funcționalitatea, vă recomandăm să urmăriți cu Wireshark răspunsurile routerului din casă. De exemplu, daca rulăm `ping -c 1  -t 1 8.8.8.8` de pe calculatorul nostru, vom putea observa raspunsul de tip `Time Exceeded Message` trimis de către router.

Putem folosi următoarea structura pentru a reprezenta header-ul ICMP:

```c
struct icmphdr
{
  uint8_t type;                /* message type */
  uint8_t code;                /* type sub-code */
  uint16_t checksum;           /* checksum header */
  union
  {
    struct
    {
      uint16_t        id;
      uint16_t        sequence;
    } echo;                          /* echo datagram. Vom folosi doar acest câmp
din union */
    uint32_t        gateway;         /* Nu este relevant pentru tema */
    struct
    {
      uint16_t        __unused;
      uint16_t        mtu;
    } frag;                          /* Nu este relevant pentru tema */
  } un;
};
```

O captură de pachet ICMP request în Wireshark este următoarea:

```
▸ Frame 130: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface enp37s0, id 0
▾ Ethernet II, Src: ASRockIn_7d:ee:86 (70:85:c2:7d:ee:86), Dst: Microsof_14:02:63 (00:15:5d:14:02:63)
  ▸ Destination: Microsof_14:02:63 (00:15:5d:14:02:63)
  ▸ Source: ASRockIn_7d:ee:86 (70:85:c2:7d:ee:86)
    Type: IPv4 (0x0800)
▸ Internet Protocol Version 4, Src: 172.19.2.160, Dst: 142.250.186.78
▸ Internet Control Message Protocol
```

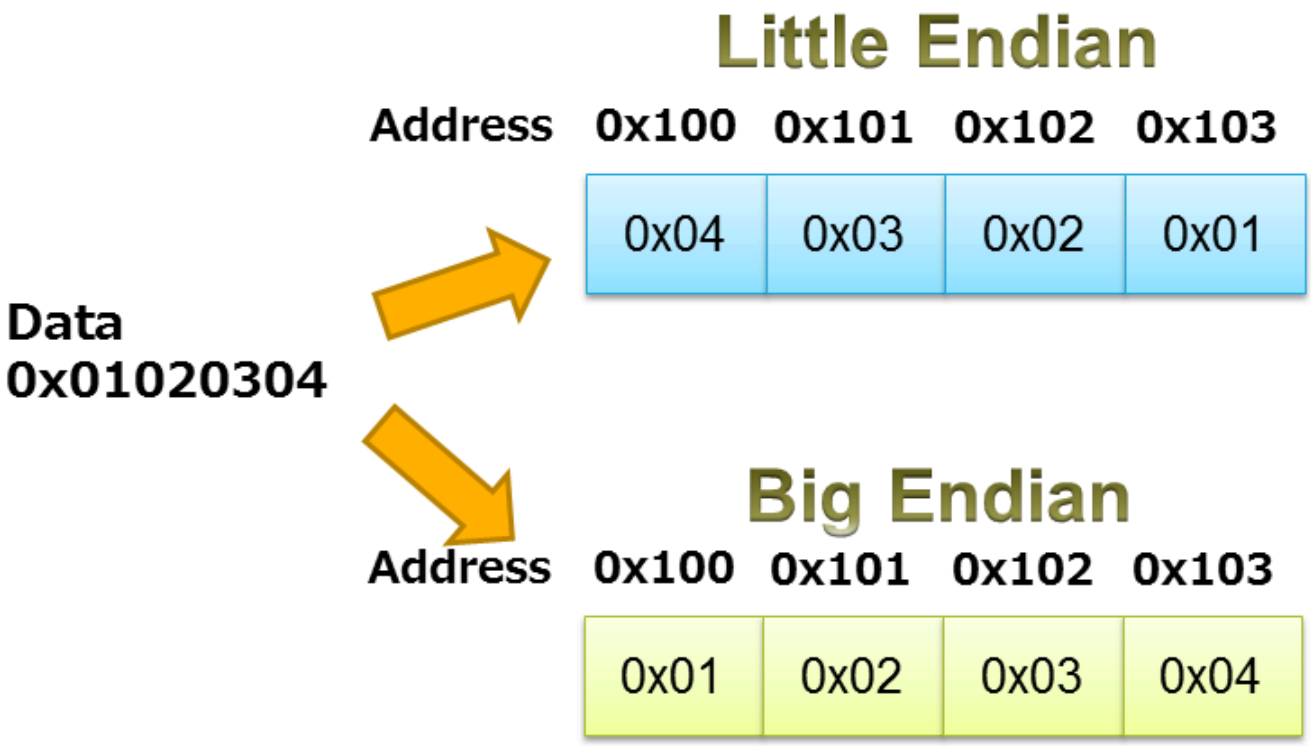
Endianness

Toate datele pe care le vom primi de pe un link sunt în `Network Order`, pe când procesorul de pe calculatorul nostru lucrează în `Host Order`. Astfel, când o să afișăm tipuri de date mai mari de un byte, va trebui să le trecem în host order cu funcții precum `ntohl`.

În același fel, atunci când completăm diferitele headere, va trebui să o facem în `Network Order`, altfel celelalte dispozitive din internet care folosesc această reprezentare le vor interpreta greșit. De exemplu, atunci când completăm câmpul `ethertype` din header-ul Ethernet, vom folosi `eth_hdr->ether_type = htons(ETHERTYPE_ARP);` unde `ETHERTYPE_ARP` este `0x0806`.

```
#include <arpa/inet.h>

// host to network long
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
// network to host long
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort)
```



API

Pentru rezolvarea temei vă punem la dispoziție un schelet de cod care implementeze unele funcționalități esențiale pentru rezolvarea cerințelor, precum și unele funcții ajutătoare a căror utilizare este opțională. În **protocols.h** găsiți structuri pentru protocoalele cu care vom interacționa. În **lib.h** veți găsi mai mulți funcții auxiliare utile, printre care:

- **Recepționare/trimitere pachete:** aveți la dispoziție următoarele două funcții de la nivelul Datalink:

```
/* Scrie în frame_data continutul unui cadru L2 de Ethernet. În cazul nostru,
frame_data
   va fi structurat astfel:

   |Alte protocoale encapsulate de IP (e.g. ICMP)|
   |IP or ARP|
   |Ethernet L2 frame| |<IP or ARP>|

   Returnează interfața pe care cadrul a fost primit. Funcția este blocantă.
 */
int recv_from_any_link(char *frame_data, size_t *length);

/* Trimite frame_data ca payload al unui cadru Ethernet L2.
   Link-ul pe care îl va trimite este identificat de interfață.
 */
int send_to_link(int interface, char *frame_data, int length);
```

Pentru a primi punctajul pe temă, trebuie să folosiți acest API de send/recv

---

- **Intrări în tabela de routare:** puteți modela o intrare în tabela de routare folosind următoarea structură:

```
/* Route table entry */
struct route_table_entry {
    uint32_t prefix;
    uint32_t next_hop;
    uint32_t mask;
    int interface;
} __attribute__((packed));
```

- **Parsare tabela de routare:** pentru a parsa tabela de routare, puteți folosi funcția:

```
int read_rtable(const char *filepath, struct route_table_entry *rtable);
```

Intrările din tabela de rutare sunt deja in network order.

- **Intrări în tabela ARP:** puteți modela o intrare în tabela ARP folosind următoarea structură:

```
struct arp_table_entry {
    uint32_t ip;
    uint8_t mac[6];
};
```

- **Parsare tabela statică ARP:** în cazul în care doriți să folosiți tabela statică de ARP, puteți să o parsați folosind funcția:

```
int parse_arp_table(char *path, struct arp_table_entry *arp_table);
```

- **Calcul sume de control:** pentru a realiza calcularea/verificarea sumelor de control din IPv4, respectiv ICMP, puteți folosi următoarea funcție:

```
/* Atunci cand calculam checksum-ul header-ului, vom pune
   campul checksum din header pe 0. */
uint16_t checksum(uint16_t *data, size_t len);
```

- **Coadă de pachete:** după cum este menționat în descrierea protocolului ARP, veți avea nevoie să folosiți o coadă pentru pachete. Vă punem la dispoziție implementarea unei cozi cu elemente generice; urmăriți comentariile din `include/queue.h`.

- **Determinarea MAC interfață proprie:** pentru a determina adresa MAC a unei interfețe a routerului, folosiți funcția:

```
void get_interface_mac(int interface, uint8_t *mac);
```

Argumentul `mac` trebuie să indice către o zonă de memorie cu cel puțin șase octeți alocați.

Cerințe temă

În acest repo găsiți scheletul temei, infrastructura și checker-ul automat.

Pentru rezolvarea temei, trebuie să implementați dataplane-ul unui router. **Va recomandăm să folosiți cel puțin `ping` pentru a testa implementarea și `Wireshark` sau `tcpdump` pentru depanare și analiza corectitudinii.** Punctajul este împărțit în mai multe componente, după cum urmează:

- **Procesul de dirijare (30p).** Va trebui să implementați pașii prezentați în secțiunea IPv4. Pentru acest exercițiu nu este nevoie să implementați și funcționalitatea referitoare la ICMP și puteți folosi o tabela statică de ARP.
- **Longest Prefix Match eficient (16p).** La laborator, am implementat LPM folosind o căutare liniară, i.e. trecând de fiecare dată prin toate întrările din tabel. În practică, o tabelă de routare poate conține foarte multe întrări [1] -- astfel de căutare este ineficientă. O abordare mai bună este să folosim o trie.

  Orice implementare mai eficientă decât căutarea liniară valorează 16 puncte.
- **Protocolul ARP (33p).** Vom implementa pașii din secțiunea ARP pentru a popula dinamic tabela ARP. Va trebui să implementați și funcționalitatea de caching; după ce primiți un răspuns ARP, rețineți adresa MAC a hostului interogat. În realitate, intrările dintr-un cache sunt temporare, fiind șterse după un anumit interval de timp. Pentru implementarea temei, este suficient să folosiți **intrări permanente**. **Pentru a nu bloca routerul, pachetele pentru care așteptăm un `ARP Reply` vor fi puse într-o coadă.**

  Vă recomandăm să folosiți comanda `arping` pentru a face cereri ARP. Comanda `arp` afișează tabela ARP a unui host.

  Tema se poate preda și cu o tabelă de ARP statică, cu pierderea punctajului aferent. În arhivă aveți inclusă o astfel de tabelă. **Prezența acestui fișier în arhivă 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`.**

  Este normal ca testul forward să pice daca folositi tabela statica de ARP.

- **Protocolul ICMP (21p).** Implementați pe router funcționalitatea descrisă în secțiunea ICMP. Puteți folosi `ping -t 1` pentru a trimite un pachet cu TTL 1. Va recomandăm să

testați și cu `traceroute`.

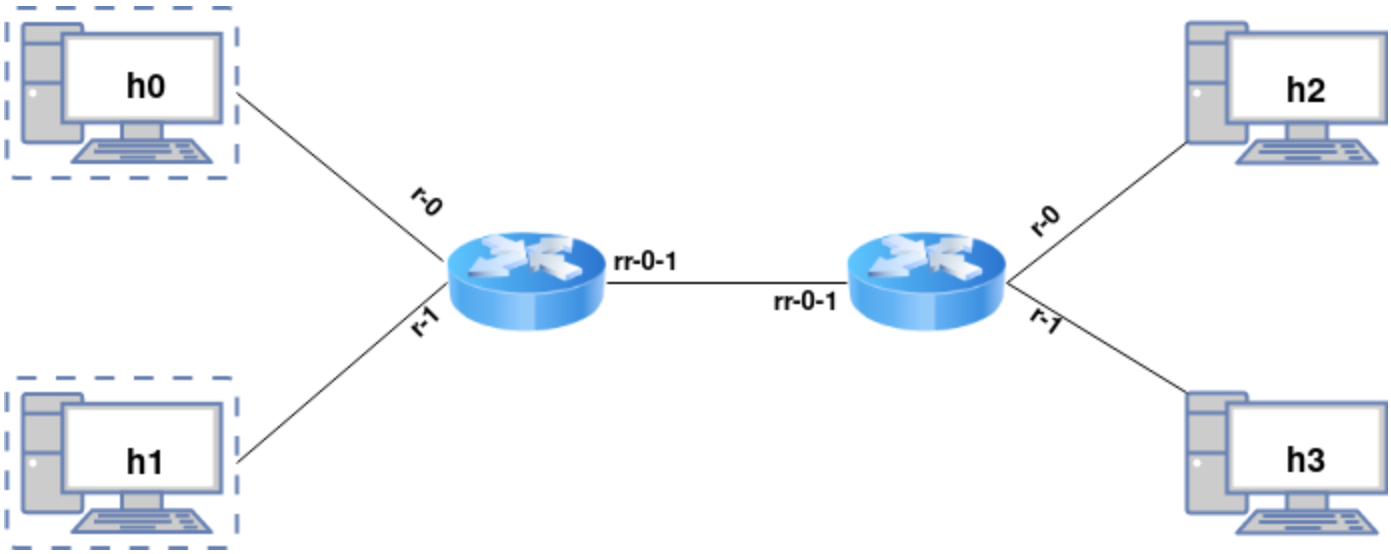
Pentru a fi punctată o temă, în README trebuie prezentată soluția voastră, pe scurt. De asemenea, trebuie menționat și ce subcerințe ați rezolvat.

Notă: Puteți scrie implementarea în C sau C++.

[1] https://blog.apnic.net/2022/01/06/bgp-in-2021-the-bgp-table/

Testare

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



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 checker/topo.py
```

Astfel, se va inițializa topologia virtuală și se va deschide câte un terminal pentru fiecare host și câte un terminal pentru fiecare router; terminalele pot fi identificate după titlu.

Fiecare host e o simplă mașină Linux, din al cărei terminal puteți rula comenzi care generează trafic IP pentru a testa funcționalitatea routerului implementat. Vă recomandăm arping, ping și netcat. Mai mult, din terminal putem rula `wireshark` sau `tcpdump` pentru a face inspecția de pachete.

Pentru a compila codul vom folosi `make` care creează binarul `router`.

Pentru a porni routerele manual folosim următoarele comenzi, prima pe router 0 și a doua pe router 1:

```
make run_router0    # din terminalul lui router0
make run_router1    # din terminalul lui router1
```

Ca sa nu scrieti manual ip-ul unui host, puteti folosii `h0`, `h1`, `h2` si `h3` in loc de IP. (e.g. `ping h1`)

Testare automată

> Înainte de a folosi testele automate, vă recomadam să folosiți modul interactiv al temei pentru a vă verifică corectitudinea implementării. Testarea automată durează câteva minute, așa că este mult mai rapid să testați manual.

Deasemenea, vă punem la dispoziție și o suită de teste:

```
$ ./checker/checker.sh
```

În urma rulării testelor, va fi generat un folder `host_outputs` care conține, pentru fiecare test, un folder cu outputul tuturor hoștilor (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`. Folderul conține și câte un **fișier pcap** pentru fiecare router, pe care îl puteți inspecta apoi în `Wireshark` (captura este făcută pe toate interfețele routerului, deci pachetele dirijate vor apărea de două ori; urmăriți indicațiile de aici pentru a obține o vizualizare mai bună).

Puteți rula un singur test folosind argumentul run . De exemplu:

```
sudo python3 checker/topo.py run router_arp_reply
```

Nu veți primi un raport *PASS/FAIL*, dar veți putea obține ușor outputul din `host_output`.

> **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`).

Descrierea testelor automate este următoarea:

```
router_arp_reply - De pe h0 se trimite un ARP request către router; se verifică că routerul a trimis un reply; (practic un arping)
router_arp_request - De pe h0 se trimite un pachet către h1; ca routerul să-l poată livra, trebuie deci să trimită un ARP request către h1 - acesta e singurul verificat (nu se verifică dacă pachetul trimis de h0 chiar a ajuns pe h1)
forward - De pe h0 se trimite un pachet către h1, se verifică că a ajuns
forward_no_arp - De pe h0 se trimit, pe rând, două pachete către h1; pentru că routerul trebuie să reţină informaţiile primite de la un ARP reply, se verifică că pe h1 ajunge *cel mult* un ARP request
ttl - La fel că "forward", dar se verifică şi dacă TTL-ul a fost decrementat
checksum - La fel că "forward", dar se verifică şi dacă checksumul e corect
wrong_checksum - De pe h0 se trimite pe h1 un pachet IP cu checksum greşit; se verifică că nu ajunge pe h1
forwardXY - Se trimite un pachet de la hX la hY; în rest, identic cu "forward"
router_icmp - De pe h0, se trimite un ICMP echo request către router; se verifică că routerul răspunde
icmp_timeout - Se trimite de pe h0 un pachet cu TTL=1; se verifică că routerul răspunde cu ICMP time-exceeded
host_unreachable - Se trimite de pe h0 un pachet către un host necunosct; se verifică că routerul răspunde cu ICMP host-unreachable
forward10packets - Se trimit 10 pachete de la h0 la h1
forward10across - Se trimit 10 pachete de la h0 la h3 (deci trebuie să treacă prin două routere)
```

Trimitere

Pentru a fi notată în catalog, tema va fi trimisă pe Moodle, unde checker-ul va fi rulat automat și va pune la feedback nota și output-ul rulării. O rulare completă durează cam 6 minute. Se vor trimite doar arhive generate folosind scriptul `create_arhive.sh` din schelet. Vă rugăm să trimiteți tema pe Moodle doar după ce ați verificat că aceasta rulează bine pe local

FAQ & probleme

- **Q:** Pe local am mai multe puncte decât pe checkerul online

  A:

  - Problema poate apărea atunci când implementarea voastră are o performanță scăzută (e.g. faceți sortare la fiecare apel de LPM). Pentru a rezolva problema, asigurați-vă că aveți o implementare bună din punct de vedere al performanței.
  - Verificați ca tabela de rutare să fie conform celei din schelet
  - Aveți funcții care vă scad performanța codului (e.g. printf)
  - Aveți undefined behaviour care este vizibil doar pe checker. În acest caz ar trebui să folosiți valgrind și address sanitization pentru a îl detecta
  - Ați uitat să includeți tabela ARP statică în arhivă în cazul în care o folosiți

- **Q:** Cum pornesc mai mult de un terminal?

  A: rulați în background.

  ```
  xterm &
  ```

- **Q:** Primesc următoarea eroare:

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

  A: în cazul în care portul este ocupat rulați sudo fuser -k 6653/tcp

- **Q:** Cand rulez checker-ul primesc o eroare legata de lispa fisierelor `router0.pcap` si `router1.pcp`.

  A: Nu a fost instalat tshark: `apt install tshark`

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

  A:

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

- **Q:** Cum pot vedea traficul de pe interfață X?

  A: Folosind `Wireshark`

- **Q:** Cum afisez interfețele unui host?

  A: `ip address show`

- **Q:** Valorile observate într-o captură de rețea nu coincid cu cele emise.

  A: Cel mai probabil este o problemă de endianness. Câmpurile unor pachete în tranzit trebuie să fie în forma "Network Byte Order" (care este de fapt big endian), pe când mașinile voastre sunt foarte probabil little endian. Folosiți următoarele funcții pentru a obține endiannessul corect.

```
uint32_t htonl(uint32_t);     /* host to network long */
uint16_t htons(uint16_t);     /* host to network short */
uint32_t ntohl(uint32_t);     /* network to host long */
uint16_t ntohs(uint16_t);     /* network to host short */
```

- **Q:** Cum inițializez o coadă?

  A:

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

- **Q:** Cum adaug un element în coadă?

  A:

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

- **Q:** Am implementat tot și nu îmi trec testele.

  A:

  - În cazul în care aplicați diferite operații asupra tabelei de rutare la etapă de preprocesare, aveți grijă ca acestea să aibă o complexitate O(n log(n)) deoarece testarea temei începe după două secunde de inițializare.
  - Verificați ca tabela de rutare să nu fi fost modificată

- **Q:** Cum determin adresa IP și MAC-ul unei interfețe a routerului?

  A:

```
/* Întoarce adresa în format zecimal, e.g. "192.168.2.1" */
char *get_interface_ip(int interface);
/* Adresa e întoarsă prin parametrul de ieşire mac; sunt întorşi octetii. */
void get_interface_mac(int interface, uint8_t *mac);
```

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

  A: Probabil routerul primește segmentation fault înainte să dea flush la output; setați `stdout` să fie unbuffered.

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

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

  A:

  ```
  $ make clean
  $ make
  ```

- **Q:** În WireShark văd "Internet Protocol, bogus version".

  A: Completarea antetului IP/ICMP nu este corectă.

- **Q:** Putem folosi C++?

  A: Da.

- **Q:** Ce biblioteci sunt permise?

  A: Este permisă folosirea oricărei funcții din biblioteca standard de C sau C++. Mai mult, puteți folosi orice header standard de Linux, cu precizarea că trebuie în continuare să rezolvați manual cerința. De exemplu, Linux știe să răspundă singur la ICMP; pe router această funcționalitate este dezactivată de framework-ul de testare. Reactivarea ei cu apeluri din C nu reprezintă o soluție validă și nu va fi punctată.

- **Q:** Primesc eroarea "[14]: ioctl SIOCGIFINDEX No such device" când încerc să rulez pe mașina virtuală de Linux.

  A: Routerul caută anumite interfețe după nume; acestea probabil nu există pe sistemul vostru, sunt create de mininet. Asigurați-vă că rulați binarul de router dintr-un terminal de router după pornirea topologiei.

- **Q:** Când încerc să accesez informația din antetul ICMP, în timp ce dau ping de la un host la altul, toate informațiile din header sunt 0.

  A: Spre deosebire de laborator, în temă trebuie implementat și protocolul ARP. În cazul de față, se încearcă extragerea antetului ICMP dintr-un pachet ce conține doar Ethernet + ARP.

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

  A: 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`.

- **Q:** La testul "timeout is unreachable" primesc următoarea eroare:

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

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

- **Q:** Îmi pică ultimele 2 teste.

  A: Ultimele două teste o să pice, în general, dacă realizați LPM ineficient.

- **Q:** Nu îmi apar terminalele când rulez mininet.

  A: probabil nu aveți `xterm` instalat.

  ```
  sudo apt install xterm
  ```

- **Q:** Dacă aveți eroarea de mai jos trebuie să instalați openvswitch-testcontroller și creat fișierul /usr/bin/ovs-controller

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