Tema 1 - Switch

Lectura recomandata

- Tema ar trebui făcută doar după ce ați parcurs **Cursul si laboratorul de VLAN & STP**
- Ethernet Switches
- STP Animation
- The Spanning Tree Protocol (802.1d)
- Virtual LANs

Infrastructura

Pentru a simula o rețea virtuală vom folosi Mininet. Mininet este un simulator de rețele ce folosește în simulare implementări reale de kernel, switch și cod de aplicații. Pentru a nu avea probleme de compatibilitate, vom folosi următoarea masina virtuala de ubuntu.

Tabela de comutare (MAC Table)

La primirea unui cadru (frame) Ethernet, switch-ul aplică un algoritm simplu pentru a lega o adresa MAC de un port. Acesta introduce în tabela de comutare o intrare care leagă, dacă este cazul, portul (interfața) pe care a sosit cadrul cu adresa MAC sursă din antetul (header-ul) Ethernet. Dacă nu există o intrare în tabela de comutare pentru adresa MAC destinație, atunci switch-ul va transmite cadrul pe toate celelalte porturi. Pseudocodul este prezentat în imaginea de mai jos.

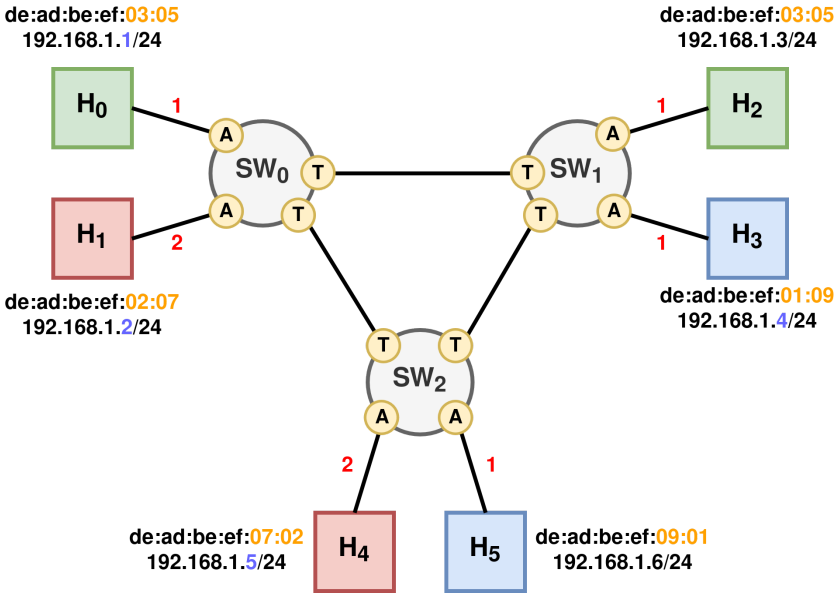
```
# Cadrul F este primit pe portul P
# MAC_Table : Tabela MAC ce face maparea adresa MAC -> port
# Ports : lista tuturor porturilor de pe switch
src = F.SourceAddress
dst = F.DestinationAddress

# Am aflat portul pentru adresa MAC src
MAC_Table[src] = P

if is_unicast(dst):
    if dst in MAC_Table:
        forward_frame(F, MAC_Table[dst])
    else:
        for o in Ports:
          if o != P:
              forward_frame(F, o)
else:
    # trimite cadrul pe toate celelalte porturi
    # Atentie, acest broadcast va fi diferit in cazul in
    # care avem VLAN-uri
    for o in Ports:
        if o != P:
            forward_frame(F, o)
```

În cazul în care switch-ul folosește funcționalitatea de VLAN, **broadcast-ul o să se facă doar către porturile cu aceeași etichetă VLAN** fie către porturile de tip trunk.

VLAN

O funcționalitate importantă a switch-urilor Ethernet este capacitatea de a crea rețele locale virtuale (Virtual Local Area Networks - VLANs). O rețea locală virtuală (VLAN) poate fi definită ca un set de porturi de pe switch care au același identificator VLAN. Calculatoarele din VLAN-uri diferite nu pot comunica direct, chiar dacă sunt conectate la același switch. Avem astfel o izolare la nivelul data link.

În contextul suportului VLAN apare următoarea terminologie: un port de tip **trunk** este un port prin care pot fi transmise cadre din mai multe VLAN-uri și se află între două switch-uri, în timp ce un port de tip **access** este un port care conectează un host la switch.

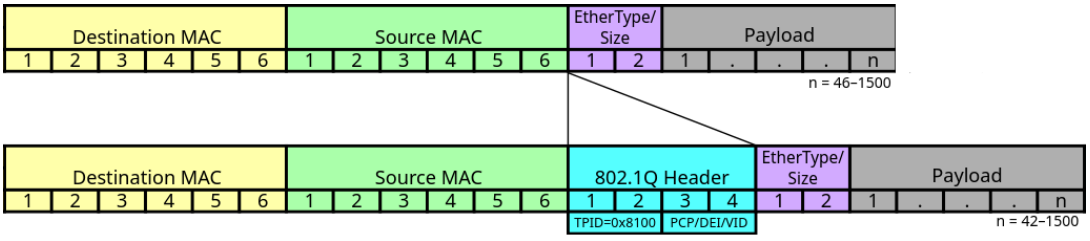


H2 în diagrama are MAC-ul de:ad:be:ef:05:03, nu de:ad:be:ef:03:05.

Topologia implementată în mininet. Valorile numerice scrise în roșu pe link-uri reprezinta VID-ul tip 802.1Q. Partiționarea rețelei pe baza VID-ului extins (unde se ia în calcul adresa MAC) este reprezentată prin culorile sistemelor H0-H5.

> În mod normal, VLAN-ul este configurat la nivelul portului de switch, nu la nivelul unui host, chiar dacă în multe reprezentări grafice stațiile sunt ilustrate ca făcând parte dintr-un VLAN. În acest task, vom încălca acest principiu.

Protocolul IEEE 802.1Q este folosit pentru a introduce sistemul de marcare VLAN (VLAN tagging) în Ethernet, sub forma unei extensii a antetului (header-ului) Ethernet. În imaginea de mai jos putem observa că au fost introduși 4 octeți (bytes). De interes pentru noi este câmpul VID pe 12 biți, ce reprezintă identificatorul VLAN-ului din care cadrul face parte. **De acum înainte, când ne vom referi la un pachet cu header-ul 802.1Q, vom înțelege că este vorba despre un cadru Ethernet plus acești 4 octeți.**



802.1Q tag format

16 bits	3 bits	1 bit	12 bits
TPID	TCI		
	PCP	DEI	VID

Noi vom implementa un sistem custom de VLAN tagging similar cu 802.1Q. Sistemul se va numi **Poli VLAN tagging**. Acesta se va distinge prin două aspecte:

- **TPID**-ul va fi *0x8200*.
- Biții **PCP** și **DEI** vor fi utilizați ca extensie pentru câmpul **VID**. Acesta din urmă va conține în continuare tag-ul de VLAN configurat pe portul switch-ului, dar cei mai semnificativi biți din **TCI** vor lua valoarea sumei nibble-urilor adresei MAC a host-ului asociat access port-ului. Overflow-ul sumei poate fi ignorat.

Switch-ul când **primește** un cadru de pe orice interfața va comuta cadrul mai departe astfel:

- Cu header-ul nostru custom (802.1Q-like) dacă se transmite pe un port de tip trunk (către un switch). De notat că, în funcție de portul sursă, cadrul primit poate fi sau nu să conția deja acest tag. De exemplu, dacă este un cadru primit pe un port trunk, atunci acesta va fi tagged. În schimb, dacă este un cadru primit de pe o interfață de tip access, acesta nu va avea header-ul *EtherType=0x8200,* iar implementarea noastră de switch va trebui să-l adauge.

- Fără header-ul personalizat dacă se transmite pe o interfață de tip access. Comutarea pachetului este condiționată de verificarea câmpului **VID extins**. VID-ul stocat în cadru trebuie sa fie egal cu cel configurat pe portul switch-ului, iar cei mai semnificativi 4 biți din TCI trebuie să coincidă cu suma nibble-urilor adresei MAC a host-ului conectat pe portul respectiv. Pentru a simplifica implementarea, extensia de 4 biți poate fi ignorată dacă frame-ul este multicast sau dacă adresa MAC a host-ului incă nu a fost adaugată in tabela CAM.

Un exemplu de calcul a extensiei de VID pentru host-ul H5 cu MAC-ul **de:ad:be:ef:09:01**:

```
(0xd + 0xe + 0xa + 0xd + 0xb + 0xe + 0xe + 0xf + 0x9 + 0x1) & 0xf = 114 & 0xf = 0x2
```

Deoarece VLAN ID-ul clasic configurat pe switch este **2**, VID-ul extins va fi **0x2002**.

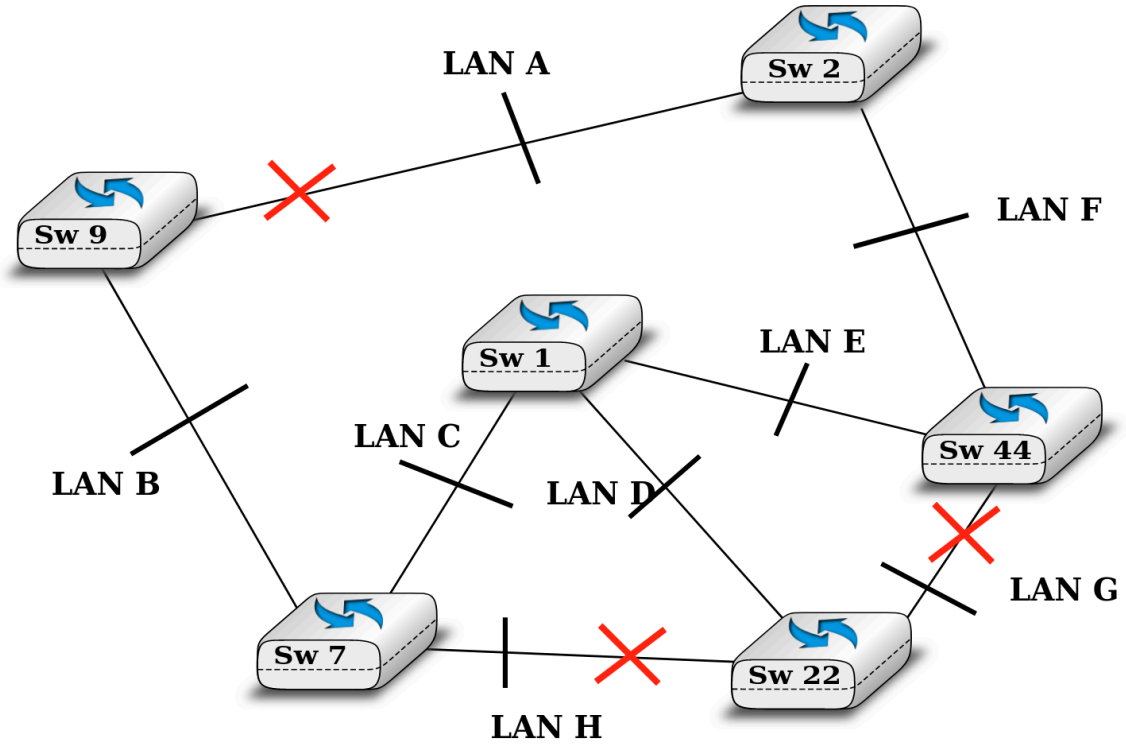
Atentie, nu uitați ca dimensiunea cadrului crește cu 4 bytes.

Legăturile dintre switch-uri vor fi configurate în modul trunk, ceea ce permite trecerea tuturor VLAN-urilor. În cadrul acestei teme nu vom lua în considerare VLAN-ul nativ.

VLAN-urile atașate porturilor și porturile în mod trunk vor fi configurate în switch prin intermediul unui fișier de configurare, descris în secțiunea API.

The Spanning Tree Protocol(802.1d)

Protocolul Spanning Tree (Spanning Tree Protocol - STP) este un protocol distribuit utilizat de switch-uri pentru a reduce topologia rețelei la un **arbore de acoperire**, astfel încât să nu existe bucle în topologie. În figura de mai jos, rețeaua conține multiple bucle care trebuie eliminate pentru a permite switch-urilor Ethernet să transmită cadre fără riscul inundării rețelei cu trafic redundant, care ar conduce la colapsul rețelei. După mai multe runde (secunde pentru noi), toți participanții (switch-urile) vor converge către un lider (root bridge). STP folosește un algoritm de tip leader election.



Pentru a înțelege mai bine acest protocol, înainte de a începe tema vom simula în Packet Tracer o topologie cu trei switch-uri. Vom studia cadrele BPDU (Bridge Protocol Data Units), evidențiate cu roz în simulator, care sunt transmise periodic de către switch-uri.

În cadrul temei vom implementa o varianta de STP numita **Poli STP sau PSTP**. Astfel, vom avea un singur proces STP pentru toate VLAN-urile, iar scopul este de a bloca link-urile care conduc la formarea de bucle. În cele ce urmează vom descrie structura pachetelor ce vor fi folosite de implementarea noastra de protocol. Vom avea doua tipuri de cadre trimise în cadrul protocolului:

- **Hello Protocol Data Units (HPDU)**. Este un pachet trimis la fiecare secundă de fiecare switch. Îl vom folosi ca formă de verificare a faptului că un link este activ. În această implementare doar vom trimite aceste cadre, nu ne interesează funcționalitatea de marcare a unui link ca fiind nefuncțional. **Acesta este trimis pe toate porturile, si cele trunk, si cele catre hosts. Pachetul HPDU va fi un simplu cadru de ethernet (type 0x0800) care contine ca date un byte cu valoarea 255. Vom folosi MAC-ul destinatie** `ff:ff:ff:ff:ff:ff`

- **Poli Protocol Data Units (PPDU)**. Vor conține trei informații importante: identificatorul switch-ului rădăcină (root bridge ID - 64 biți), identificatorul switch-ului expeditor (sender bridge ID - 64 biți) și costul drumului până la rădăcină (root path cost - 64 biți). Switch-ul rădăcină (root bridge) este switch-ul cu identificatorul cel mai mic.

Structura cadrelor PPDU. Cadrele PPDU folosesc encapsulare*a 802.2 Logical Link Control header.* Mai jos putem vedea structura unui cadru PPDU.

```
Size (bytes) 6        6       2           3           4           31
           DST_MAC|SRC_MAC|LLC_LENGTH|LLC_HEADER|PPDU_HEADER|PPDU_CONFIG
```

LLC_LENGTH este dimensiunea totala a cadrului, inclusiv dimensiunea PPDU.
LLC_HEADER are urmatoarea structura:

```
Size            1                                   1                           1
  DSAP (Destination Service Access Point)|SSAP (Source Service Access Point)|Control
```

Pentru a identifica protocolul PPDU, **DSAP** si **SSAP** vor fi 0x42. Pentru **control** vom pune 0x03.

Structura Poli PPDU_HEADER este următoarea:

```
Size (bytes)
     2             1                1               4
Protocol ID|Protocol Version|PPDU Type|Sequence Number|
```

Detalii de implementare pentru câmpuri:

- **Protocol ID** - Vom folosi 0x0002 pentru PPDU Protocol
- **Protocol Version** - Vom folosi 0
- **PPDU Type** (1 byte): Vom folosi 0x80
- **Sequence Number** (4 bytes): Numărul de secvență. Fiecare cadru PPDU va avea acest număr incrementat. Vom porni numărul de secvență de la 0. Valoarea va fi modulo 100. Astfel, primul cadru trimis la prima cuantă de timp va avea numărul de secvență 0, al doilea va avea numărul de secvență 1, al 100-lea va avea numărul de secvență 99, iar al 101-lea va avea numărul de secvență 0.

Structura *PPDU_CONFIG* este următoarea:

```
Size (bytes)
     1          8              4            8       2       2          2       2          2
  flags|root bridge ID|root path cost|bridge id|port id|message age|max age|hello time|forward delay|
```

- Vom presupune fiecare link ca fiind de 100 Mbps, costul pe link va fi conform celor descris în curs pentru STP (pentru switch-uri cu legături normale, nu rapide). Este folosit in calculul root path cost.
- **bridge id** va fi calculat conform descrierii din curs. Prioritatea este disponibilă în config.
- **Max age** va fi setat pe maximul posibil din specificatia de 802.1D-2004
- **Hello Time** va fi setat pe valoarea recomandata din 802.1D-2004
- **Forward Delay** va fi setat pe valoarea minima din 802.1D-2004
- **port id** este construit conform standardului 802.1D. Vom folosi varianta implicită definită în standard.
- **flags** il setam pe 0
- **message age** este setat pe 0

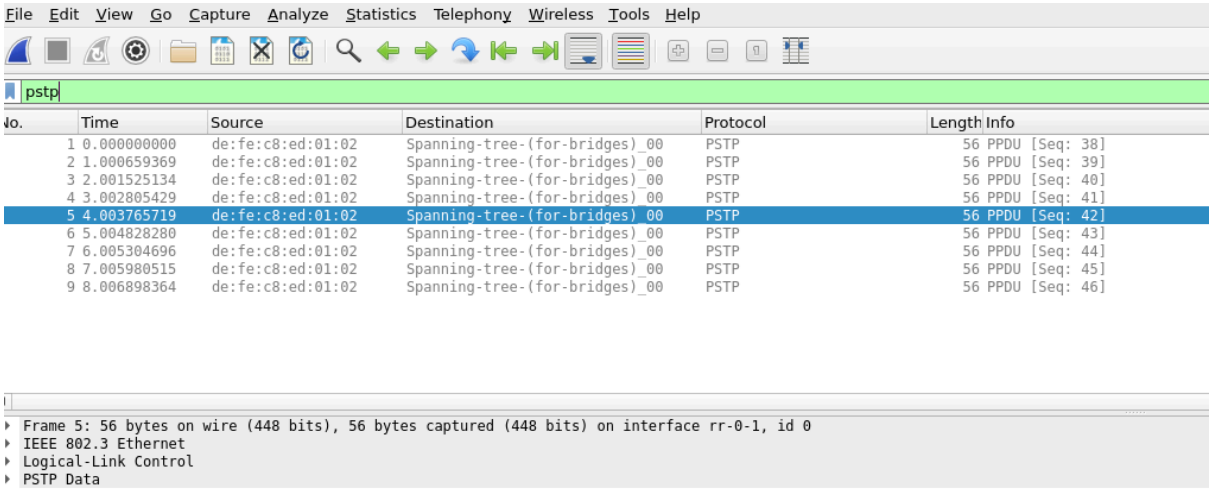
Nu uitati, in networking folosim **Big Endian**.

Pentru a fi punctat acest subpunct, cadrele trebuie să respecte indicațiile de mai sus. Pentru fiecare valoare din cadru care nu este exact specificată în enunț (e.g., pentru cele unde se specifică că valoarea este conform 802.1D-2004), va trebui să argumentați decizia și să includeți o referință. Fie un link, fie un PDF în arhivă în directorul sources. Sursa trebuie să includă locația exactă în document, fie prin pagină, fie prin numele secțiunii etc. Checker-ul verifică doar funcționalitatea, nu și respectarea cerințelor de fond.

Pentru a vizualiza pachetelor trimite în cadrul protocolului nostru custom, vom folosi un dissector de Wireshark, pe care îl găsiți în scheletul temei. Pentru a-l încărca, va trebui să îl puneți în `/usr/lib/x86_64-linux-gnu/wireshark/plugins` pe VM (găsiți path-ul pe un setup custom în *Wireshark → About → Folders → Global LUA Plugins*). Va trebui să dați restart la Wireshark. Găsiți dissector-ul în scheletul temei.

Cadrele PPDU sunt identificate prin adresa multicast MAC destinatie descrisa in standardul 802.1D.

Pachetele trimise trebuie să fie conforme și să fie identificate corect de către dissectorul de PSTP din Wireshark.



Scopul temei este de a implementa protocolul STP peste pachetele descrise mai sus. Pentru simplitate, vom presupune că switch-urile nu se pot strica și vom considera doar două stări pe port: Blocking și Forwarding pentru ca pachetele sunt trimise

Pentru a primi punctaj pe acest subpunct, va trebui să includeți în arhivă un fișier **state_machine.pptx** care să includă reprezentarea state machine-ului implementat pentru PSTP.

Prin stări, ne referim la stările protocolului. Ce se dorește este o reprezentare vizuală a modului în care ați implementat protocolul din temă. Nu avem constrângeri de cum ar trebui să arate, ideea este ca, dacă se uită cineva la ea, să poată înțelege rapid cum ați făcut implementarea. În STP întâlnim aceste stări de protocol date de: stările porturilor (e.g. FORWARDING) și rolurile lor (e.g. DESIGNATED, BLOCKING). Avem starea inițială, în care fiecare switch se consideră root bridge, după care este primit un pachet BPDU, și în funcție de ce conține pachetul acesta (e.g. un root bridge id mai mic, mai mare, egal), o să ne mutăm în altă stare de protocol (unde porturile o să aibă alte roluri și stări).

API

Pentru rezolvarea temei vă punem la dispoziție un schelet de cod care implementează unele funcționalități esențiale pentru rezolvarea cerințelor, precum și unele funcții ajutătoare ale căror utilizare este opțională. În **wrappers.py** găsiți un set de funcții Python peste cele din C care permit interacțiunea cu nivelul data link.

wrappers.py pune la dispoziție o serie de funcții în Python care, de fapt, apelează în spate o serie de funcții C din biblioteca **dlink.so**. Biblioteca dlink.so este compilată cu GCC și se găsește în fișierul aici. În spate, biblioteca C folosește Linux sockets.

```
# Initializeaza switch-ul. Primeste ca argument un string cu interfetele.
init(switch_interfaces)

# Functie blocanta, primeste un cadru ethernet.
port, eth_frame, length = recv_from_any_link()

# Trimite un cadru ethernet pe o interfata. eth_frame este de tip bytes string.
send_to_link(interface, length, eth_frame)

# Returneaza adresa MAC a switch-ului in bytes string
get_switch_mac()

# Returneaza numele unei interfete.
get_interface_name(interface)
```

De asemenea, vom avea și fișiere de configurare a switch-urilor, switchX.cfg. Acestea au următorul format.

```
SWITCH PRIORITY
INTERFACE_NAME [VLAN]/T (T de la Trunk)
...
```

De exemplu:

```
1931
r-0 4
r-1 3
rr-0-1 T
rr-0-2 T
```

> Nu exista API pentru citirea configurației.

Cerinte

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

Pentru rezolvarea temei, trebuie să implementați funcționalitatea unui switch. Va recomandăm să folosiți cel puțin **ping** pentru a testa implementarea și **Wireshark** pentru depanare și analiză corectitudinii. Punctajul este împărțit în mai multe componente, după cum urmează:

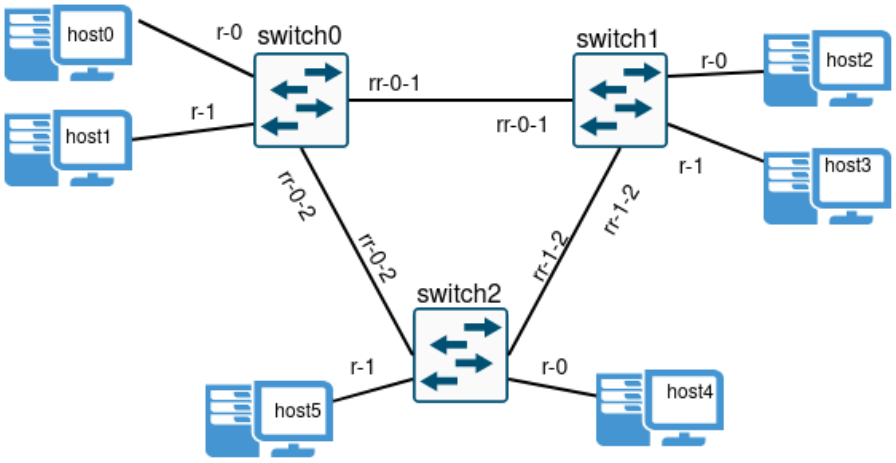
- **Procesul de comutare (30p)**. Va trebui să implementați funcționalitatea descrisă în secțiunea **Procesul de Comutare**. Pentru acest exercițiu nu este nevoie să implementați funcționalitatea referitoare la VLAN sau STP. Pentru a evita buclele, vom porni doar switch-urile 0 și 1.
- **VLAN (30p)**. Vom implementa funcționalitatea de Virtual Local Area Networks (VLANs) descrisă în enunț. Fișierele de configurație ale switch-urilor se găsesc în directorul **configs**. Pentru a evita buclele, vom porni doar switch-urile 0 si 1.
- **STP (40p)**. În acest exercițiu vom introduce și switch-ul 2, care va determina apariția unei bucle. Se cere implementarea protocolului STP folosind simplificarile și pachetele descris în enunț.

Pentru a fi punctată o temă, arhiva trebuie să conțină câte un screenshot relevant pentru fiecare exercițiu în care să apară numele vostru (e.g. cu un echo), una sau mai multe ferestre de Wireshark și unul sau mai multe terminale care să dovedească rularea manuală a funcționalității relevante. Fiecare screenshot este însoțit și de o explicație în README.md. De asemenea, în README.md trebuie să documentați deciziile de design (e.g., cum ați stocat stările în STP), dificultățile întâmpinate și cum le-ați rezolvat, optimizările aduse dacă este cazul, și o secțiune de auto-evaluare a implementării.

Screenshot-urile vor fi incluse în root-ul arhivei cu numele **ex1.jpg**, **ex2.jpg** și **ex3.jpg**

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, câte un terminal pentru fiecare switch; 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 ping. Mai mult, din terminal putem rula Wireshark sau tcpdump pentru a face inspecția de pachete. Pentru a compila codul vom folosi **make**.

Pentru a porni switch-urle manual folosim urmatoarea comanda:

```
make run_switch SWITCH_ID=X   # din terminalul unui switch,
                              # unde X este 0, 1 sau 2 si reprezinta ID-ul switch-ului.
```

Ca sa nu scrieți manual ip-ul unui host, puteti folosii host0, host1, host2 si host3 in loc de IP. (e.g. ping host1)

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 hosts (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 switch, 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)

Testarea este incrementala. De asemenea, toate testele de la un subpunct trebuie sa treaca pentru a primi punctajul aferent.

Testarea este incrementala. De asemenea, toate testele de la un subpunct trebuie sa treaca pentru a primi punctajul aferent.

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).

Punctajul per cerința de functionalitate se acorda doar dacă trec toate testele relevante.

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. Arhiva (zip) trebuia să includă pe langa surse și Makefile, un fișier **README.md și toate celelalte fișiere specificate în enunț.** În fișierul README.md va trebui să specificați pe prima linie cerințele rezolvate în format **1 2 3** (toate), **1 2** sau **1**. De exemplu, pentru o tema trimisă cu doar primele două subiecte rezolvate, vom pune astfel:

```
1 1 2
2
3 Decizile de design au fost...
```

O rulare completă durează cam 9 minute. Vă rugăm să trimiteți tema pe Moodle doar după ce ați verificat că aceasta rulează bine pe local.

Subiectele rezolvate prin soluții hard coded pot aduce depunctări intre 0 si 100p.

Pentru a nu degrada performanța implementării voastre, aveți grijă să nu aveți print-uri lăsate prin cod. Acestea sunt utile doar pentru debug.

FAQ

Q: Pe Moodle primesc punctaj 0, pe local 100.

A: Cele mai intalnite probleme sunt faptul ca arhiva trimită nu contine in root-ul ei fisierul switch.py (acesta e intr-un subdirector) sau de la versiune de python rulata pe checker

Q: Cum știu ce fișier de config sa citesc?

A: Primul argument pe care îl primește switch-ul la rulare este identificatorul (switch_id = sys.argv[1])

A: Nu se aplica depunctări pentru coding style

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

A:

Problema poate apărea atunci când implementarea voastră are o performanță scăzută. Pentru a rezolva problema, asigurați-vă că aveți o implementare bună din punct de vedere al performanței.

- funcțiile de debug care vă scad performanța codului in productie (e.g. print)
- undefined behaviour care este vizibil doar pe checker. În acest caz ar trebui să folosiți valgrind și address sanitization pentru a îl detecta

Q: Cum pornesc mai mult de un terminal?

A: Rulați în background.

xterm &

Q: Cum pornesc wireshark pe un terminal, dar sa fac si alte actiuni in acelasi terminal?

A: Rulati in background wireshark.

wireshark &

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 legată de lipsa fișierelor switch0.pcap și switch1.pcap.

A: Nu a fost instalat tshark: apt install tshark

Q: Nu merge `ping hostX`

A: Posibil sa fie o problema cu intrările din /etc/hosts. Verificati ca acolo sa aveti intrările bune.

Q: Pot face tema si in C/C++?

A: Da, funcțiile relevante se găsesc în lib.h. De asemenea, vă recomandăm să aruncați o privire peste wrappers.py. În submisia voastră va trebui să includeți și un Makefile cu regulile voastre de run_switch și build. Regulile trebuie să primească aceleași argumente ca și în cazul regulilor de Python.

Q: Cand testez manual cu ping, imi merge, dar pe checker pica.

A: Atentie la conceptul de stare. De exemplu, testul `ICMP_0_3_NOT_ARRIVES_2` este rulat in contextul in care si celelalte teste au fost rulate, astfel au fost trimise mai multe ping-uri pana la rularea acestui test.