Tema 1 - Implementare Switch (0.8p)

Lectura recomandata

- Tema ar trebui facuta doar dupa ce ati 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 rula pe Linux aceasta tema. Va recomandam aceasta masina virtuala de ubuntu.

sudo apt update

sudo apt **install** mininet openvswitch-testcontroller tshark python3-click python3-scapy xterm python3-pip

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

Când o să rulăm simularea, e posibil să întâlniți următoarea eroare: **Exception: Please shut down the controller which is running on port 6653:**. Pentru a rezolva problema, va trebui să rulați **pkill ovs-test**.

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.

 # 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)
```

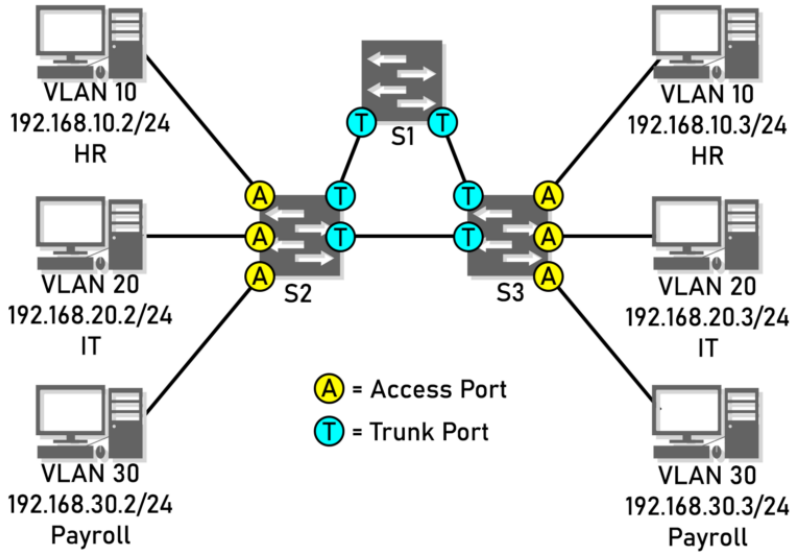
În cazul în care switch-ul folosește funcționalitatea de VLAN, **broadcast-ul o să se facă doar către porturile cu aceeași etichetă VLAN** fie către porturile de tip trunk.

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 acelasi 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.

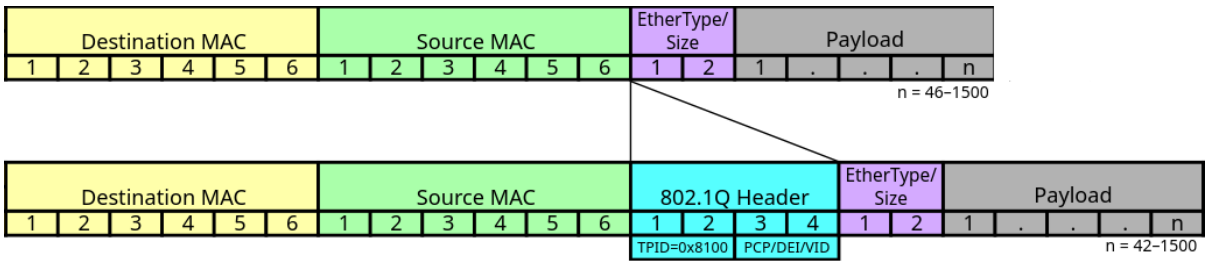
În contextul suportului VLAN apare următoarea terminologie: un port de tip **trunk** este un port prin care pot fi transmise cadre din mai multe VLAN-uri și se

află între două switch-uri, în timp ce un port de tip **access** este un port care conectează un host la switch.



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.

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 byți.



802.1Q tag format

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

Switch-ul cand **primeste** un cadru de pe orice interfata va comuta cadrul mai departe astfel:

- cu header-ul 802.1Q 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 în format 802.1Q. De exemplu, dacă este un cadru primit pe un port trunk, atunci acesta va fi în format 802.1Q. În schimb, dacă este un cadru primit de pe o interfață de tip access, acesta nu va avea header 802.1Q, iar implementarea noastră de switch va trebui să îl adauge.
- fără header-ul 802.1Q dacă se transmite pe o interfață de tip access și VLAN ID-ul este egal cu cel al interfeței de pe care a venit (către un host direct conectat la switch, parte a aceluiași VLAN)

Nu vom comuta cadrele atunci când VLAN ID port destinație != VLAN ID cadru.

Stațiile din simulare rulează Linux, iar stiva de networking din Linux face VLAN filtering, pentru a nu pierde eticheta VLAN (VLAN tag), vom folosi pentru TPID valoarea **0x8200** în loc de **0x8100**. Practic vom avea o implementare custom de 802.1q. Câmpurile PCP și DEI vor fi setate la valoarea 0.

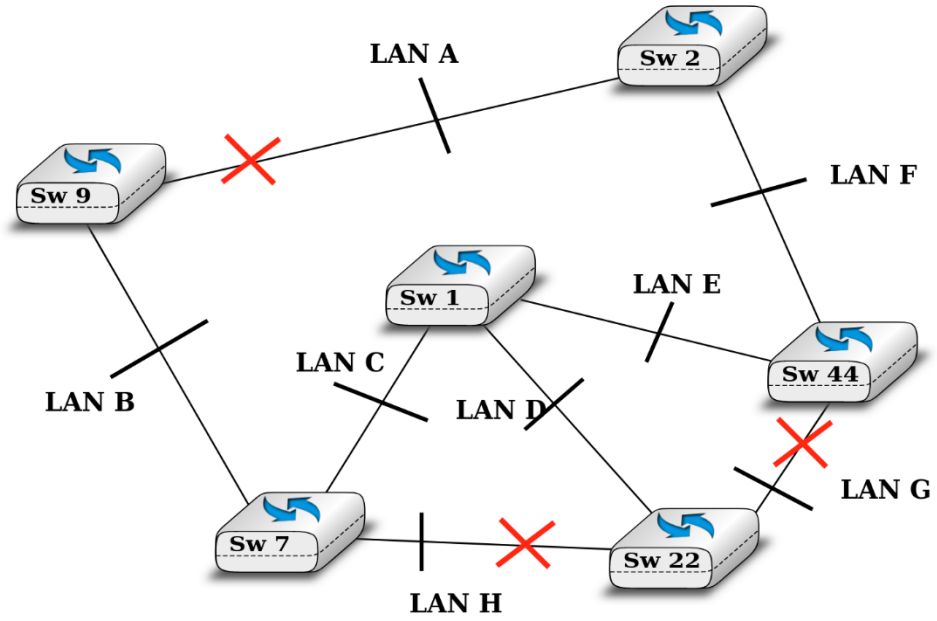
Atentie, nu uitati ca dimensiunea cadrului creste cu 4 bytes

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. Dupa mai multe runde (secunde pentru noi), toti participantii (switch-urile) vor converge catre un lider (root bridge). Acesta este un algoritm de tip leader election.



Pentru a înțelege mai bine acest protocol, 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.

Implementarea noastră va fi una simplificată: vom avea un singur proces STP pentru toate VLAN-urile, iar scopul este de a bloca link-urile care conduc la formarea de bucle. Cadrele transmise în cadrul protocolului se numesc Bridge Protocol Data Units (BPDU) și conțin 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.

Algoritmul simplificat este descris în pseudocodul de mai jos. În implementarea noastră, un port de switch poate avea două stări: **Blocking** și **Listening**. În starea Listening, portul funcționează normal pentru comutarea cadrelor. La pornire, fiecare switch se consideră Root Bridge și își setează toate porturile în starea Listening, deoarece acestea sunt considerate porturi **designated** - porturi care au costul cel mai mic catre Root Bridge.

În implementarea algoritmului ne interesează doar porturile de tip trunk, deoarece doar prin acestea se pot forma bucle. Astfel, orice referință la porturi în pseudocod se referă la legăturile trunk. Legăturile către hosts (porturile de tip access) rămân în starea listening pe toată durata funcționării switch-ului.

Initialize:

 # Punem pe block-ing port-urile trunk pentru ca

 # doar de acolo pot aparea bucle. Port-urile catre

 # statii sunt pe deschise (e.g. designated)

```
for each trunk port on the switch:
    Set port state to BLOCKING

# In mod normal bridge ID este format si din switch.mac_address
# pentru simplitate vom folosi doar priority value ce se gaseste in
# configuratie
own_bridge_ID =  switch.priority_value
root_bridge_ID = own_bridge_ID
root_path_cost = 0

# daca portul devine root bridge setam porturile ca designated
if own_bridge_ID == root_bridge_ID:
    For each port on the bridge:
        Set port state to DESIGNATED_PORT
```

La fiecare secunda, daca suntem root bridge, vom trimite un pachet BPDU.

```
Every 1 second:
    if switch is root:
        Send BPDU on all trunk ports with:
            root_bridge_ID = own_bridge_ID
            sender_bridge_ID = own_bridge_ID
            sender_path_cost = 0
```

În cazul în care am primit un pachet de tip BPDU, dacă acesta are un BID (Bridge ID) mai mic decât al nostru, atunci switch-ul de la care am primit acest pachet devine root bridge pentru noi. Mai mult, vom retransmite propriul nostru BPDU actualizat pe toate celelalte porturi.

```
On receiving a BPDU:
    if BPDU.root_bridge_ID < root_bridge_ID:
        root_bridge_ID = BPDU.root_bridge_ID
        # Vom adauga 10 la cost pentru ca toate link-urile sunt de 100 Mbps
        root_path_cost = BPDU.sender_path_cost + 10
```

root_port = port where BPDU was received

if we were the Root Bridge:

set all interfaces not to hosts to blocking except the root port

if root_port state is BLOCKING:

Set root_port state to LISTENING

Update and forward this BPDU to **all** other trunk ports with:

sender_bridge_ID = own_bridge_ID

sender_path_cost = root_path_cost

Else if BPDU.root_bridge_ID == root_bridge_ID:

If port == root_port and BPDU.sender_path_cost + 10 < root_path_cost:

root_path_cost = BPDU.sender_path_cost + 10

Else If port != Root_Port:

 # Verifica daca portul ar trebui trecut pe designated.

 # Designated inseamna ca drumul catre root este prin

 # acest switch. Daca am bloca acest drum, celelalte

 # switch-uri nu ar mai putea comunica cu root bridge.

 # Nota: in mod normal ar trebui sa stocam ultimul BPDU

 # de pe fiecare port ca sa calculam designated port.

if BPDU.sender_path_cost > root_path_cost:

If port is not the Designated Port for this segment:

Set port as the Designated Port and **set** state to LISTENING

Else if BPDU.sender_bridge_ID == own_bridge_ID:

Set port state to BLOCKING

Else:

Discard BPDU

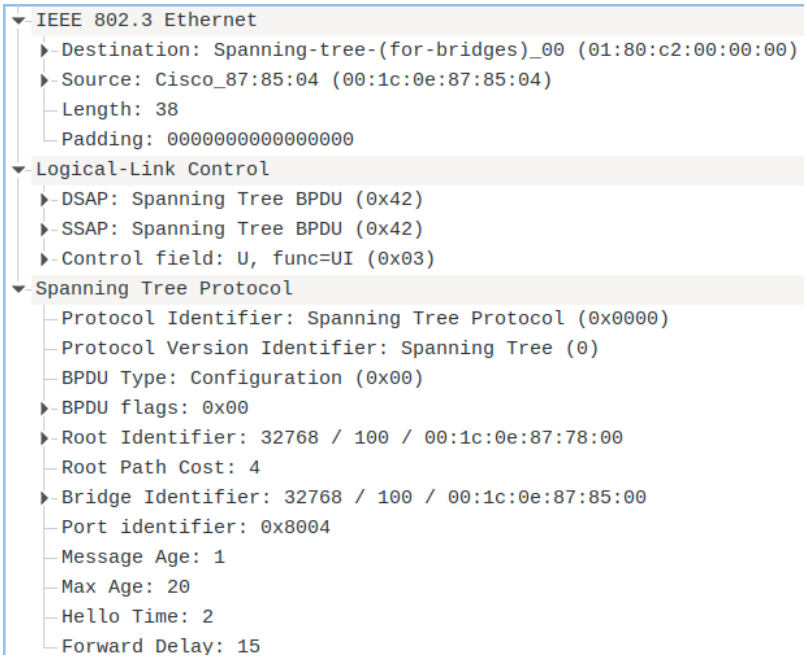
if own_bridge_ID == root_bridge_ID:

For each port on the bridge:

Set port as DESIGNATED_PORT

Pentru simplitate, vom presupune ca switch-urile nu se pot strica.

Structura cadrelor BPDU. Cadrele BPDU folosesc encapsularea 802.2 Logical Link Control header. Figura de mai jos prezinta un astfel de cadru:



Următorul articol prezintă pe scurt structura lor. Aici gasiti o captura de trafic STP.

Size (bytes) 6 6 2 3 4 31

DST_MAC|SRC_MAC|LLC_LENGTH|LLC_HEADER|BPDU_HEADER|BPDU_CONFIG

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

Size 1 1 1

DSAP (Destination Service Access Point)|SSAP (Source Service Access Point)|Control

Pentru a identifica protocolul STP, DSAP si SSAP vor fi 0x42. Pentru control vom pune 0x03. Structura BPDU Config este urmatoare:

```
uint8_t  flags;
uint8_t  root_bridge_id[8];
uint32_t root_path_cost;
uint8_t  bridge_id[8];
uint16_t port_id;
uint16_t message_age;
uint16_t max_age;
uint16_t hello_time;
uint16_t forward_delay;
```

Cum toate switch-urile implementeaza protocolul scris de noi, puteti folosi fie aceasta structura, fie va definiti propria voastra reprezentare.

Cadrele BPDU sunt identificate prin adresa multicast MAC destinatie, 01:80:C2:00:00:00.

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)

Pentru cei ce vor sa lucreze direct din C/C++ in README.md gasiti si API-ul echivalent.

De asemenea, vom avea si fisiere de configurare a switch-urilor, **switchX.cfg**. Acestea au urmatorul 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 configuratiei.

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). 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)**. In acest exercițiu vom introduce si switch-ul 2, care va determina apariția unei bucle. Se cere implementarea protocolului STP simplificat, descris in enunț, pentru a evita trimiterea pachetelor la infinit.

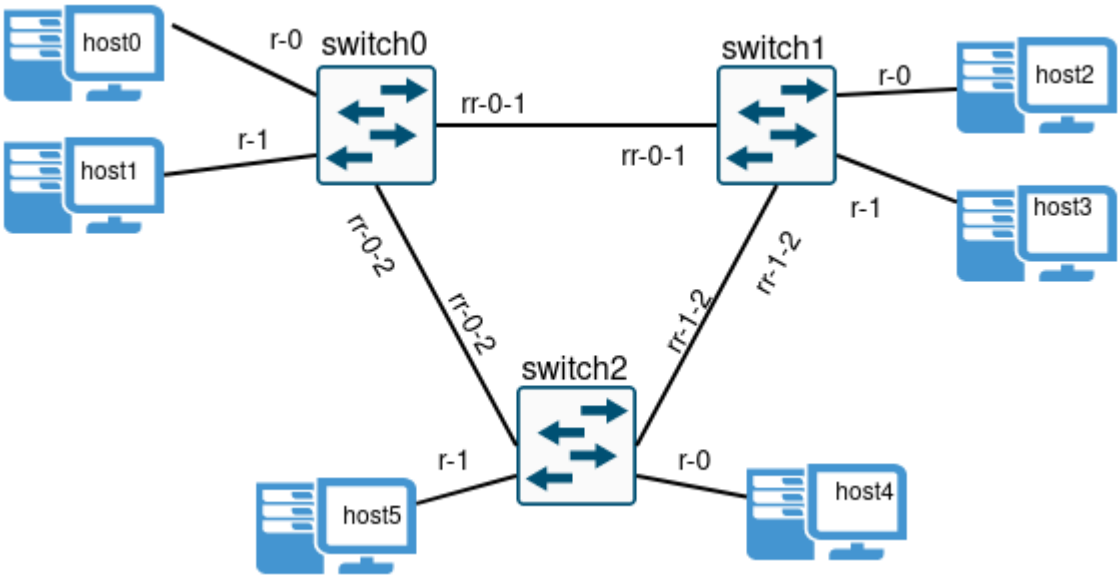
Pentru a fi punctata o tema, in README trebuie prezentata soluția voastră pe scurt.

Este foarte important să folosiți Wireshark pentru verificarea corectitudinii implementării.

Puteți scrie implementarea în Python sau C/C++. Daca lucrati in C/C++ va trebui sa schimbati regula de rulare a switch-ului din **Makefile**.

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**.

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 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 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 https://osqa-ask.wireshark.org/questions/30636/traces-from-multiple-interface/aici pentru a obține o vizualizare mai bună)

Testarea este incrementala. De asemenea, toate testele de la un subpunct trebuie sa treaca pentru a prrimi 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 cerinta se acorda doar daca 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 un fisier numit **README** în care să detaliați implementarea, și fișierul python switch.py în care ați făcut rezolvarea. În fișierul README va trebui să specificați pe prima linie cerințele rezolvate în format **1 2 3** (toate), **1 2** sau **1**.

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.

Temele rezolvate in C/C++ trebuie sa includa si un Makefile cu regulile de build si run_switch aferente.

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: Ce versiune de python ruleaza checker-ul de pe Moodle?

A: Python 3.8.10 (Atentie la ce functionalitati folositi)

Q: Pe Moodle primesc punctaj 0, pe local 100. **A:** Cele mai intalnite probleme sunt faptul ca arhiva trimita 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 legata de lispa fisierelor switch0.pcap si switch1.pcap.

A: Nu a fost instalat tshark: apt install tshark

Q: Nu merge `ping hostX`

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

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

A: Da, functiile relevante se gasesc in lib.h. De asemenea, va recomandam sa aruncati un ochi peste wrappers.py. In submisia voastra va trebui sa includeti si un Makefile cu regula voastra de run_switch si build. Regulile trebuie sa primeasca aceleasi argumente ca si in 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.

In fisierul Dockerfile gasiti imaginea docker folosita pe vmchecker.

docker build -f Dockerfile -t tema1 .

docker run --privileged -v .:/data/ -it tema1 /bin/**bash**

 # Din bash-ul deschis in container rulati vom rula asa:

cd /data/

ulimit -c 1024

/bin/**bash** -c "ulimit -c 1024 ; python3 checker/topo.py tests"

 # inspectati switch_0_err dupa ce a picat testul cu probleme