Tema 1 - Implementare Switch (0.8p)

- **Data publicării:** 04.11.2023
- **Deadline HARD:** 20.11.2023, 23:50

Lectura recomandata

- Ethernet Switches [https://beta.computer-networking.info/syllabus/default/protocols/lan.html?highlight=spanning%20tree%20protocol#ethernet-switches]
- The Spanning Tree Protocol (802.1d) [https://beta.computer-networking.info/syllabus/default/protocols/lan.html?highlight=spanning%20tree%20protocol#the-spanning-tree-protocol-802-1d]
- Virtual LANs [https://beta.computer-networking.info/syllabus/default/protocols/lan.html?highlight=spanning%20tree%20protocol#virtual-lans]

Infrastructura

Pentru a simula o rețea virtuală vom folosi Mininet [http://mininet.org/]. Mininet este un simulator de rețele ce folosește în simulare implementări reale de kernel, switch și cod de aplicații. Mininet poate fi folosit atat pe Linux cat si WSL2.

```
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

La primirea unui cadru (frame) Ethernet, switch-ul aplică un algoritm simplu descris de pseudocodul de mai jos. Introduce în tabela de comutare o intrare în care leagă, dacă e cazul, portul (interfața) de pe care a venit cadrul de adresa MAC sursă din header-ul Ethernet. Dacă nu are o intrare în tabela de comutare pentru MAC destinație, atunci va trimite 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
    for o in Ports:
        if o != P:
```

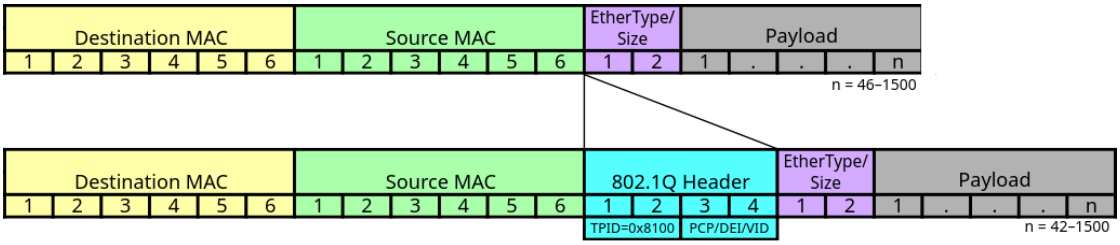
```
            forward_frame(F, o)
```

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

Un avantaj important al switch-urilor Ethernet este capacitatea de a crea Virtual Local Area Networks (VLANs). O rețea LAN virtuală (VLAN) poate fi definită ca un set de porturi atașate la unul sau mai multor VLAN-uri. Un switch poate suporta mai multe VLAN-uri. Când un switch primește un cadru cu o destinație necunoscută sau broadcast, îl transmite peste toate porturile care aparțin aceluiasi VLAN (inclusiv trunk-uri), dar nu peste porturile care aparțin altor VLAN-uri.

IEEE 802.1Q este folosit pentru a introduce sistemul de VLAN tagging in Ethernet. In imaginea de mai jos putem observa ca au fost introdusi 4 bytes. De interes pentru noi este campul VID pe 12 biti ce reprezinta identificatorul VLAN-ului din care cadrul face parte.



802.1Q tag format

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

Switch-ul de fiecare data cand **primeste** un cadru de pe o interfata de tip **access**(catre host) va trimite cadrul urmator:

- cu header-ul 802.1q daca se duce pe o interfata de tip trunk
- fara header-ul 802.1q daca se duce pe o interfata de tip access si vlan id egal cu cel al interfetei de pe care a venit

Switch-ul de fiecare data cand **primeste** un cadru de pe o interfata de tip **trunk** va **scoate tag-ul** si va trimite cadrul urmator:

- cu header-ul de 802.1q (inclusiv tag) daca se duce pe o interfata de tip trunk
- fara header-ul de 802.1q daca se duce pe o interfata de tip access si vlan id egal cu cel al cadrului primit

In implementarea noastra, switch-ul **nu va pastra tag-urile** de VLAN atunci cand comuta un cadru catre un host.

Linux face VLAN filtering, asa ca pentru a nu pierde tag-ul de VLAN, vom folosi pentru TPID valoarea **0x8200** in loc de **0x8100**. PCP si DEI ii vom seta pe 0.

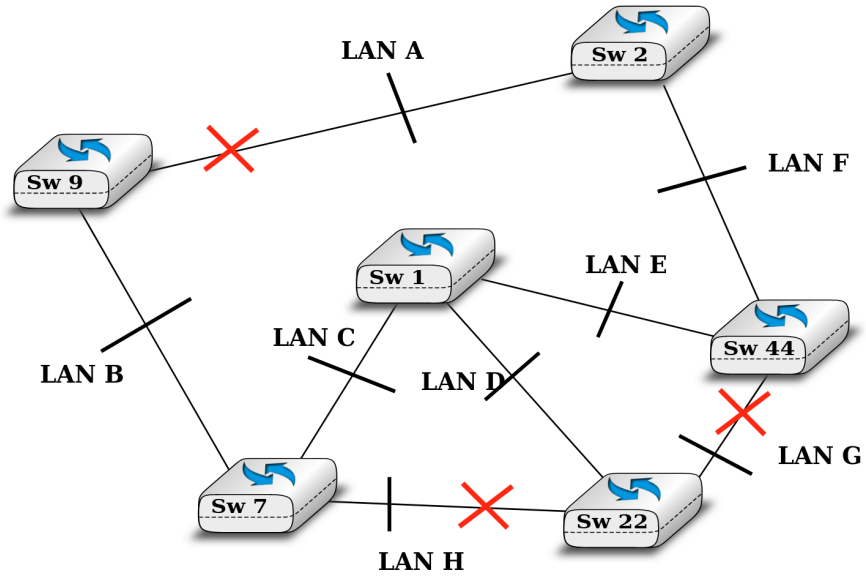
Atentie, nu uitati ca dimensiunea cadrului creste cu 4 bytes

Legaturile dintre switch-uri vor fi in modul **trunk**, adica vor permite trecerea tuturor VLAN-urilor. La aceasta tema nu ne intereseaza VLAN-ul nativ.

VLAN-urile atasate porturilor si porturile in mod trunk vor fi primite de catre switch printr-un fisier de configurare descris in sectiunea API.

The Spanning Tree Protocol(802.1d)

The 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 cicluri în topologie. În figura de mai jos, rețeaua conține mai multe cicluri care trebuie rupte pentru a permite switch-urilor Ethernet să schimbe cadre fără a întâmpina inundarea rețelei din cauza buclelor.



Implementarea noastră va fi una simplificată, vom avea un singur proces de STP pentru toate VLAN-urile și scopul este de a bloca legătura de date care conduce la bucle. Pachetele transmise în cadrul protocolului se numesc Bridge Protocol Data Unit (BPDU) și conțin trei lucruri importante: root bridge ID (64 biți), sender bridge ID (64 biți) și root path cost (64 biți). Root bridge-ul reprezintă switch-ul cu ID-ul cel mai mic.

Algoritmul simplificat este descris în pseudocodul de mai jos. În implementarea noastră vom avea două stări pe un port de pe switch: **Blocking** și **Listening** (port deschis). În modul Listening folosim portul în mod normal. Vom porni fiecare port pe modul blocking. Inițial, fiecare switch va considera că el este Root Bridge-ul, asta înseamnă că o să treacă toate porturile în modul listen pentru că acestea sunt designated. Urmează etapa de stabilire a consensului între toate switch-urile pe cine este root, cel ce are ID-ul minim. În această etapă fiecare switch ce se consideră root bridge va trimite regulat un cadru BPDU. La primirea unui astfel de cadru, fiecare switch reacționează în funcție de ID-ul root bridge-ului de care acesta știe. Dacă BPDU-ul primit are un ID mai mic decât al root brdidge-ului cunoscut, atunci acesta își actualizează starea porturilor și transmite mai departe BPDU-ul. Doar porturile Root (Rădăcină) și Designated (Desemnate) sunt folosite pentru a transmite cadre de date, acestea fiind pe listening.

În implementarea algoritmului ne interesează doar porturile de tip Trunk, pentru că de acolo pot apărea bucle. Astfel, orice referință la porturi în pseudocod se referă la legăturile Trunk. Link-urile către stații sunt pe listening pe tot parcursul rulării 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 switch-ul root, vom trimite un pachet BPDU.

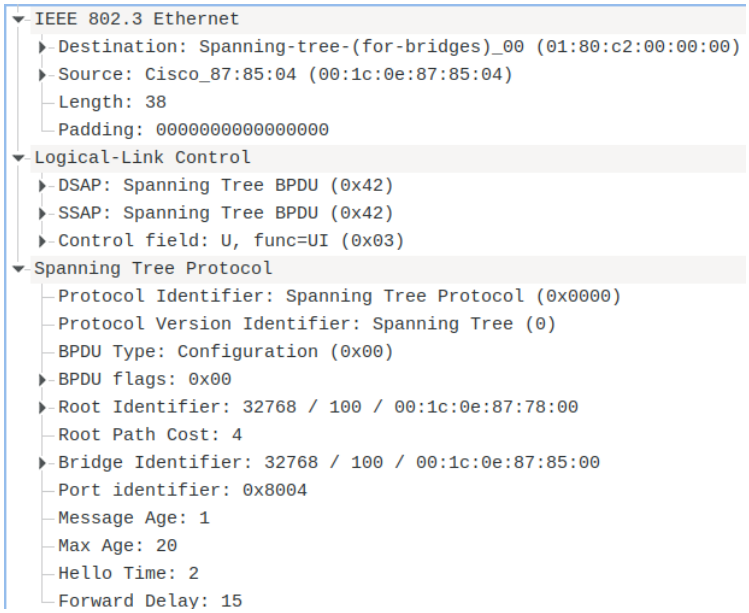
```
Every 1 second:
    if switch is root:
        Send BPDU on all trunk ports with:
            root_bridge_ID = own_bridge_ID
            sender_bridge_ID = own_bridge_ID
            sender_path_cost = 0
```

In cazul in care am primit un pachet de tip BPDU, daca acesta are un BDPU mai mic decât al nostru, atunci switch-ul de la care am primit acest pachet este considerat root. Mai mult, vom trimite acest pachet 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 [https://techhub.hpe.com/eginfolib/networking/docs/switches/5980/5200-3921_l2-lan_cg/content/499036672.htm] prezintă pe scurt structura lor. Aici [https://wiki.wireshark.org/uploads/8d3d0627231ab1e2fa5d3fe8be2390a7/stp.pcap] 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 si permit interacțiunea cu nivelul data link.

```
# 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, eth_frame, length)

# 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
```

Cerinte

În acest repo [https://gitlab.cs.pub.ro/rl/tema1-public] 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:

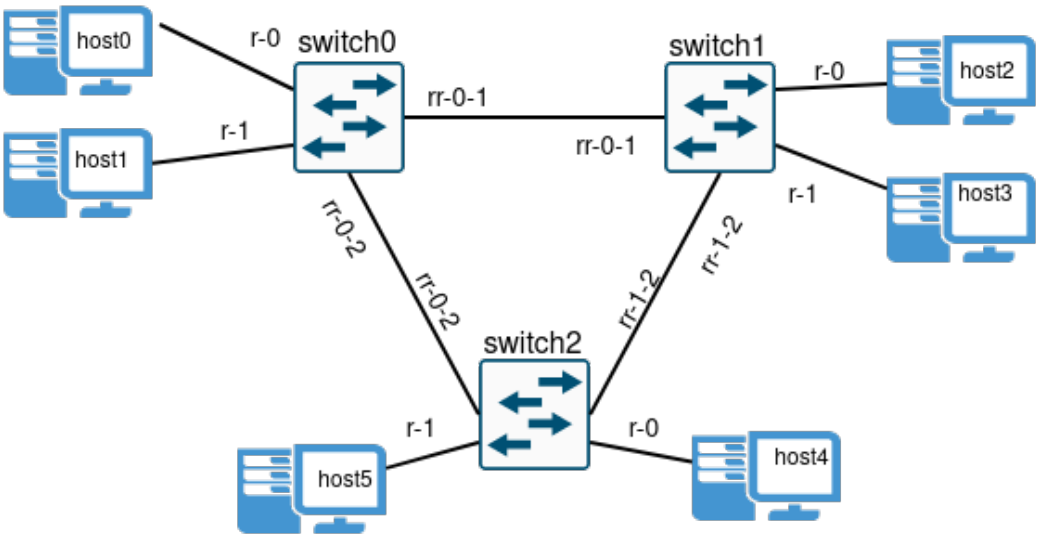
- **Procesul de comutare (30p)**. Va trebui sa implementați funcționalitatea descrisă in secțiunea **Procesul de Comutare**. Pentru acest exercițiu nu este nevoie sa implementați funcționalitatea referitoare la VLAN sau STP. Pentru a evita buclele, vom porni doar switch-urile 0 si 1.
- **VLAN (30p)**. Vom implementa funcționalitatea de Virtual Local Area Networks (VLANs). Fișierele de configurație ale switch-urilor se găsesc in 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.

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/ [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 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 ce cerințe ați făcut 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: 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 wrapper-ele din 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.