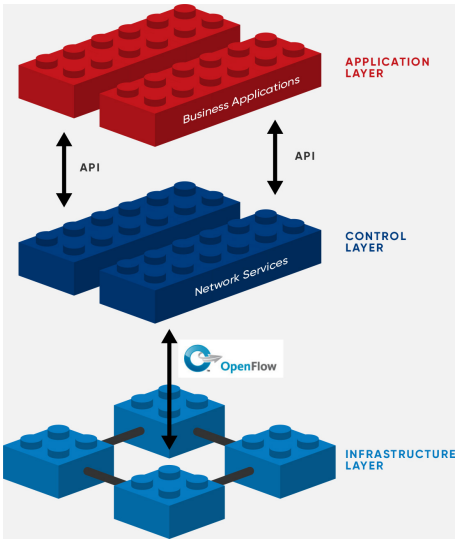
2 Prezentare generala

Prin intermediul acestei teme, va aducem in vedere cunostinte despre implementarea retelelor 5G, prin utilizarea arhitecturii de tip *SDN - Software Defined Network*. Arhitectura *SDN* reprezinta o noua paradigma de proiectare a retelelor, separand *data plane*-ul de *control plane* asigurand, in acest mod, o mai buna securitate si o mai usoara gestionare a rutelor.

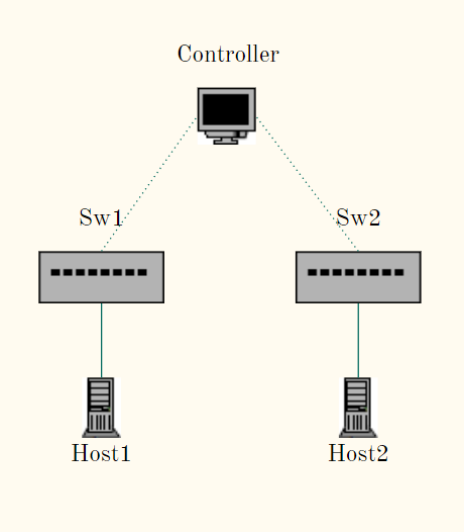


Aceasta este o reprezentare a intregii imagini pentru un *SDN*[1].

Ideea generala este aceea ca, in proiectarea retelelor, echipamentele de nivel 2 si nivel 3 (Switchurile si Routerele) nu mai iau decizii logice, acestea fiind exclusiv la latitudinea unui *Controller* care sa le gestioneze. Inca de la prima publicare a acestei idei, au fost identificate mai multe tipuri de atacuri numite *atacuri de teleportare*.

Se numeste *teleportare* acea vulnerabilitate a arhitecturii *SDN* prin care switch-uri *SDN* malitioase pot transmite informatie prin *control plane*, ocolind complet *data plane*-ul.

[1] `https://opennetworking.org/sdn-definition/`



Aceasta imagine prezinta doua host-uri, fiecare host legat de un Switch, iar Switch-urile legate la un acelasi *controller* logic.

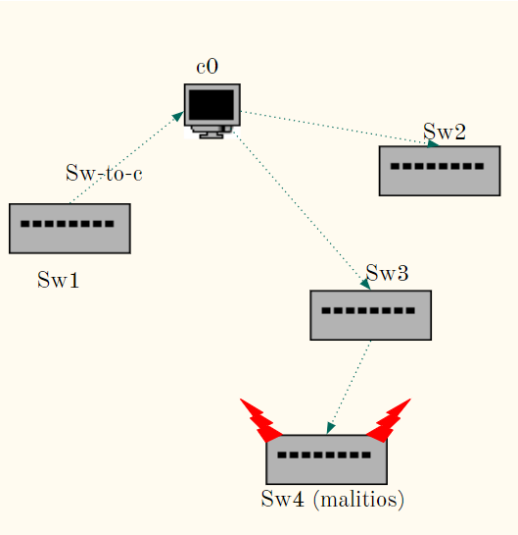
In cele ce urmeaza, ne vom concentra asupra *switch*-urilor malitioase. Este important de stiut ca nu exista restrictii pentru ce poate si ce nu poate face un *switch* - el poate crea si transmite orice tip de mesaj, dar nu poate stabili unde va fi plasat acest mesaj, poate altera *MAC*, *IP*, *ARP* si poate fi un insider. Modul de functionare al protocolului de comunicare utilizat este urmatorul:

- pe *switch*-uri sunt instalate, de catre *controllerul* logic, reguli de *matching*;
- in comunicare sunt permise numai pachetele care *match*-uiesc acele reguli;
- actiunile posibile care pot fi luate in legatura cu un pachet sunt *drop*, *send*, *modify header*.

Modul in care se realizeaza *teleportarea* este urmatorul:

- *switch-to-controller*: un *switch* trimite intentionat (sau nu) o informatie modificata catre *controller*;
- *controller-to-switches*: *controller*-ul reactioneaza la acest eveniment, transmitand informatia unuia sau mai multor *switch*-uri; *destination processing*: un *switch* malitios proceseaza informatia primita de la *controller*, putand cauta anumite *pattern*-uri sau transmitand informatia la un alt *switch* malitios.

Fie urmatoarea reprezentare:



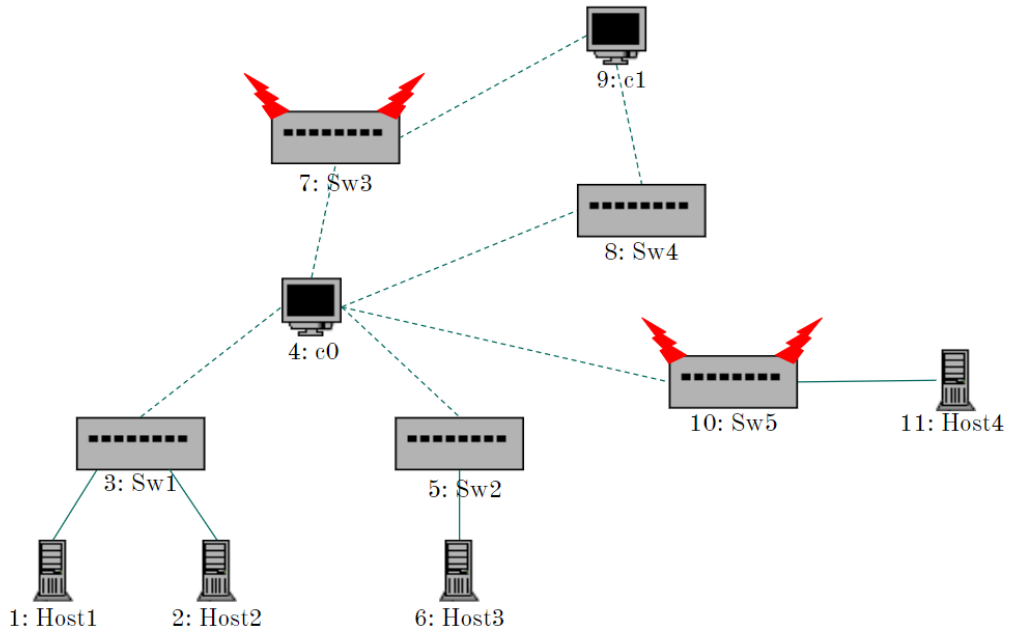
Aici, *Sw1* efectueaza un *switch-to-controller* catre *c0*, care efectueaza *controller-to-switches* catre *Sw2* si *Sw3*. Cand *Sw3* transmite informatia mai departe, ea ajunge la *Sw4* care este un *switch* malitios, loc in care se declanseaza *destination processing*.

Prin intermediul teleportarii, au fost formulate si exploatate 4 atacuri. Arhitectura de tip *SDN* reprezinta un fundament pentru retelele *5G* dezvoltate astazi, si sunt un punct de plecare in intelegerea acestora. Articolul din care au fost sintetizate aceste informatii si pe baza caruia au fost construite cerintele temei se gaseste la adresa citata [2].

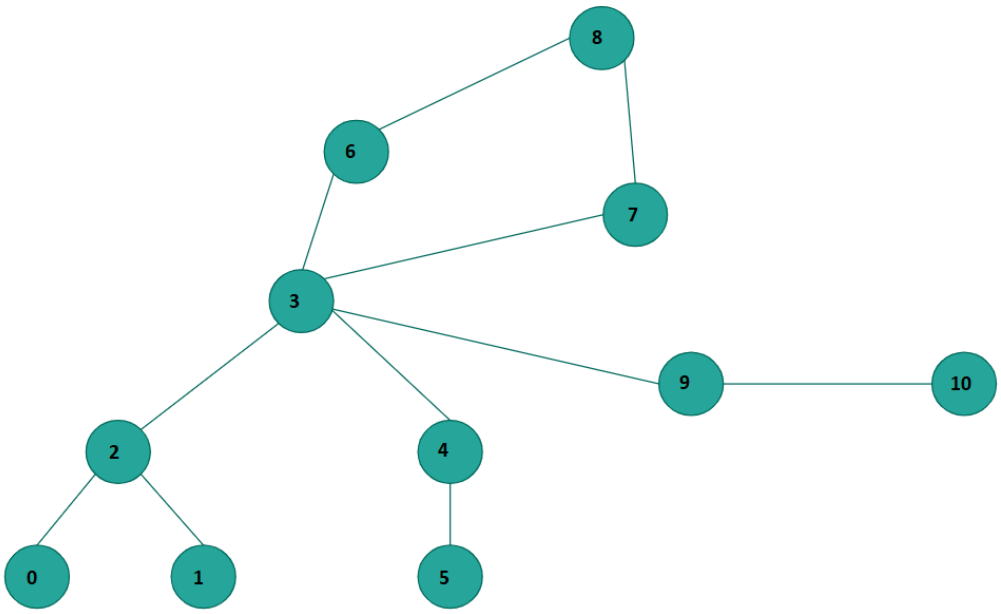
[2] `https://ieeexplore.ieee.org/document/7962003`

3 Formularea temei

Fie urmatoarea arhitectura de retea:



care poate fi privita ca un graf



unde stim, pentru fiecare element in parte, ce rol are, astfel:

- 1 - host;
- 2 - switch;
- 3 - switch malitios;
- 4 - controller logic.

Aceste informatii sunt stocate intr-un *array* de intregi, de forma:

```
roles = [1, 1, 2, 4, 2, 1, 3, 2, 4, 3, 1]
```

cu urmatoarea semnificatie: nodul `1` este, in prima imagine, un *host*, astfel ca il codificam prin rolul `1`, identic al doilea nod, al treilea nod este un *switch* obisnuit, astfel ca il codificam prin rolul `2`. Se procedeaza analog cu restul nodurilor. **Atentie!** In imagine, numerotarea nodurilor a fost facuta de la 1, dar consideram ca, in implementare, nodurile sunt de la 0. Astfel, pentru `n` noduri date, vom avea indecsii de la `0` la `n - 1`.

Toate cerintele care vor fi formulate sunt cerinte care se pot rezolva utilizand **cel mult** array-uri de elemente intregi, **NU** vor fi necesare conceptele de functii si proceduri.

3.1 Cerinta pentru nota 5

Fie reteaua de mai sus reprezentata prin $\mathcal{R} = (\mathcal{G}, roles)$, unde $\mathcal{G}$ este un graf dat prin matricea lui de adiacenta. Pentru a retine o matrice intr-un limbaj de asamblare, folosim aceeasi reprezentare ca la *array*-uri: o matrice este, in memorie, tot inlantuit alocata, doar ca in loc sa avem accesare prin intermediul unui singur index, avem accesare prin intermediul a doi indecsi. Avand aceasta informatie, sa se afiseze, la **STANDARD OUTPUT** (pe consola) toate echipamentele de retea (nodurile) care sunt direct conectate la un *switch* malitios: la ce *switch* malitios sunt conectate, care sunt aceste echipamente si ce rol au in retea. Reprezentarea se va face exclusiv prin index-ul lor in graful asociat.

Matricea de adiacenta pentru reteaua de mai sus este urmatoarea:

```
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0
1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0
0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0
0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0
0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0
0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0
0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0
```

Exemplu: pentru reteaua de mai sus, se va afisa pe consola:

```
switch malitios index 6: controller index 3; controller index 8;
switch malitios index 9: controller index 3; host index 10;
```

3.2 Cerinta pentru nota 7

Plecand din primul echipament din retea (i.e. din echipamentul cu `index 0`), sa se afiseze toate *host*-urile cu care acest echipament poate comunica (indiferent daca este interpus in comunicare un *switch* malitios). In plus, sa se decida si daca exista o conexiune (sigura sau nesigura, nu ne intereseaza) intre oricare doua echipamente.

Hint: se va aplica algoritmul **BFS**, care utilizeaza o coada. Pentru a nu ne complica foarte tare prin eliminarea elementelor din coada, putem sa avem un *array* care doar creste, si indexul curent la care am ajuns, considerand ca elementele de la stanga au fost "eliminate". De exemplu, pentru

```
queue = [0, 2, 1, 3, 4, 6, 7]
queueIndex = 2
queueLength = 7
```

inseamna ca elementul curent din coada este `1`. Nu eliminam elementul ca sa scapam de el, ci pur si simplu facem un `queueIndex := queueIndex + 1`. Pentru a adauga elemente, le punem la final (ne asiguram ca am declarat suficient spatiu pentru `queue`) si incrementam `queueLength`-ul.

Exemplu: pentru reteaua de mai sus, se va afisa pe consola:

```
host index 0; host index 1; host index 5; host index 10;
Yes
```

cu explicatia ca, plecand din `host index 0`, atunci il consideram si pe el ca fiind *host* accesibil. Pana la celelalte trei *host*-uri, este evidenta conexiunea prin intermediul grafului de mai sus. Pe a doua linie se va scrie un mesaj *Yes* sau *No*, dupa cum toate echipamentele pot comunica, fara sa tinem cont de faptul ca unele *switch*-uri pot fi sau nu malitioase.

Important! *Host*-urile vor fi afisate in ordinea descoperirii lor prin parcurgerea in latime.

3.3 Cerinta pentru nota 10

Consideram 2 *host*-uri din retea care vor sa comunice intre ele. Aceasta comunicare se poate realiza fie pe un **drum sigur**, fie pe unul nesigur (este interpus in comunicare un *switch malitios*). In cazul in care drumul este sigur, mesajul transmis de *host*-ul sursa va ajunge intact la *host*-ul destinatie. Altfel, mesajul va fi modificat de catre *switch*-ul malitios, ajungand intr-o forma indescifrabila la destinatie.

Administratorul retelei este totusi un adevarat specialist si, urmarind traficul de date reuseste, prin criptanaliza, sa descopere maniera in care *switch*-ul modifica mesajul. *Switch*-ul malitios preia mesajul original primit de la sursa si il cripteaza folosind *cifrul Caesar* cu deplasare *10*. Mesajul criptat este apoi transmis catre destinatie. In cazul in mesajul trece prin 2 *switch*-uri malitioase, acestea comunica intre ele pentru a nu realiza amandoua criptarea.

In contextul acesta, se introduc de la tastatura 2 *host*-uri si un mesaj format numai din literele mici ale alfabetului englez, reprezentand mesajul ce a ajuns in host-ul destinatie. Se cere sa se afiseze mesajul original transmis de sursa, avand in vedere faptul ca **daca exista macar un drum sigur** intre cele 2 *host*-uri, **va fi ales acest drum sigur** pentru realizarea comunicarii.

Important! *Inputul* pentru aceasta cerinta are alta structura decat cel primit pentru primele doua, intrucat vor trebui specificate cele doua hosturi care trebuie sa comunice. Puteti vedea structura *input*-urilor in sectiunea dedicata.

Urmatoarele doua exemple sunt pentru reteaua descrisa in cerinta.

Exemplul 1 - comunicare sigura

```
// Input:
0
2
hello

// Output:
hello
```

Exemplul 2 - comunicare nesigura

```
// Input:
0
10
rovvy

// Output:
hello
```

4 Transmiterea temei si evaluarea

4.1 Forma inputului

Pentru evaluare, va exista un *input* specializat pentru tratarea fiecareia dintre cerintele prezentate. Vor fi asigurate teste de evaluare automata pentru a se obtine nota 5 din tratarea corecta a primei cerinte, nota 7 pentru tratarea corecta, in plus, a celei de-a doua cerinte, si nota 10 pentru tratarea corecta a tuturor cerintelor. Programul va trebui sa raspunda corect inputurilor cu structura:

```
11          numar noduri
11          numar legaturi
0
2           prima legatura
1
2           a doua legatura
2
3           a treia legatura
5
4           a patra legatura
4
3           a cincea legatura
3
6           a sasea legatura
3
7           a saptea legatura
3
9           a opta legatura
6
8           a noua legatura
7
8           a zecea legatura
9
10          a unsprezecea legatura
1           aici incepe array-ul roles
1
2
4
2
1
3
2
4
3
1           aici se termina array-ul roles
3           numarul cerintei, poate fi 1, 2 sau 3
0           Observatie! Aceste citiri se fac doar
10          daca numarul cerintei este 3
rovvy
```

Important! Se asigura faptul ca NU vor fi grafuri cu un numar mai mare de 20 de noduri.

4.2 Forma outputului

In functie de cerinta care va fi rezolvata conform inputului, adica `1`, `2` sau `3`, outputul va avea urmatoarea forma:

4.2.1 Output 1

Vor fi afisate linii separate, initial *switch*-ul malitios, cu informatia *index* si numarul corespunzator, apoi *doua puncte*, apoi informatii despre echipamente, separate prin *punct si virgula*. Urmatorul *switch* malitios va fi afisat pe *o noua linie*.

```
switch malitios index 6: controller index 3; controller index 8;
switch malitios index 9: controller index 3; host index 10;
```

4.2.2 Output 2

Vom afisa pe *aceeasi linie* host-urile la care putem ajunge, prin indicarea *host*, *index* si numarul corespunzator. Dupa ce au fost insiruite aceste *host*-uri, se afiseaza linie noua, si apoi mesajul *Yes* sau *No*, dupa cum reteaua are conectivitate totala sa nu.

```
host index 0; host index 1; host index 5; host index 10;
Yes
```

4.2.3 Output 3

Vom afisa mesajul original transmis de sursa.

```
hello
```

ATENTIE! Sursele pe care le veti trimite vor fi evaluate automat, astfel ca va trebui sa va asigurati ca programul functioneaza **corect** pe toate cele trei inputuri si ca sunt afisate mesajele in forma prezentata! Sursele trimise vor fi evaluate pe un numar de 18 teste, dintre care 8 pentru validarea primei cerinte, 4 pentru validarea celei de-a doua cerinte, si 6 pentru validarea celei de-a treia cerinte. Nota finala se obtine in functie de numarul de teste pe care le-ati validat. **NU** se va lucra prin intermediul fisierelor text, testarea va fi asigurata prin simularea introducerii inputului de la tastatura. Programele citesc valorile de la tastatura si afiseaza la *standard output*.

5 Algoritmul BFS

Vom prezenta structura unui algoritm **BFS** in forma care sa va ajute in implementarea cerintei 2.2.

```
queueLen := 0
queueIdx := 0
queue    := []
visited  := [0, 0, ...0]    // are lungimea = nr de noduri
N                           // numarul de noduri
graph                       // matricea de adiacenta

queue[queueLen] := 0
queueLen        := queueLen + 1
visited[0]      := 1

while queueIdx != queueLen
    currentNode := queue[queueIdx]
    queueIdx    := queueIdx + 1

    if roles[currentNode] == 1
        print currentNode

    columnIndex := 0
    while columnIndex < N
        if graph[currentNode][columnIndex] == 1
            if visited[columnIndex] != 1
                queue[queueLen]      := columnIndex
                queueLen             := queueLen + 1
                visited[columnIndex] := 1
```