UNIVERȘITATEA „PETRU MAIOR”
DIN TÂRGU MUREȘ
FACULTATEA DE ȘTIINȚE ȘI LITERE
Programul de studiu: INFORMATICĂ

LUCRARE DE LICENȚĂ

Interfață grafică pentru configurare rețele

Coordonator științific:
Conf. dr. ing. Haller Piroska

Absolvent:
László János

-2017-

Cuprins

UNIVERSITATEA "PETRU MAIOR" DIN TG. MUREȘ **LUCRARE DE LICENȚĂ**

FACULTATEA DE ȘTIINȚE ȘI LITERE **Candidatul: László János**

Programul de studiu: Informatică

Anul absolvirii : 2017

Conducătorul științific : Haller Piroska **Viza facultății**

a) Tema lucrării de licență : Interfață grafică pentru configurare rețele

b) Problemele principale tratate :

- Introducere în domeniul rețelelor definite de software
- Proiectarea componentelor aplicației
- Implementarea aplicației folosind subsistemul WPF al framework-ului .NET
- Rutarea traficului într-o rețea de calculatoare

c) Bibliografia recomandată :

[1] Reaz Ahmed, Raouf Boutaba, Design considerations for managing wide area software defined networks, IEEE Communications Magazine, 15 iulie 2014
[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms (3rd Edition), The MIT Press, 31 iulie 2009
[3] Daniel F. Machedo, Dorgival Guedes, Luiz F. M. Vieira, Marcos A. M. Vieira, Michel Nogueira, Programmable Networks - From Software-Defined Radio to Software-Defined Networking, IEEE Communications Surveys & Tutorials, 11 februarie 2015
[4] Adam Nathan, WPF 4.5 Unleashed, Sams Publishing, 9 august 2013
[5] Chris Sells, Ian Griffiths, Programming WPF, O'Reilly Media; 2nd Edition, 7 septembrie 2007
[6] Andrew S. Tanenbaum, David J. Wetherall, Computer Networks (5th Edition), pag. 465-487, Pearson, 7 octombrie 2010

d) Termene obligatorii de consultații : săptămânal
e) Locul și durata practicii : laboratoarele UPM

Primit tema la data de : 16.02.2016

Termen de predare : 23.06.2017

Semnătura directorului de departament **Semnătura conducătorului**

Semnătura candidatului

INTRODUCERE

PROBLEMA

Creșterea în număr mare a dispozitivelor mobile, a site-urilor web și apariția serviciilor cloud, sunt motivele care fac industria rețelei să reexamineze arhitecturile de rețea tradiționale. Rețelele convenționale sunt ierarhice, aranjate într-o formă arborescentă, construite din switch-uri. Acestea, prin arhitectura lor statică nu îndeplinesc cerințele dinamice de calcul și stocare a marilor întreprinderi. Unele dintre tendințele cheie care duc la necesitatea unei noi paradigme de rețea sunt:

- SCHIMABAREA MODELELOR DE TRAFIC – În centrele de date, modelele de trafic încep să se schimbe. Spre deosebire de aplicațiile client-server, unde comunicarea are loc între client și server, aplicațiile de astăzi accesează servere din mai multe locații, generând o reacție „machine-to-machine" înainte de a returna datele prin clasicul model client-server. În același timp, utilizatorii schimbă modelele de trafic din rețea pentru că accesează diferite aplicații de pe diferite dispozitive.

- FOLOSIREA ACCEANTUATĂ A DISPOZITIVELOR MOBILE - Utilizatorii folosesc tot mai des dispozitivele mobile pentru a accesă rețele de socializare, contul firmei unde lucrează, așa că rețeaua este sub presiunea de a le servii pe acestea și în același timp să păstreze securitatea.

- APARIȚIA SERVICIILOR CLOUD - Întreprinderile folosesc cu entuziasm serviciile de cloud, ceea ce a dus la o creștere fără precedent a acestor servicii. Serviciile cloud trebuie să ofere securitate, conformitate, să se adapteze la reorganizările, fuziunile care pot schimba ipotezele peste noapte.

- "BIG DATA" - Centrele de date se confruntă cu sărcina descurajantă de a scala dimensiunea rețelei la dimensiuni anterioare de neimaginat, menținând conectivitatea.

SOLUȚIA

Software defined networking rezolvă problema prin decuplarea, dezasocierea sistemului care ia decizii în privința rutării traficului, de sistemele care stau la bază și au rolul de a trimite traficul către destinația selectată.

SDN este o abordare care permite administratorilor de rețea să inițializeze, controleze, administreze comportamentul unei rețele în mod dinamic prin intermediul interfețelor deschise și abstractizarea funcționalităților nivelelor de mai jos.

OBIECTIVELE APLICAȚIEI

Aplicația are rolul de a stabili rutarea pachetelor într-o rețea. Utilizatorul importă o topologie din Mininet, selectează echipamentele între care dorește să stabilească un flux, două câte două, ajustează setările fluxului precum protocolul, lățimea de bandă, iar aplicația caută ruta cea mai scurtă, ținând cont de fluxurile stabilite anterior.

Când s-au stabilit fluxurile dorite, acestea se trimit controller-ului, în acest caz Floodlight, care aplică în rețeaua din Mininet această configurație.

Cu ajutorul aplicației se pot construi topologii și se pot stabili fluxurile din acestea, nu este nevoie de a importa una. O alta trăsătură este posibilitatea de a salva ceea ce s-a făcut până la un moment dat și a se continua altă dată.

MOTIVUL ALEGERII TEMEI

Motivul pentru care am ales această temă este că îmi doream să fac o aplicație desktop cu o interfață grafică și nu aveam experiență în construirea lor. Mi s-a părut o ocazie oportună ca să îmi dezvolt această calitate. Știind că este vorba despre lucrarea de licență m-am gândit că efortul depus va fi unul îndeajuns pentru a mă dezvolta.

La timpul alegerii temei mi se părea cea mai atractivă dintre cele care mi-au fost sugerate ca opțiuni de către profesorul îndrumător, pentru că era vorba despre construirea unei interfețe grafice, dar și despre configurarea unei rețele cu ajutorul acesteia, încă un domeniu în care aș căpăta cunoaștere.

STRUCTURA DOCUMENTULUI

Capitolul 1 prezintă software-ul și framework-ul cu ajutorul cărora s-a făcut această aplicație. Conține informații despre SDN, emularea rețelelor și puține informații despre sistemele folosite, emulatorul de rețea Mininet, controller-ul SDN OpenFlow Floodlight.

Capitolul 2 începe cu prezentarea problemei și soluției. Este prezentat modul în care aplicația este gândită. În continuare urmează descrierea interacțiunii dintre componentele care sunt în planul problemei, adică aplicația mea, controller-ul Floodlight și emulatorul Mininet.

Se descriu funcționalitățile necesare interfeței grafice pentru a permite utilizatorului să aibă o imagine cât mai clară asupra muncii pe care o desfășoară.
Se descrie structura fișierelor în care se salvează sau din care se importă configurații.
Ultimul punct din acest capitol descrie algoritmii folosiți în această aplicație, aceștia fiind din teoria grafurilor:

- algoritmul care verifică dacă există o cale între două noduri
- algoritmul lui Dijkstra pentru cea mai scurtă cale

Capitolul 3 prezintă modul în care aplicația este implementată. Sunt descrise componentele care alcătuiesc interfața grafică, cum ar fi meniul, zona de creare a topologiei, elementele de rețea: PC, Switch, Router și link, aceste au rolul de a lucra cu datele care rezulta din crearea și modificarea topologiei rețelei. Funcționalități precum codificarea și transpunerea topologiei într-o matrice de adiacență și costuri, calcularea conexității a două noduri, calcularea drumului cel mai scurt între două noduri.

ABREVIERI

SDN – Software Defined Networking
WPF – Windows Presentation Foundation
PDU – Protocol Data Unit
ARP – Address Resolution Protocol
JSON – Javascript Object Notation

Cap.1. ASPECTE PRELIMINARE

1.1. SOFTWARE DEFINED NETWORKING

Consensul actual în ceea ce privește programabilitatea rețelelor, indică spre separarea datelor și a controlului, în literatura de specialitate cunoscut sub numele SDN. SDN este o nouă paradigmă de rețea, prin care un program de nivel înalt controlează comportamentul rețelei. SDN-urile se caracterizează prin existența unui sistem de control fiind capabil să controleze mecanismele planului de date printr-o interfață de programare bine definită. În rețelele cu fir, de exemplu, elementele de comutare expun o interfață de programare care permite software-ului să inspecteze, să stabilească și să modifice intrările tabelului de flux, cum ar fi cu switch-urile OpenFlow.

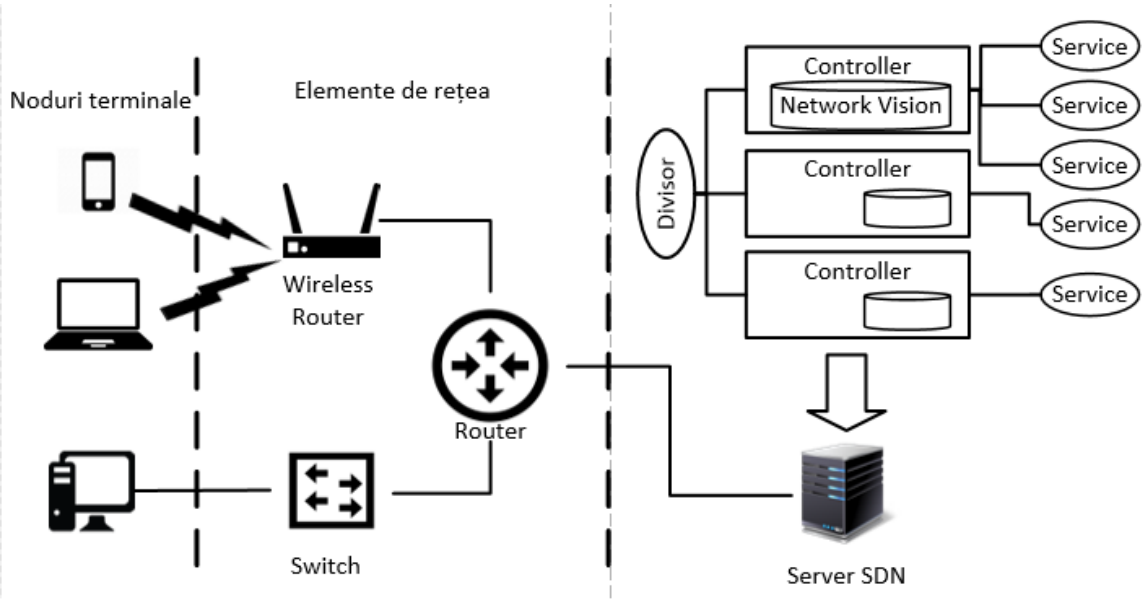

Figura 1.1 Componente SDN

O arhitectură SDN poate fi separată în trei sau patru componente:

- **Noduri terminale programabile** - pot fi clienți, servere sau alte tipuri de dispozitive de calcul.
- **Elementele de rețea programabile** - vor executa un program de control
- **Controller-ul**, controlează funcționarea elementelor programabile. Controller-ul poate implementa de asemenea, un divizor
- **Divizorul** - împarte rețeaua în rețele virtuale, fiecare având un program separat de control.

Fig. 1.1 ilustrează această organizare pentru o rețea cu elemente cu fir și fără fir. În vizualizarea SDN tradițională, planul de date programabil se oprește la elementele de comutare. Cu toate acestea, programabilitatea este de asemenea benefică la nivelul nodurilor terminale, după cum sugerează mai multe rezultate recente, care arată îmbunătățiri obținute datorită unei programabilități sporite a nodurilor terminale. În continuare, descriem fiecare dintre aceste componente.

1.1.1. NODURI TERMINALE PROGRAMABILE

În SDN-urile implementate în prezent utilizând OpenFlow, nodurile terminale nu sunt programabile. Cu toate acestea, rețelele de acces, de exemplu WiFi ar trebui, de asemenea, să fie optimizate pentru a oferi o experiență de rețea mai bună. În timp ce în rețelele celulare antenele coordonează accesul la mediu, standardele bazate pe conflictele din rețea, cum ar fi WiFi, efectuează control distribuit, astfel încât în prezent nu este posibilă schimbarea comportamentului unui nod greșit configurat sau adaptarea operării nodurilor terminale pentru a găzdui un nou serviciu.

1.1.2. ELEMENTE DE REȚEA PROGRAMABILE

Principiul de bază al SDN constă în programarea elementelor rețelei. Acest lucru nu se întâmplă prin injectarea codului, ci cu o interfață care oferă un set de operații. O astfel de interfață este OpenFlow, care este adaptată funcționării router-elor și switch-urilor. Operația de redirecționare respectă un principiu simplu: fiecare pachet primit pe una dintre porturile switch-ului este inspectat și generează o interogare la tabela de flux. Dacă interogarea nu reușește, pachetul este eliminat. De obicei, există în continuare posibilitatea de a defini un comportament implicit în caz de eșec. Odată identificată destinația pachetului, acesta trece prin interconexiunile interne ale switch-ului pentru a obține portul destinație. Astfel, în SDN-urile bazate pe OpenFlow, operațiile programabile sunt: forward, drop, modify header și send to controller, cea din urmă fiind folosită pentru a face față fluxurilor necunoscute sau neașteptate.

1.1.3. CONTROLLER-UL

Controller-ul este entitatea din rețea care monitorizează și modifică comportamentul elementelor de rețea și nodurilor terminale. În SDN, controller-ul utilizează un set de API-uri pentru a interacționa cu elementele programabile. Controller-ul trimite comenzile sale către elementele programabile printr-un canal de control securizat. Făcând o analogie cu caracteristicile unui sistem de operare, controller-ul funcționează ca un middleware pentru rețea: expune o interfață de programare de nivel superior pentru dezvoltatori care

abstractizează detaliile de operare ale fiecărei componente și sprijină coexistența diferitelor programe de control. Programele de control pot fi dezvoltate folosind mai multe limbi de nivel înalt, iar controller-ul traduce acele programe în acțiuni care pot fi trimise la fiecare element de rețea.

Controller-ele SDN au două interfețe, numite Northbound și Southbound. Interfețele Southbound se ocupă de interacțiunea dintre controller, elementele rețelei și interfețele programabile ale nodurilor terminale. Un exemplu de API Southbound este OpenFlow. Interfața Northbound este punctul de comunicare dintre controller și aplicațiile de control. Northbound este o interfață de programare care permite programelor de nivel înalt să controleze rețeaua.

Pentru mai multe detalii despre SDN, citiți din [1, 3, 7].

1.2. EMULAREA REȚELELOR

Emularea rețelelor este o tehnică pentru testarea performanțelor aplicațiilor reale printr-o rețea virtuală. Acest lucru este diferit de simularea unei rețele, unde se aplică modele de rețele, modele de trafic matematice, canale și protocoale. Scopul este de a evalua performanța, de a anticipa impactul schimbării sau un mod de a optimiza procesul de alegere al tehnologiilor de folosit în implementarea rețelei.

Emularea unei rețele diferă de simularea uneia prin faptul că un emulator de rețea pare a fi o rețea. Computerele, laptop-urile sau orice alte sisteme finale, pot fi atașate de emulator și acestea se vor comporta ca și cum ar fi atașate de o rețea reală. Un emulator de rețea emulează rețeaua care conectează computerele, nu sistemele finale.

Simulatoarele de rețea sunt programe care rulează pe un singur computer, iau o formă abstractă a traficului din rețea, cum ar fi un proces de sosire a fluxului și oferă statistici de performanță, cum ar fi ocuparea buffer-ului în funcție de timp.

Emularea de rețea reprezintă actul de introducere a unui dispozitiv într-o rețea de testare, de obicei într-un mediu de laborator, care modifică fluxul de pachete astfel încât să imite comportamentul unei rețele reale, cum ar fi un LAN sau WAN. Acest dispozitiv poate fi un calculator care rulează software pentru a efectua emularea de rețea sau un dispozitiv de emulare dedicat. Dispozitivele includ o varietate de atribute standard de rețea în competențele lor, cum ar fi: timpul de deplasare în rețea (latența), lățimea de bandă disponibilă, gradul de pierdere a pachetelor, duplicarea pachetelor, reordonarea pachetelor, coruperea, modificarea pachetelor și severitatea bruiajului rețelei. Emulatoarele de rețea de

nivel superior, pot să imite erorile fizice tipice ale nivelului fizic, cum ar fi rata de eroare a biților, pierderea semnalului și altele.

Este cunoscut faptul că rețelele sunt imperfecte, fie acestea private sau publice. Acestea produc întârzieri, erori, pierd pachete și pică. Scopul principal al emulării rețelelor este crearea unui mediu în care utilizatorii să poată conecta dispozitivele, aplicațiile, produsele și serviciile lor și să evalueze performanța, stabilitatea sau funcționalitatea față de scenariile de rețea din lumea reală. Odată testate într-un mediu controlat, care oferă condițiile unei rețele reale, utilizatorii pot avea încredere că lucrul testat de ei va funcționa conform așteptărilor.

1.3. SISTEME DE EMULAT REȚELE

Emulatoarele pot fi construite pentru testarea aplicațiilor dezvoltate în mediile de dezvoltare integrate sau a celor dezvoltate pentru browser web, cu alte cuvinte testarea site-urilor web. Printre programele software de simulare și emulare a rețelelor sunt OPNET, NetSim, Mininet, NS-3.

OPNET este un program software care face managementul performanței unei rețele sau a unei aplicații. Managementul performanței unei aplicații constă în monitorizarea, gestionarea performanței și a disponibilității aplicațiilor software. Managementul performanței are rolul de a detecta și diagnostica problemele complexe legate de performanța aplicațiilor, pentru a menține nivelul așteptat al serviciului.

NetSim este un program software pentru simularea și emularea rețelelor, utilizat pentru proiectarea și planificarea rețelelor, aplicații pentru apărare și pentru cercetare și dezvoltare în domeniul rețelelor. Diferite tehnologii, cum ar fi: Cognitive Radio, Wireless Sensor Networks, Wireless LAN, Wi Max, TCP, IP, etc. sunt acoperite în NetSim.

În cele anterioare am discutat despre sistemele de emulat rețele. Următoarele două subcapitole descriu sistemul de emulat rețele folosit în testarea aplicației dezvoltate de mine și controller-ul folosit pentru modificarea fluxul de pachete.

1.3.1. MININET

Mininet este un emulator de rețea care creează o rețea de gazde virtuale, switch-uri, controlere și linkuri. Mininet rulează software de rețea Linux, iar switch-urile săle suportă OpenFlow pentru o rutare personalizată și SDN. Mininet sprijină cercetarea, dezvoltarea, învățarea, dezvoltarea de prototipuri, testarea, depanarea și orice alte sărcini care ar putea beneficia de o rețea experimentală completă pe un laptop sau pe un calculator.

Mininet:

- Oferă un mediu de testare simplu pentru dezvoltarea aplicațiilor OpenFlow
- Permite mai multor dezvoltatori să lucreze în mod simultan și independent pe aceeași topologie
- Susține testele de regresie la nivel de sistem, care sunt repetabile și ușor de ambalat
- Permite testarea complexă a topologiei, fără a fi necesară conectarea unei rețele fizice
- Include un CLI care este conștient de topologie și este familiarizat cu OpenFlow pentru depanarea sau rularea testelor la nivel de rețea
- Sprijină topologiile arbitrare personalizate și include un set de bază de topologii parametrizate
- Pregătirea acestuia pentru utilizare se poate face într-un set de pași minimali.
- Oferă de asemenea un API Python simplu și extensibil pentru crearea și experimentarea rețelelor

Mininet oferă o modalitate ușoară de a obține un comportament corect al sistemului, în măsura în care este susținut de hardware, de performanță, și de a experimenta topologii.

Rețelele Mininet rulează cod real, incluzând aplicațiile standard de rețea Unix / Linux, precum și nucleul real și stiva de rețea, inclusiv orice extensii de nucleu pe care le puteți avea, atâta timp cât sunt compatibile cu spațiile de nume ale rețelei.

Din acest motiv, codul pe care îl dezvolți și îl testezi pe Mininet, pentru un controler OpenFlow, un comutator modificat sau o gazdă, se poate muta într-un sistem real cu modificări minime, pentru testarea în lumea reală, evaluarea performanțelor și implementarea. Important, acest lucru înseamnă că un design care funcționează în Mininet se poate deplasa, de obicei, direct la comutatoarele hardware pentru redirecționarea pachetelor.

Aproape fiecare sistem de operare virtualizează resursele de calcul făcând abstracție de proces. Mininet utilizează virtualizarea bazată pe proces pentru a rula gazde și comută pe un singur kernel. De la versiunea 2.2.26, Linux a sprijinit spațiile de nume de rețea, o caracteristică de virtualizare care oferă proceselor individuale interfețe de rețea separate, tabele de rutare și tabele ARP. Arhitectura completă a Linux adaugă procesele, spațiile de nume ale utilizatorilor, limitele CPU și de memorie pentru a oferi o virtualizare completă la nivel de sistem, dar Mininet nu necesită aceste caracteristici suplimentare. Mininet poate crea switch-uri OpenFlow pentru nucleu sau pentru utilizatori, controllere pentru a controla

switch-urile și gazde pentru a comunica prin rețeaua simulată. Mininet conectează switch-urile și gazdele folosind perechi virtuale, virtual ethernet. În prezent Mininet depinde de nucleul Linux, în viitor poate sprijini alte sisteme de operare cu virtualizare pe procese, cum ar fi containerele Solaris sau FreeBSD.

Mininet este aproape în întregime dezvoltat în Python, cu excepția unei mici utilități scrise în C.

Pentru mai multe detalii vizitați [9].

1.3.2. FLOODLIGHT

Controller-ul SDN Open Floodlight este un controller OpenFlow bazat pe tehnologii Apache și Java. Este susținut de o comunitate de dezvoltatori, inclusiv un număr de ingineri de la Big Switch Networks.

Controller-ul Floodlight este destinat a fi o platformă pentru o gamă largă de aplicații de rețea. Acestea reprezintă principalul obiectiv al dezvoltării Floodlight, deoarece oferă un mediu pentu soluționarea problemelor legate de rețelele.

Despre:

- funcționează cu switch-uri fizice și virtuale care înțeleg protocolul OpenFlow
- aflat sub licența Apache, vă permite să utilizați Floodlight pentru aproape orice scop
- testat și îmbunătățit activ de o comunitate de dezvoltatori profesioniști
- dezvoltat de o comunitate deschisă de dezvoltatori care primesc cu plăcere contribuțiile de cod ale participanților activi și distribuie în mod deschis informații despre starea proiectului, foaia de parcurs, bug-uri, etc.
- ușor de instalat și executat.

OpenFlow este un standard deschis gestionat de Open Networking Foundation. Specifică un protocol prin intermediul unui switch. Acest protocol de comunicații oferă acces la nivelul 3 al unui switch sau al unui router prin rețea. OpenFlow permite controller-ului de rețea să determine calea pachetelor într-o rețea de switch-uri.

Un controller poate modifica comportamentul echipamentelor din rețea printr-un set de instrucțiuni de redirecționare. Floodlight este proiectat să funcționeze cu numărul tot mai mare de switch-uri virtuale și fizice, router, și puncte de acces care acceptă standardul OpenFlow.

Floodlight are în prezent patru metode pentru expedierea pachetelor, care au comportamente diferite și lucrează cu topologiile de rețea după cum urmează:

- Forwarding: activată implicit, Forwarding-ul permite redirecționarea pachetelor end-to-end între oricare două dispozitive din următoarele topologii de rețea:
    - În cadrul unei insule OpenFlow : Când un dispozitiv A trimite un pachet către dispozitivul B în aceeași insulă OpenFlow, Forwarding-ul calculează calea cea mai scurtă între A și B.
    - Insule OpenFlow cu insule non-OpenFlow între ele: Fiecare insulă OpenFlow poate avea exact o legătură care se leagă de o insulă non-OpenFlow. În plus, insulele OpenFlow și non-OpenFlow împreună nu pot forma bucle, așa cum arată figura 1.3. Fiecare dispozitiv ar avea un punct de legătură pe fiecare insulă OpenFlow. Redirecționarea calculează o singură cale scurtă în fiecare insulă OpenFlow și se așteaptă ca pachetele să fie transmise în insulele care sunt non-OpenFlow, adică presupunând că fiecare insulă non-OpenFlow este un singur domeniu de broadcast L2.

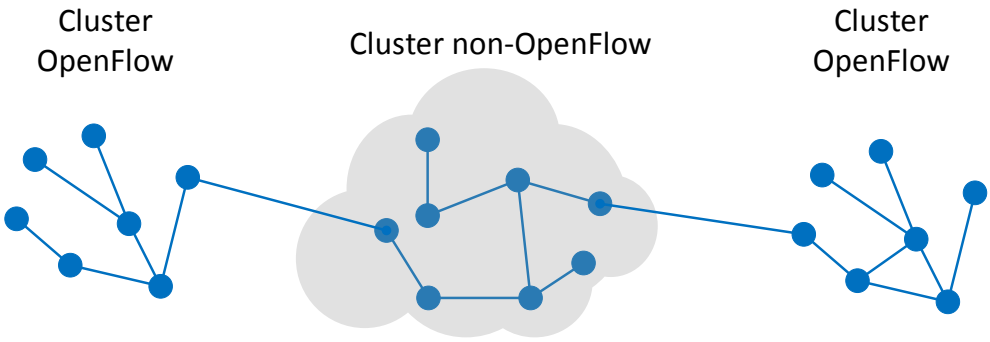

Figura 1.1 Exemplu de topologie care funcționează cu Floodlight

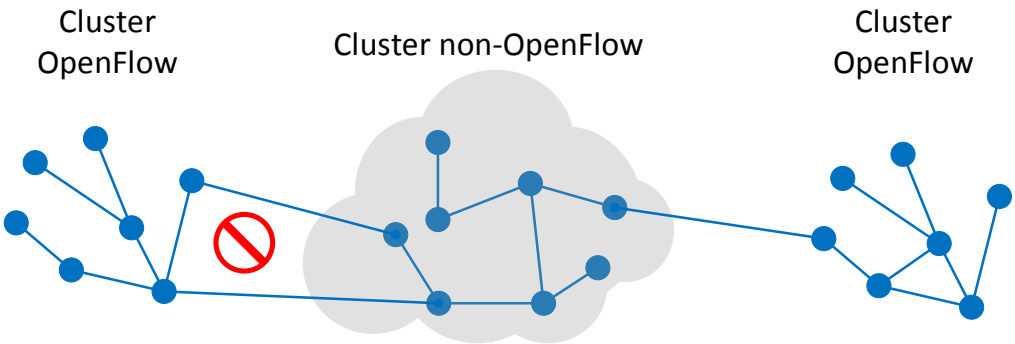

Figura 1.2 Exemplu de topologie care nu funcționează cu Floodlight pentru că cluster-ul OpenFlow din stânga are două legături cu clusterul non-OpenFlow

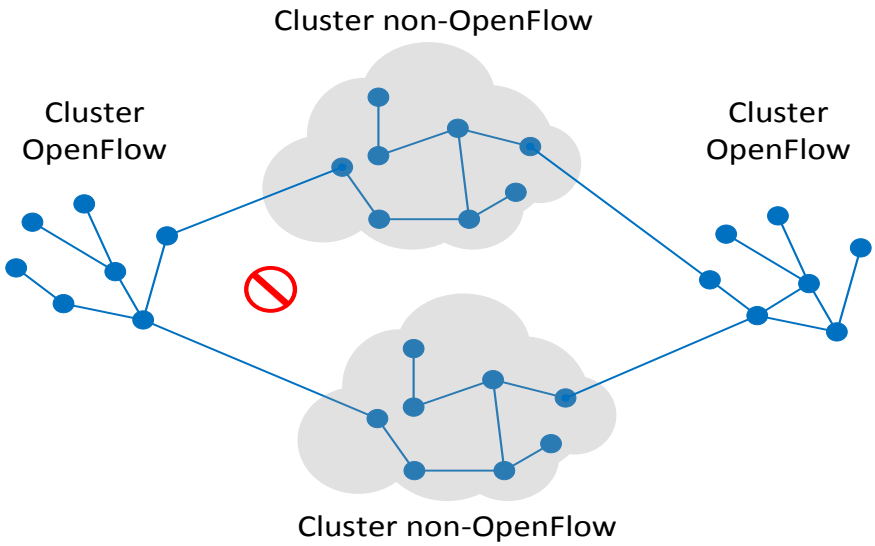

Figura 1.3 Exemplu de topologie care nu funcționează cu Floodlight pentru că clusterele OpenFlow și non-OpenFlow sunt conectate în ciclu

- Switch Learning: Un simplu switch L2 cu abilitatea de a învăța
    - Pentru utilizarea cu orice număr de clustere OpenFlow, chiar și în cazul clusterelor L2 care nu sunt OpenFlow
    - Nu poate funcționa dacă switch-urile dintr-o cluster formează o buclă sau o topologie de clustere formează o buclă
    - ineficient în transmiterea performanțelor comparativ cu celelalte abordări
- Static Flow Entry Pusher: permite setarea fluxului în fiecare switch care face parte din flux, creând astfel căi de redirecționare pe baza alegerii explicite a porturilor, creând astfel căi de redirecționare pe baza alegerii explicite a portului
- Circuit Pusher care permite utilizatorilor să instaleze căi de redirecționare în rețea
    - Este construit pe Static Flow Entry Pusher, Device Manager și se bazează pe API-ul serviciilor de rutare pentru a construi cele mai scurte trasee într-o singură insulă OpenFlow.

Un cluster OpenFlow este un set conectat de switch-uri OpenFlow cu computere conectate la oricare dintre switch-uri. În mod analog, un cluster non-OpenFlow este un set conectat de switch-uri non-OpenFlow cu dispozitive conectate la oricare dintre ele.

Dintre aceste 4 variante aplicația folosește Static Flow Entry Pusher pentru setarea rutelor în rețea. Pentru mai multe detalii despre Floodlight vizitați [8].

Cap.2. ASPECTE DE PROIECTARE

2.1. CERINȚE

Se consideră o topologie de rețea construita în Mininet. Aceasta având switch-uri, posibil conectate direct între ele prin unul sau mai multe link-uri, sau indirect prin alte switch-uri și gazde conectate la switch-uri. Gazdele au configurate plăcile de rețea, au adrese IP și MAC. Switch-urile au configurat ID-ul, porturile. Controller-ul Floodlight este pornit. Utilizatorul nu are o imagine vizuală a topologiei, decât dacă el a construit-o și o are în memorie sau dacă și-o desenează cu ajutorul comenzi *net* din Mininet, care afișează pentru fiecare echipament din rețea, în format text, echipamentul de rețea urmat de legăturile directe pe care le are cu restul rețelei. Aceasta sarcină devenind tot mai grea odată cu dimensiunea rețelei. Conform acestei premise putem observa că rutarea încă nu este rezolvată, o altă problemă este efortul pe care utilizatorului trebuie să-l facă pentru a vizualiza topologia. Având aceste probleme putem observa necesitatea unui soluții care să faciliteze utilizatorului rețelei vizualizarea și configurarea traficului din aceasta.

Definim următoarele cerințe:

- Construirea unei aplicații cu interfață grafică care are rolul de a oferi o imagine vizuala a topologiei
- Posibilitatea de a edita configurația echipamentelor din rețea.
- Salvarea topologiei și a configurațiilor efectuate pana la un moment dat pentru posibilitatea de a continua altă dată
- Încărcarea/deschiderea unei topologii anterior salvate.
- Importarea unei topologii dintr-un fișier care conține output-ul comenzii *net* din Mininet
- Stabilirea rutei pachetelor care circulă între două gazde

2.2. ARHITECTURA APLICATIEI

Figura de mai jos ilustrează, într-un mod general, cum se realizează interacțiunea între componentele majore, incluzând aplicația mea, care fac parte din rezolvarea problemei descrise în subcapitolul Cerințe. Se observă că după ce s-au stabilit flux-urile dorite și s-au configurat echipamentele din rețea, aplicația trimite flux-urile către controller pentru a le aplica în switch-urile din rețea.

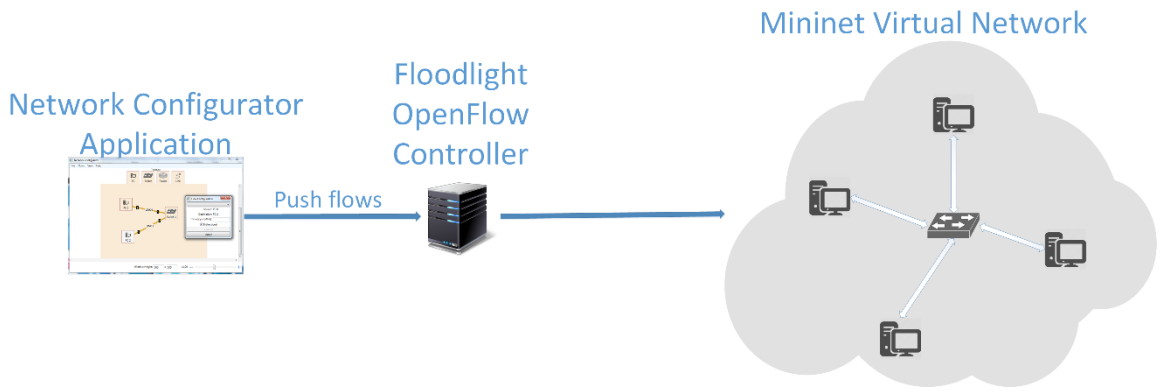

Figura 2.1

Diagrama use case de mai jos descrie interacțiunea utilizatorului cu aplicația.

După cum se observă, la pornirea aplicației utilizatorul poate construi o configurație luând, prin procedeul de drag&drop, echipamente și link-uri din lista de echipamente și le pune, aruncă în zona de configurare, construire a rețelei. Aici el poate conecta echipamentele între ele cu link-uri, edita proprietățile echipamentelor și link-urilor. Poate încărca, deschide o configurație sau după ce a lucrat la ea o poate salva. Dacă are nelămuriri în privința modului de funcționare al aplicație poate deschide manualul utilizatorului. Considerând aceste aspecte, în funcționarea aplicației se vor distinge trei faze de lucru principale. Prima fază constă în achiziționarea datelor topologiei rețelei, fie dintr-o configurație salvată anterior, fie din Mininet, și afișarea lor în zona de configurare creată în acest scop. După terminarea acestei faze, începe construirea, modificarea și configurarea echipamentelor și link-urilor. Următoarea fază constă în trimiterea datelor controller-ului, care are ca și rol aplicarea datelor primite de către aplicație în rețea. Pentru ca datele să fie cu succes trimise controller-ului, înainte de a le trimite trebuie să setăm adresă la care se află controller-ul, în caz contrar vom fi notificați că o eroare a apărut la conectare.

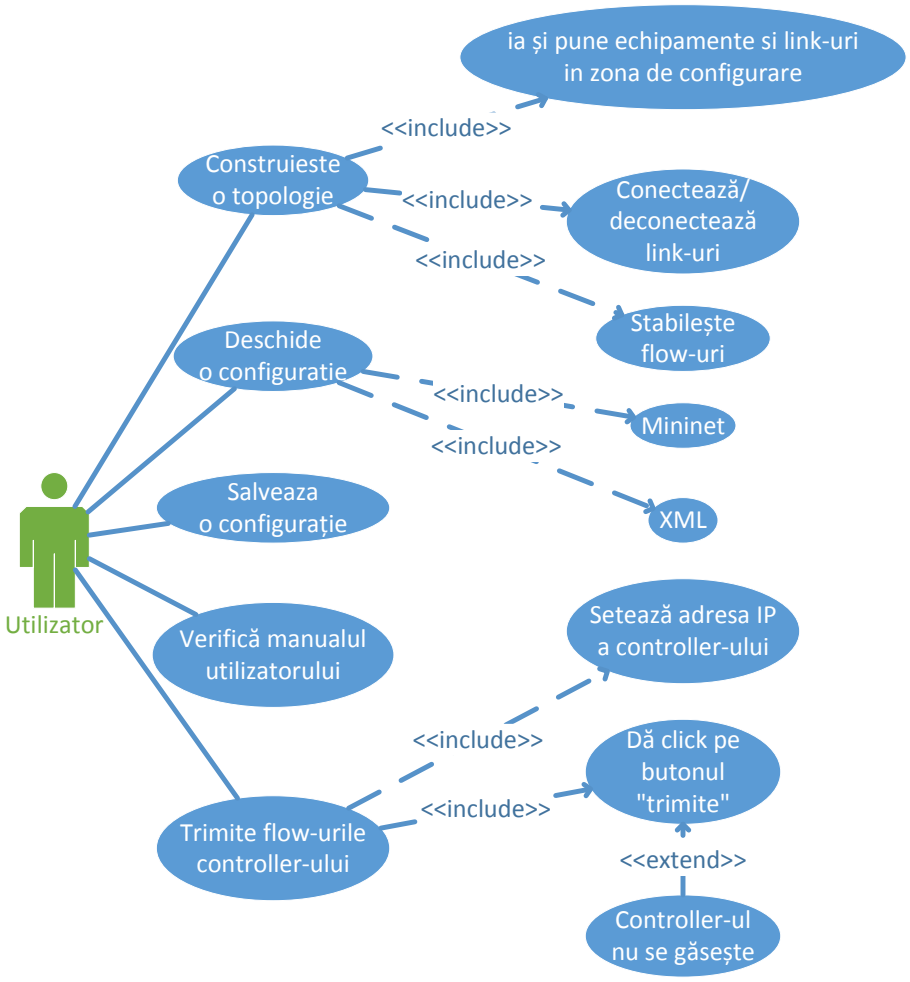

Figura 2.2

2.2.1. COMPONENTE

Meniul, conform cerințelor aplicației, conține funcționalități precum:

- deschiderea, citirea, salvarea unei configurații
- golirea zonei de configurare
- stabilirea adresei controller-ului
- trimiterea flux-urilor
- posibilitatea de afișa/ascunde lista cu echipamente de rețea
- vizualizarea unei liste cu toate flux-urile stabilite până la un moment dat
- deschiderea manualului utilizatorului
- Închiderea aplicației

Unele dintre aceste acțiuni, precum golirea zonei de configurare, închiderea aplicației, ar putea fi declanșate neintenționat, așa că utilizatorul trebuie întrebat dacă dorește să salveze ceea ce a lucrat pana acum sau dacă dorește să anuleze acțiunea tocmai selectată.

Zona de configurare, unde se construiește topologia, este asemenea unei hărți. Aceasta are rolul de a oferi o imagine vizuală a rețelei importate. Conține elemente de rețea precum gazde, switch-uri, routere și link-uri, acestea pot fi mutate cu drag&drop după cum dorește utilizatorul pentru a-și facilita efortul depus pentru vizualizarea topologiei. Se poate modifica lățimea și înălțimea acesteia pentru a putea plasă echipamentele de rețea conform dorințelor utilizatorului. Totodată, pentru a nu ocupa spațiu în plus, zona se redimensionează automat atunci când o anumita suprafață nu conține niciun echipament. O altă trăsătură importantă este abilitatea de a mări și micșora dimensiunea zonei vizibile, pentru a putea avea o imagine de ansamblu asupra topologiei. Pornind de la faptul că putem mări și micșora dimensiunea zonei vizibile observam necesitatea abilitații de a putea derula harta cu ajutorul unor bare de derulare orizontale și verticale.

Gazdele sunt, dacă facem analogia cu un graf, nodurile între care se stabilește flux-ul. Acestea pot fi adăugate, șterse, mutate și editate odată ajunse în zona de configurare. Din punctul de vedere al aplicației, relevant este ca o gazda să aibă o placă de rețea la care se poate conecta un link. Pentru editarea proprietăților gazdei este nevoie de o fereastra unde se pot seta numele gazdei, adresă IP și MAC a plăcii de rețea.

Switch-ul are rolul de a înainta traficul primit pe un port, printr-un alt port, numit port de ieșire. Switch-ul, la fel ca și oricare alt echipament din zona de configurare se poate adaugă, șterge, muta și edita cu ajutorul unei ferestre. Acesta se identifică cu un ID în aplicație, iar în Mininet cu un ID în alt format. În fereastra de editare se poate seta numele switch-ului, ID-ul din Mininet și numărul de porturi disponibile pentru a conecta link-uri. În momentul în care un link se conectează la switch, acesta se conectează la primul port liber.

Router-ul din puct de vedere rețelistic este total diferit de un switch, dar în aceasta aplicație router-ul este la fel ca și un switch, doar că fiecare port al router-ului este o placa de rețea cu adresă IP și MAC. Când se conectează un link la un router, o fereastra apare cu toate porturile goale, dintre care unul se selectează pentru a conecta link-ul. În fereastra de editare a router-ului se poate seta numele router-ului, numărul de porturi, adresele IP și MAC ale fiecărui port, totodată se poate vedea și numele link-ului care este conectat la un anumit port.

Linkul este elementul care face legătura între două echipamente de rețea. Acesta are un nume care poate fi schimbat și desigur o lățime de bandă. Linkul afișează pe ambele sale capete numărul portului la care este conectat, -1 atunci când nu e conectat. În fereastra de

editare a link-ului se poate seta numele și lățimea de banda. Linkul când nu este conectat se poate muta prin zona de configurare, fie trăgând de unul dintre capete, fie tot linkul.

Alte componente care au un rol în aplicație sunt obiectul cu structura pe care o conține un fișier XML, clasa care asignează ID-uri echipamentelor din rețea, IDGenerator și clasa FlowPusher care trimite flux-urile controller-ului.

Între clasele care alcătuiesc aplicația exista dependențe care trebuie luate în considerare pentru a preveni aducerea rețelei configurate într-o stare inconsistentă.

Următorul tabel conține pe prima coloana o acțiune care se efectuează asupra unui echipament sau flux, iar cea de a doua coloană conține consecințele pe care acțiunea o are asupra altor componente din topologie. În prealabil definim următoarea listă numerotată de consecințe:

1. Ștergerea flux-urilor care trec prin acel link
2. Actualizarea lățimii de bandă a link-urilor
3. Colorarea link-urilor care fac parte din rută
4. Resetarea culorii link-urilor care fac parte din rută
5. Codificarea echipamentelor din rețea într-o matrice de adiacență și costuri
6. Verificarea conexității
7. Calcularea rutei cele mai scurte
8. Eliberarea tuturor ID-urilor
9. Ștergerea tuturor flux-urilor
10. Ocuparea portului
11. Redimensionarea zonei de configurație

Acțiune	**Consecințe**
Ștergerea unui link	1, 2
Ștergerea unui flux	2
Editarea unui flux	2, 3, 5, 6
Crearea unui flux	2, 3, 5, 6
Ștergerea configurației	8, 9
Deconectarea unui link	1, 2
Click pe zona de configurare	4

Conectarea unui link	10
Adăugarea unui echipament de rețea	11

Figura 2.3

2.2.2. INTERFAȚA GRAFICĂ

În cazul acestei aplicații interacțiunea dintre interfața grafică și utilizator în mare majoritate se realizează prin intermediul tastaturii și al mouse-ului, cu care un desktop sau un laptop este dotat. În principiu interfața utilizator se bazează mai mult pe funcționalități, are ca scop să ofere o vizualizare a topologiei și să ușureze configurarea rețelei. Elementul cel mai complex necesar, suprafața de configurare și desenare a topologiei, va avea rolul de a asigura o suprafață grafică pe care echipamentele de rețea și legăturile dintre acestea să fie stocate, fie prin adăugarea acestora prin drag&drop din lista cu echipamente și conectarea lor cu link-uri , fie prin deschiderea unei configurații salvate anterior într-un fișier sau importarea unei configurații din Mininet dintr-un fișier. Funcționalitățile acestuia sunt în totalitate construite cu elemente grafice oferite de WPF, iar logica și interacțiunea dintre acestea au fost proiectate de către mine.

În principiu, utilizatorul când pornește aplicația va avea în față o fereastră care conține în partea de sus un meniu. Acesta conține funcționalități precum crearea unei noi configurații, deschiderea, salvarea și importarea uneia, setarea adresei controller-ului Floodlight, trimiterea rutelor către controller pentru a fi aplicate în rețea, afișarea listei cu toate rutele configurate pana acum, afișarea/ascunderea listei de echipamente de rețea și deschiderea manualului utilizatorului. Majoritatea acestor funcționalități din meniu având comenzi rapide din tastatură. Urmează o listă cu echipamentele de rețea disponibile spre a fi folosite în alcătuirea topologiei, zona de configurare și desenare a topologiei, care este asemenea unei hărți pe care poți derula vertical și lateral, mări sau micșora, muta echipamentele dintr-un loc în altul, conecta/deconecta legăturile dintre acestea, configura fiecare echipament în parte și desigur, scopul principal al aplicației, stabilirea rutei pachetelor care circulă între gazde. În afara acestor funcționalități legate de zona de configurare, aplicația mai conține elemente grafice precum butoane, un slider, texte de notificare a proceselor aflate în curs de desfășurare.

2.2.3. CITIREA CONFIGURAȚIEI

O parte importantă a acestei aplicații consta în importarea configurațiilor, fie din Mininet, fie dintr-un fișier XML. În acest proces este important ca datele obținute să fie punctuale, altfel, o alterare a fișierului de importat ar putea rezulta cu eșec.

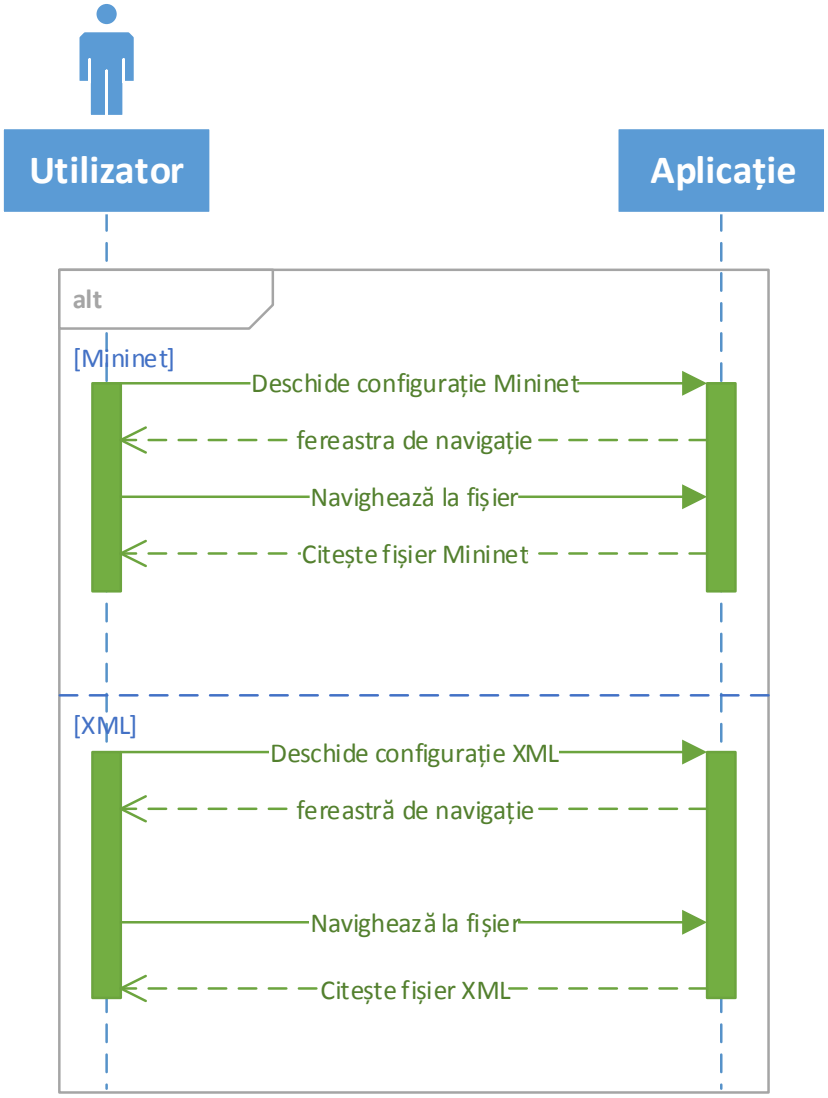

Figura 2.4

Importarea unui fișier din Mininet constă în pornirea aplicației Mininet, și executarea comenzi *net*, după care se copiază rezultatul comenzi și se salvează într-un fișier text. Când se pornește aplicația pentru configurarea rețelei, din meniu se selectează opțiunea de a importa un fișier din Mininet și se navighează până la calea unde se află fișierul anterior salvat. Când structura fișierul este construita în aplicație, echipamentele se afișează, fiecare categorie, pe o coloană, fiind necesară aranjarea acestora pentru o imagine mai clară asupra

topologiei. De vreme ce fișierul conține doar tipurile de echipamente și legăturile dintre acestea, trebuie stabilite adresele IP ale gazdelor și ID-urile switch-urilor conform rețelei din Mininet, altfel flux-urile stabilite nu vor avea efect când sunt trimise controller-ului pentru a fi aplicate în rețeaua din Mininet.

Importarea unei configurații salvate într-un fișier XML este un proces mai scurt deoarece tot ce trebuie făcut este ca din meniu să se selecteze opțiunea de a deschide o configurație anterior sălvată într-un fișier XML, să navigheză în sistemul de fișiere la calea unde se află fișierul dorit a fi deschis. Odată deschis, acest fișier conține exact structura și setările care au fost la momentul salvării.

2.2.4. SALVAREA CONFIGURAȚIEI

Salvarea configurației este o trăsătură importantă a aplicației deoarece permite utilizatorului să își întrerupă munca depusă pana la un moment dat și să revină atunci când el dorește.
Spre deosebire de citirea configurației, salvarea se poate face într-un singur format, XML.
Meniul din interfața grafică oferă două opțiuni pentru salvarea configurației, “Save” și “Save As…”. Prima opțiune este disponibila atunci când zona de configurare nu este goală și s-au produs schimbări de la ultima salvare. O schimbare ar putea însemna adăugarea, ștergerea unui flux, echipament de rețea sau link. După prima salvare a configurației, cele ulterioare efectuate cu opțiunea “Save” se vor face tot în fișierul selectat întâia oară. A doua opțiune “Save As...” cere de fiecare dată utilizatorului să selecteze fișierul în care să salveze configurația.
Fișierul XML conține echipamentele de rețea, link-urile, dimensiunea zonei de configurare, nivelul de zoom și fluxurile stabilite. Fiecare dintre aceste obiecte are salvate în fișier doar acele atribute care sunt îndeajuns pentru reconstruirea topologiei atunci când se citește fișierul. Alterarea fișierului XML salvat în ceea ce privește vreun element ar duce la o eroare de citire. Alterarea datelor conținute de un element ar putea aduce topologia într-o stare inconsistentă. Structura fișierului în care se salvează configurația este descrisă detaliat în capitolul 3.

2.2.5. CONFIGURAREA REȚELEI

Partea cea mai importantă în ceea ce privește funcționalitatea aplicației, ar fi mecanismul prin care informațiile achiziționate din fișierul importat sunt ulterior făcute disponibile

utilizatorului spre a le modifica în continuare. Această fază constă din parcurgerea următorilor pași:

1. Citirea datelor din fișierele XML sau Mininet.
2. Folosirea acestor date pentru construirea topologiei și setarea configurației echipamentelor.
3. Afișarea topologiei pe o hartă.
4. Adăugarea, ștergerea, editarea, stabilirea fluxurilor, conectarea și deconectarea echipamentelor de rețea.
5. Aplicarea unui algoritmi pentru determinarea conexității a două gazde și unul pentru determinarea rutei celei mai scurte între două gazde luând în considerare rutele deja stabilite.

După importarea unui fișier trebuie setate adresele IP ale gazdelor și ID-urile switch-urilor dacă pana acum încă nu s-a făcut asta.

2.2.6. STABILIREA FLUXURILOR

Într-o rețea pentru a stabili o conexiune între două gazde este necesar a se obține câteva informații despre acestea, care trebuie puse în antetele PDU-urilor din stiva de protocoale TCP/IP. De exemplu, antetul cadrului nivelului legătură de date cere adresă MAC a destinatarului, informație care încă nu este cunoscută. Pentru a o obține se trimite prin difuzare un mesaj ARP în rețea, care cere tuturor gazdelor să răspundă mesajului ARP cu adresă lor MAC, dacă adresă IP destinație din mesajul ARP recepționat se potrivește cu a lor. Astfel după ce s-a obținut adresă MAC a destinatarului, sursă poate completa antetul cadrului cu informația lipsă și poate trimite pachetul nivelului fizic. Următoarea figura ilustrează cum are loc acest proces.

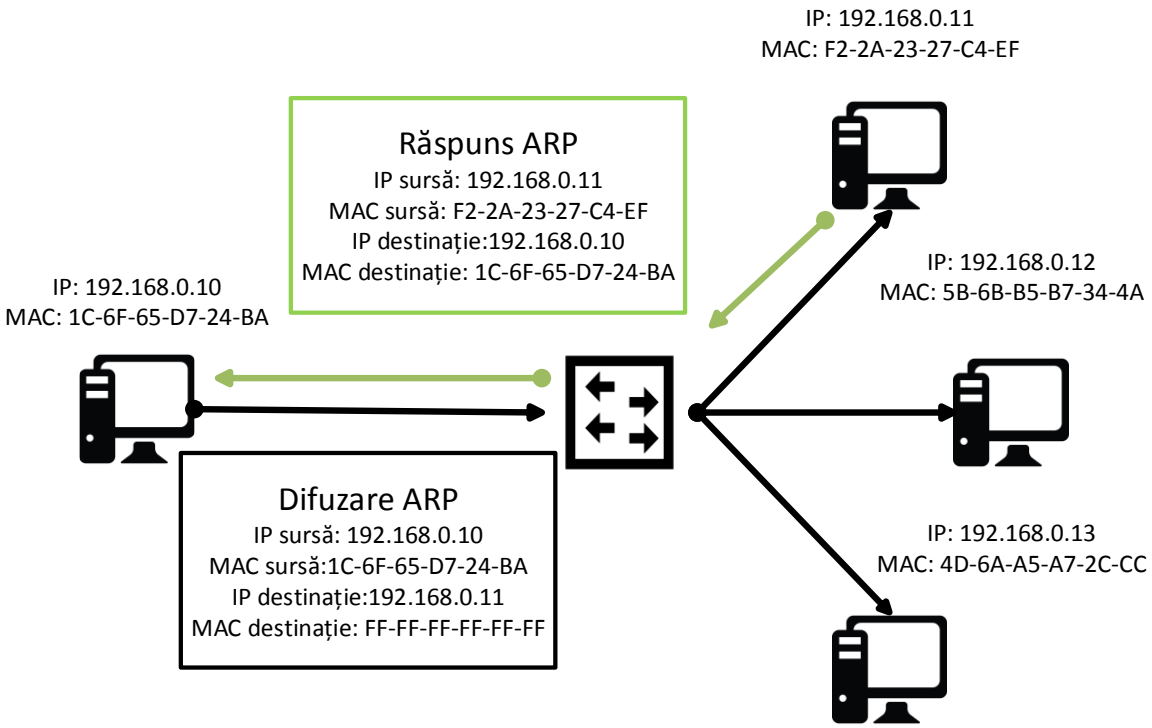

Figura 2.5 Difuzarea ARP

Pentru mai multe informații despre ARP citiți din [6].

Fluxurile, după ce au fost stabilite în aplicație, o listă cu ele este trimisă controller-ului.

Un flux conține o listă de elemente care stochează informații privind configurarea fiecărui switch care face parte din ruta parcursă de flux. Pentru fiecare switch care face parte dintr-un flux, trebuie setat, pentru traficul ARP, portul de intrare și ieșire pentru o anumită adresă IP destinație. Analog pentru traficul IP, în plus adăugăm adresă IP sursă. Un element referitor traficului ARP conține informații precum ID-ul global al elementului din lista fluxului, ID-ul switch-ului pentru care trebuie aplicat, portul de intrare, ieșire și adresă IP destinație. Deoarece fiecare element din lista fiecărui flux are un ID global, asta înseamnă ca nu trebuie să existe două elemente, în lista aceluiași flux sau în listele a două fluxuri diferite, cu ID-uri identice. Dacă se inserează un element cu un ID care deja există, asta ar rezulta în eliminarea celui existent.

Deoarece fluxurile pot fi bidirecționale, acestea trebuie setate și în sens invers. Acest lucru se realizează prin adăugarea unui element în lista fluxului, în care inversăm portului de ieșire cu cel de intrare, adresă IP sursă cu cea destinație și pentru a da un ID unic acestuia, luăm ID-ul elementului pe care l-am inversat pentru a-l crea pe acesta și adăugăm un cuvânt, de exemplu “invers”, astfel păstrând unicitatea ID-urilor elementelor.

Din cele descrise anterior observăm că pentru fiecare trafic ARP creăm unul IP.

Stabilirea rutei unui flux este un subiect discutat în cele ce urmează.

2.3. ALGORITMI DE DETERMINARE A RUTELOR

În aplicație, fiind vorba despre rețele, putem observa cu ușurință faptul că rețeaua ar putea fi reprezentată de un graf. O întrebare care totuși se ridică este: Ce element din rețea ar trebui să reprezinte nodurile grafului? Echipamentul în șine, sau porturile acestuia? Este evident că link-urile vor fi muchiile dintre noduri. În primă instanță pare mai simplu dacă am alege echipamentul să reprezinte nodul. Chiar dacă nodurile sunt echipamentele, putem determina porturile prin care trece o rută. Cum? Memorând în fiecare link, pentru fiecare din cele două capete ale sale, echipamentele la care este conectat, respectiv porturile acestora. Totuși această soluție nu mai este valabilă atunci când între două echipamente, cum ar fi două switch-uri, există mai mult de o conexiune directă, de exemplu, imaginați-vă un graf neorientat cu două noduri și două sau mai multe muchii între aceste două noduri, vedeți figura 3.6. Așadar, în final, ajungem să construim graful astfel încât nodurile să fie reprezentate de porturile echipamentelor. În graf, porturile unui echipament care are mai mult de un port, formează un subgraf complet, pentru a reprezenta faptul că acestea, în interiorul echipamentului sunt interconectate.

2.3.1. PARCURGEREA ÎN LĂȚIME

Această aplicație conține două probleme din teoria grafurilor despre care se poate spune că rezolvarea lor ar fi reprezentat o provocare care ar fi depășit cu mult efortul depus pentru dezvoltarea acestei aplicații, în cazul în care acestea nu aveau soluții în prezent.

Una dintre probleme constă în determinarea faptului dacă există un drum între două noduri dintr-un graf neorientat. Această problemă se rezolvă prin parcurgerea, fie în lățime sau adâncime, a grafului începând cu unul dintre cele două noduri și îl numim nodul sursă. După parcurgere, dacă nodul destinație se află în șirul nodurilor parcurse, atunci asta înseamnă că un drum există între cele două noduri, în caz contrar nu există.

Algoritmul descris în pseudocod este următorul:

1. Creăm două șiruri, unul pentru nodurile de parcurs și celălalt pentru nodurile parcurse, inițial acestea sunt goale.
2. Adăugăm nodul sursă la șirul nodurilor de parcurs.
3. Cat timp șirul nodurilor de parcurs nu este gol facem următoarele:
    1. Mutăm primul element din șirul nodurilor de parcurs în șirul nodurilor parcurse.

2. Pentru fiecare nod vecin al nodului mutat la pasul anterior facem următoarele:
    1. Dacă nodul vecin este nodul destinație atunci returnăm succes, cele două noduri au un drum între ele.
    2. Dacă nodul vecin nu este în șirul nodurilor parcurse atunci îl adăugăm în șirul nodurilor de parcurs
4. Returnează eșec, un drum nu există între cele două noduri.

Acest algoritm este folosit în aplicație pentru a da o experiență plăcută utilizatorului. Rolul acestuia este de a determina dacă, între cele două gazde între care se dorește să se stabilească un flux, există un drum, neluând în considerare lățimile de bandă disponibile ale link-urilor. Spus altfel, dorim să aflăm dacă cele două noduri se află în aceeași componentă conexă din graf. Motivul pentru care dorim să aflăm acest lucru este ca să îl scutim pe utilizator de lucru în plus. Dacă continuam fără acest pas, utilizatorul ar fi continuat la fereastra de configurare a fluxului, unde ar seta protocolul și throughput-ul dorit, și ar declanșa butonul pentru calculul rutei cele mai scurte dintre cele două gazde luând în considerare lățimile de bandă rămase disponibile după stabilirea fluxurilor anterioare. În caz că utilizatorul nu a observat că un drum nu există între cele două gazde, noi dorim să îl scutim de acest lucru în plus. Pentru mai multe informații despre acest algoritm, citiți capitolul *22.2 Breadth First Search* din [2].

2.3.2. ALGORITMUL LUI DIJKSTRA

În aplicație, acest algoritm este folosit pentru a determina calea cea mai scurtă dintre două gazde, luând în considerare lățimea de bandă disponibilă a linkurilor. Matricea costurilor trebuie construită astfel încât să conțină lățimea de bandă disponibilă a linkurilor. Singura condiție pe care o adăugăm este ca costul dintre două noduri să fie mai mare sau egal decât throughtput-ul cerut de utilizator pe acea rută.

Algoritmul descris în pseudocod este următorul:

1. Creăm un șir care conține distanta cea mai scurta dintre nodul sursă și oricare alt nod.
2. Creăm un șir care conține mulțimea nodurilor care deja parcurse.
3. Creăm un șir care conține părinții nodurilor.
4. Inițial toate nodurile au distanța infinit fată de nodul sursă, nodurile nu au părinți și niciun nod nu a fost parcurs.
5. Distanța de la sursă la sursă este 0, iar sursă nu are părinte.
6. Pentru toate nodurile din graf facem următoarele:

6.1. Luăm nodul cu distanta cea mai mică față de sursă

6.2. Marcam nodul ca și parcurs.

6.3. Pentru fiecare vecin al nodului facem următoarele:

6.3.1. Dacă nodul vecin nu este parcurs, exista legătura între nod și vecinul sau, throughput-ul dintre nod și vecinul sau este mai mare sau egal decât throughtput-ul cerut și distanța de la nod la sursă, însumată cu distanța la vecinul sau este mai mică decât distanta actuala a vecinului la sursă atunci:

6.3.1.1. Părintele nodului vecin devine nodul curent

6.3.1.2. Distanța la nodul vecin devine distanta pana la nodul curent, plus distanța dintre nodul curent și vecin

7. Dacă distanta dintre nodul sursă și destinație este infinit atunci asta înseamnă ca nu se poate stabili un flux între cele două gazde, altfel este posibil, iar ruta o aflam din șirul care conține părinții nodurilor.

Pentru mai multe informații despre acest algoritm citiți capitolul *24.3 Dijkstra's algorithm* din [2].

Cap.3. ASPECTE DE IMPLEMENTARE

Aplicația a fost dezvoltata în Microsoft Visual Studio Community 2017, o versiune gratuită. Acesta este folosit pentru a dezvolta software pentru Microsoft Windows, precum și site-uri Web, aplicații web, servicii web și aplicații mobile. Visual Studio utilizează platforme de dezvoltare software cum ar fi Windows API, Windows Forms, Windows Presentation Foundation, Windows Store și Microsoft Silverlight. Oferă o mulțime de facilitați dezvoltatorului precum: un editor de cod care suportă IntelliSense, componenta de completare a codului, precum și formatarea codului, code profiler(instrument care măsoară și analizează memoria, frecventa instrucțiunilor, are rolul de a optimiza), designer pentru construirea de aplicații cu GUI, web designer, designer de clasă și designer de schemă de baze de date. inclusiv adăugarea de suport pentru sistemele de control sursă, precum Subversion și GitHub.

Aplicația este construita cu WPF, acesta este un subsistem grafic de la Microsoft pentru redarea interfețelor utilizator în aplicațiile bazate pe Windows. În loc să se bazeze pe sistemul GDI mai vechi, WPF utilizează DirectX. WPF încearcă să furnizeze un model de programare consistent pentru aplicațiile de construire și separă interfața utilizator de logica

manipulării datelor. Utilizează XAML, un limbaj bazat pe XML, pentru a defini și a lega diferite elemente de interfață.

Figura următoare prezinta prin diagrama de clasă, toate clasele care fac parte din aplicație. Am ales să le scriu doar numele și să le prezint mai în detaliu în cele ce urmează pentru că altfel ar fi ocupat un spațiu mare care ar doar ar fi îngreunat înțelegerea diagramei.

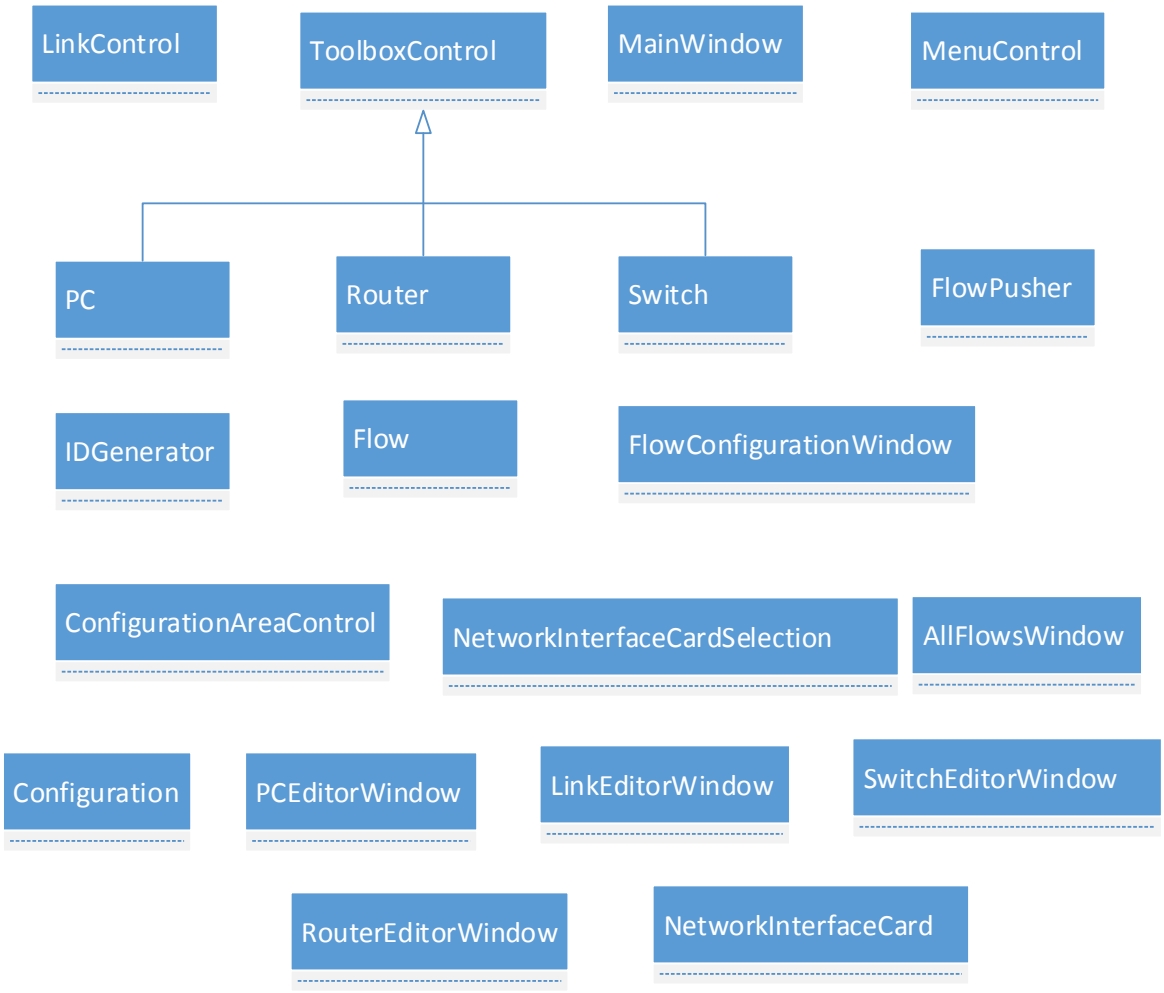

Figura 3.1 Diagrama de clasă

3.1. MENIUL

Meniul este plasat în partea de sus a ferestrei principale și are următoarea structura:

- File
  - New
  - Open
    - XML
    - Mininet
  - Save

    - Save As…
    - Exit
- Flows
    - Controller address
    - Push flows
- View
    - Toolbox
    - All flows
- Help

Opțiunile New, Open, Save, Save As…, Exit când sunt declanșate cheamă comenzi definite în fereastra principala, MainWindow.xaml.

Optiunea New, chema comanda cu același nume. Aceasta poate fi chemata atunci când numărul de elemente din zona de configurare, ConfigurationAreaControl.xaml, este mai mare decât zero. In cazul în care comanda este activa și este executata atunci se verifica dacă exista schimbări în configurare de la ultima salvare, dacă exista atunci utilizatorul este întrebat dacă dorește să salveze configurația. Acesta poate răspunde *da, nu* sau *anulează*.

Opțiunea Open cheamă comanda Open, când aceasta este chemata primește un parametru care specifică tipul de fișier care se dorește a fi citit, XML sau Mininet. Înainte de a se selecta fișierul de citit, dacă s-a lucrat, utilizatorul este întrebat dacă dorește să salveze.

Opțiunea Save apelează comanda cu același nume, este activa doar dacă s-au efectuat schimbări în zona de configurare de la crearea unei noi configurații sau de la ultima salvare. După ce s-a efectuat o salvare, aceasta comanda va efectua salvările ulterioare în fișierul în care s-a scris configurația ultima oara.

Comanda Save As... este activa atunci când numărul de elemente din zona de configurare este mai mare decât zero. Utilizatorul trebuie să selecteze de fiecare data fișierul unde vrea să salveze.

Opțiunea Exit cheamă comanda Close. Aceasta executa metoda de închidere a aplicației, care este suprascrisă pentru a putea oferi ocazia utilizatorului de a salva înainte ca aplicația să se închidă.

Metodele care sunt apelate în comenzile enumerate anterior sunt descrise în subcapitolele *Citirea fișierelor* și *Scrierea fișierelor.*

Elementul *Flows* din meniu oferă posibilitatea de a seta adresă IP a controller-ului Floodlight. Pentru a obține aceasta adresă trebuie să aflați adresă IP a sistemului de calcul

pe care se executa controller-ul. De asemenea trebuie știu și portul pe care lista cu fluxuri poate fi trimisă. Trebuie introdus un text de forma “http://192.168.0.102:8080”. După ce acest câmp este setat, se poate executa opțiunea *Push flows*, care trimite fluxurile stabilite controller-ului.

Elementul *View* din meniu oferă posibilitatea de a ascunde și a arata lista de echipamente, și de a vedea toate fluxurile stabilite pana acum într-o fereastra care se deschide dacă exista fluxuri.

Elementul *Help* deschide un fișier .chm, în care este descrisă aplicația și componentele acesteia. Aceasta opțiune se poate apela apăsând tasta F1.

Majoritatea elementelor din meniu au comenzi rapide de la tastatură, acestea sunt afișate lângă element.

Codul de mai jos descrie cum este creat meniul în XAML și cum se leagă cu atributul *Command* o comanda de un element din meniu.

```
<Menu Background="AliceBlue" Grid.Row="0">
        <MenuItem Header="_File">
            <MenuItem Header="_New" Command="New"/>
            <MenuItem Header="_Open">
                <MenuItem Name="MininetMenuItem" Header="_Mininet"
                          Command="Open" CommandParameter="Mininet"
                          InputGestureText="Ctrl+Shift+O"/>
                <MenuItem Name="XMLMenuItem" Header="_XML"
                          Command="Open" CommandParameter="XML"
                          InputGestureText="Ctrl+O"/>
            </MenuItem>
            <MenuItem Header="_Save" Command="Save"/>
            <MenuItem Header="_Save As..." Command="SaveAs"/>
            <MenuItem Header="_Exit" Command="Close"/>
        </MenuItem>
        <MenuItem Header="_Flows">
            <MenuItem Header="Controller IP"
                      Height="50">
                <MenuItem.HeaderTemplate>
                    <ItemContainerTemplate>
                        <Grid Width="150">
                            <Grid.RowDefinitions>
                                <RowDefinition Height="*"/>
                                <RowDefinition Height="*"/>
                            </Grid.RowDefinitions>

                            <TextBlock Text="Controller IP"/>
                            <TextBox Grid.Row="1"
                                     TextChanged="TextBox_TextChanged"/>
                        </Grid>
                    </ItemContainerTemplate>
                </MenuItem.HeaderTemplate>
            </MenuItem>
            <MenuItem Header="_Push flows" Click="PushFlows_Click"/>
        </MenuItem>
        <MenuItem Header="_View">
            <MenuItem Header="_Toolbox" IsCheckable="True"
                      IsChecked="True" Click="ToolboxMenuItem_Click"/>
```

```
            <MenuItem Header="_All Flows" Click="AllFlows_Click"/>
        </MenuItem>
        <MenuItem Header="_Help" Command="Open" CommandParameter="Help"
                  InputGestureText="F1"/>
    </Menu>
```

În codul de mai sus se observa ca comanda Open este apelata de două ori, pentru XML și pentru Mininet. Cu *CommandParameter* specificam parametrul care dorim să îl trimitem comenzii. Fiecare comanda are în mod implicit comenzi rapide din tastatura, a se observa comenzile Save și Save As..., dar dacă dorim să apelăm aceeași comanda cu comenzi rapide din tastatura diferite trebuie să le redefinim. Aceasta se face introducând următorul cod în fereastra principala, MainWindow.xaml:

```
<Window.InputBindings>
        <KeyBinding Command="ApplicationCommands.Open"
              Gesture="CTRL+Shift+O" CommandParameter="Mininet"/>
        <KeyBinding Command="ApplicationCommands.Open"
              Gesture="CTRL+O" CommandParameter="XML"/>
        <KeyBinding Command="ApplicationCommands.Open"
              Gesture="F1" CommandParameter="Help"/>
</Window.InputBindings>
```

Cu acest cod redefinim comenzile rapid din tastatura pentru comanda Open, astfel încât atunci când se apasă CTRL+Shift+O să se deschidă un fișier Mininet, CTRL+O să se deschidă un fișier XML și când se apasă F1 să se deschidă manualul utilizatorului.

O comanda diferă de un eveniment prin faptul ca evenimentul poate să apară în mai multe locuri și fiecare dintre acestea poate executa metode diferite. Fiecare dintre evenimente are o metoda care se apelează când evenimentul este declanșat. Aceasta metoda poate fi atașată și de alte evenimente de același tip. O comanda poate avea o singura metoda care să se apeleze atunci când se executa. O comanda poate fi chemata din diferite locuri, dar totdeauna va chema aceeași metoda. De aceea se trimit parametrii comenzilor, pentru a putea ști locul de unde s-a apelat comanda. Pentru a afla mai multe despre meniuri citiți din [4] capitolul 10.

3.2. ZONA DE CONFIGURARE

Zona de configurare este alcătuită din lista de echipamente de rețea, iar sub aceasta se afla harta, suprafața pe care este construita topologia rețelei.

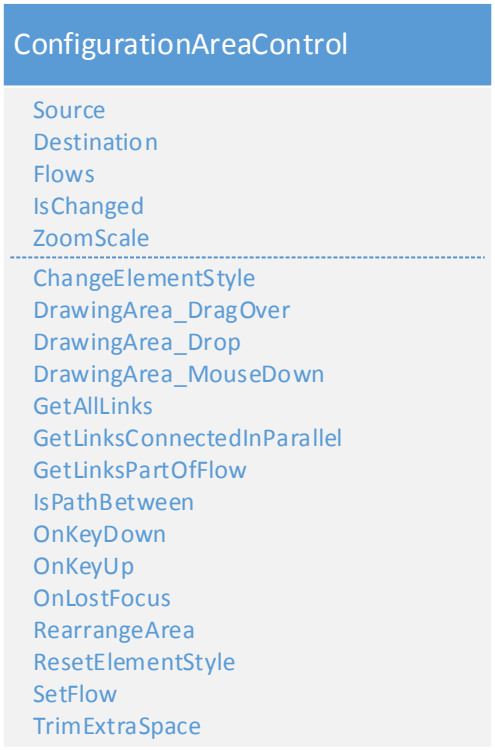

Figura 3.2 clasa ConfigurationAreaControl

Din lista de echipamente, prin procedeul de drag&drop, se pot pune echipamente pe suprafața, cu cele două TextBox-uri dedesubt se poate schimba dimensiunea suprafeței, iar cu ScrolBar-ul din dreapta se schimba zoom-ul. Suprafața este un UserControl numit ConfigurationAreaControl. Un UserControl permite utilizatorului de a crea un element nou, personalizat. ConfigurationAreaControl conține un Canvas care implementează evenimentele DragOver și Drop pentru a putea recepționa date în cazul în care ceva este aruncat pe el.

Metoda Drop face următoarele:

- Verifica dacă evenimentul a fost deja tratat.
- Extrage datele din drop.
- Dacă este un PC, Switch, Link sau Router atunci verificăm dacă părintele acestuia este suprafața, dacă este atunci trebuie doar să îl mutam, altfel trebuie să îl cream și să îl adăugăm la suprafața și să ajustăm dimensiunea suprafeței.
- Dacă este unul dintre capetele unui link atunci trebuie să mutam acel capăt al linkului.
- Dacă este un link de pe suprafața atunci îl mutam
- marcam faptul ca s-a schimbat configurația în variabila IsChanged

Metoda DragOver setează efectul pe care obiectul aruncat îl va avea. In acest caz efectul va fi unul de copiere. In metoda OnGiveFeedback din ToolboxControl se verifică efectul setat în DragOver și se schimba cursorul.

Pentru a putea schimba zoom-ul trebuie să legăm un slider de proprietatea LayoutTransform a suprafeței de configurare, în acest caz reprezentată de un Canvas.

Construim slider-ul în următorul mod:

```
<Slider x:Name="uiScaleSlider"
                ToolTip="Determines the UI scale factor."
                Value="1" Minimum="0.1" Maximum="2"
                Width="200"
                MouseDoubleClick="UiScaleSlider_MouseDoubleClick"/>
```

Declaram o proprietate, numita ZoomScale, în ConfigurationAreaControl care este legata de valoarea slider-ului numit uiScaleSlider.

```
<local:ConfigurationAreaControl x:Name="configurationArea"
      Grid.Row="3"
      ZoomScale="{Binding ElementName=uiScaleSlider, Path=Value}"/>
```

In Canvas facem în urmatorul mod:

```
<Canvas.LayoutTransform>
    <ScaleTransform CenterX="0"
        CenterY="0"
        ScaleX="{Binding ElementName=configAreaCtrl, Path=ZoomScale}"
        ScaleY="{Binding ElementName=configAreaCtrl, Path=ZoomScale}"/>
</Canvas.LayoutTransform>
```

Atunci când elementele sunt puse la marginea suprafeței, aceasta se redimensionează pentru a oferi flexibilitate. Pentru a realiza aceasta funcționalitate apelam funcțiile TrimExtraSpace, care micșorează acea parte a suprafeței care nu conține niciun element la o distanta de 550px de la margine și RearrangeArea, care mărește acea parte a suprafeței care conține element la o distanta mai mica de 50px de margine. Aceste funcții le apelam de fiecare data când facem drop pe suprafața.

Pentru a putea calcula o ruta în topologie trebuie să transformam topologia într-un graf. După cum am explicat în capitolul 2.3 Algoritmi, nodurile din graf vor fi reprezentate de porturile echipamentelor. Pentru a putea recunoaște un port și echipamentul de care aparține, am creat șirul elementEncoding, acesta conține perechi de elemente de forma (“ID:port”, index nod din matrice). Rularea algoritmul pentru a găsi ruta cea mai scurta va construi un sir care contine ruta. Elementele acestui sir vor fi indecși nodurilor din matricea de adiacenta. Pentru a afla echipamentele și porturile prin care trece ne folosim de șirul encodedElements. In șirul Path din fluxul vom insera string-uri care conțin ID-urile și porturile echipamentelor. După ce am aflat ruta, coloram linkurile prin care aceasta trece.

Pentru a găsi aceste link-uri ne folosim de calea determinata anterior. Căutăm între toate link-urile, acelea care au ID-ul și portul oricărui capăt în rută.

3.3. FERESTRELE DE CONFIGURARE

Echipamentele de rețea se pot edita. Pentru a realiza asta se face dublu click pe echipament, după care apare o fereastra în care se pot modifica proprietățile specifice acelui echipament.
Calculatorului se poate edita:

- numele
- adresă IP și MAC

Switch-ului se poate edita:

- numele
- ID-ul din Mininet
- numărul de porturi.

Router-ului se poate edita:

- numele
- numărul de placi de rețea
- adresă IP și MAC a fiecărei placi de rețea

Ferestrele acestea au și reguli de verificare a validității câmpurilor, precum adresă IP și MAC. Pentru a realiza asta sau folosit expresii regulate. Expresia regulata a unei adrese IP este: `"(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)"`, iar expresia regulată a adresei MAC este: `"([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})"`
Când un link este conectat de un Router, o fereastra apare cu un ComboBox care conține porturile libere, în aceasta se selectează portul de care dorim să conectăm linkul.
In meniu, sub View, optiunea All Flows deschide o fereastra în care se listează toate fluxurile stabilite. Pentru a realiza acest lucru am creat DataTemplate pentru elementele listei.

```
        <ItemsControl x:Name="flowsListBox"
                      ItemsSource="{Binding}">
            <ItemsControl.ItemTemplate>
                <DataTemplate>
                            <!--Protocol-->
                            <TextBlock Grid.Row="0" Grid.Column="0"
Text="Protocol:"
                               TextAlignment="Right"
                               Margin="0, 0 , 5, 0"/>
                            <TextBlock Text="{Binding Protocol}"
```

```
                    Grid.Row="0" Grid.Column="1"/>

                <!--Source-->
                <TextBlock Grid.Row="1" Grid.Column="0" Text="Source:"
                    TextAlignment="Right"
                    Margin="0, 0 , 5, 0"/>
                <TextBlock Text="{Binding Source.ControlName}"
                    Grid.Row="1" Grid.Column="1"/>
                <!--Destination-->
                <TextBlock Grid.Row="2" Grid.Column="0"
                    Text="Destination:" TextAlignment="Right"
                    Margin="0, 0 , 5, 0"/>
                <TextBlock Text="{Binding Destination.ControlName}"
                    Grid.Row="2" Grid.Column="1"/>
                <!--Throughput-->
                <TextBlock Grid.Row="3" Grid.Column="0"
                    Text="Throughput(Mb/s):"
                    TextAlignment="Right"
                    Margin="0, 0 , 5, 0"/>
                <TextBlock Text="{Binding Throughput}"
                    Grid.Row="3" Grid.Column="1"/>
                <!— Restul -->
            </DataTemplate>
             </ItemsControl.ItemTemplate>
        </ItemsControl>
```

In logica din spatele acestei ferestre legăm fluxurile de lista.

```
flowsListBox.DataContext = Flows;
```

Unde flowsListBox este numele listei, DataContext este o proprietate în care ținem datele, în acest caz lista Flows.

Un flux este modelat de clasa Flow

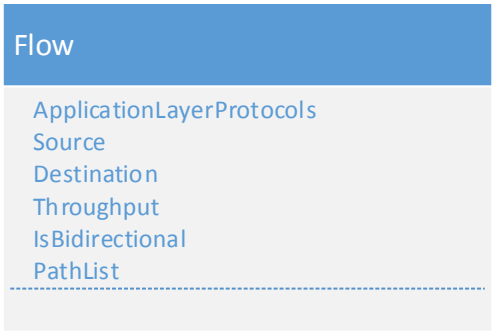

Figura 3.3 clasa Flow

In lista flowsListBox, pentru fiecare element din lista Flows se va instanția un element. Pentru a lega proprietățile unui obiect Flow de un element al listei ne folosim de clasa Binding oferită de WPF, spre exemplu:

```
<TextBlock Text="{Binding Protocol}"
```

Așa legăm proprietatea Text a unui TextBlock de proprietatea Protocol a unui obiect Flow.

Elementele din lista cu toate fluxurile mai conțin două butoane, Remove și Edit. Cu butonul Remove ștergem acel flux.

```
private void RemoveButton_Click(object sender, RoutedEventArgs e)
        {
            Button b = sender as Button;
            Flow comm = b.DataContext as Flow;

            var linksPartOfFlow =
mainWindow.configurationArea.GetLinksPartOfFlow(comm);
            foreach (var link in linksPartOfFlow)
            {
                link.UsedBandwidth -= comm.Throughput;
            }
            Flows.Remove(comm);
            flowsListBox.ItemsSource = null;
            flowsListBox.ItemsSource = Flows;
            if (Flows.Count == 0)
                Close();
        }
```

Din proprietatea DataContext a butonului aflam pentru care obiect Flow a fost butonul creat. Ștergem obiectul Flow din lista Flows, actualizam lățimile de banda a linkurilor care făceau parte din flux, reîmprospătam lista din fereastra prin asignarea referinței null și după aceea asignând din nou lista Flows. In mod normal nu așa se procedează. Pentru a reîmprospăta o lista din interfața grafică, în mod obișnuit folosim clasa ObservableCollection pentru lista cu datele, în acest caz lista cu datele, Flows este de tipul List<Flow>. Astfel nu ar mai trebui să asignam null și după aceea din nou lista cu datele pentru a se efectua reîmprospătarea listei din interfața grafică. Când am ajuns la aceasta problema am ales să păstrez tipul listei Flow să rămână List<Flow> pentru că clasa List oferă funcționalități de căutare pe care ObservableCollection nu le are și ar fi trebuie să scriu eu acele funcții de căutare.
Următoarea diagrama de clasă prezintă toate aceste ferestre:

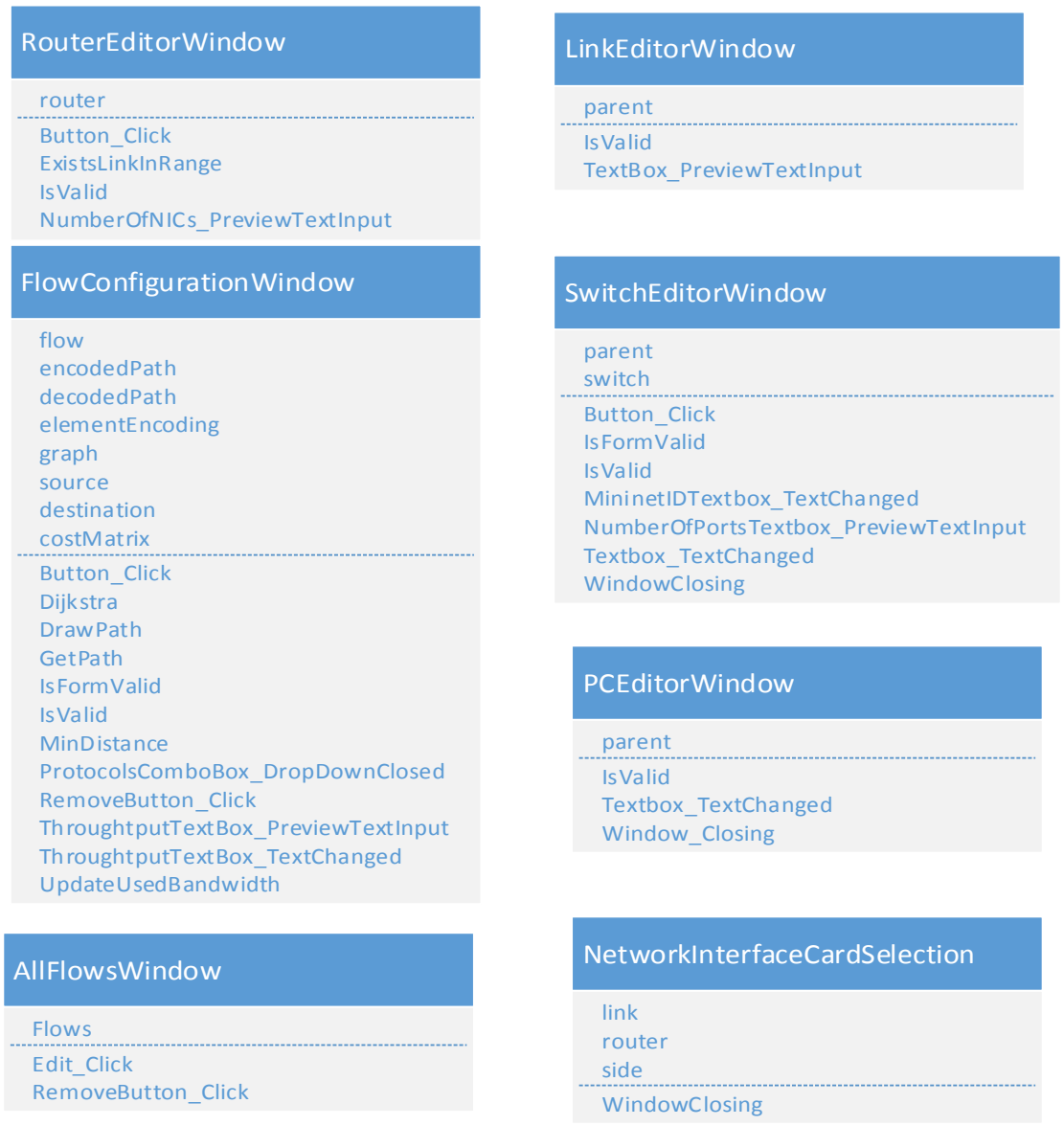

Figura 3.4 Ferestrele de configurare ale echipamentelor

3.4. ECHIPAMENTE DE REȚEA

Gazdele, switch-urile și routerele au proprietăți comune cum ar fi ID-ul, numele, sursă imagini, coordonatele, astfel că ele pot fi tratate în același mod pana la un anumit punct. De aceea am creat clasa ToolboxControl. Manipularea acestora pentru acțiunile de mutare, conectare, ștergere și adăugare pot fi tratate de către o singură operațiune pentru fiecare dintre acțiuni . Diferențele apar atunci când trebuie să selectam portul la care să conectam linkul. Conectarea poate tratata la fel pentru toate echipamentele, acea parte în care legăm coordonatele capătului linkului de mijlocul echipamentului, însă în continuare trebuie să verificăm tipul echipamentului.

- Dacă este un PC atunci trebuie să conectam linkul de placa de rețea a acestuia
- Dacă este un switch atunci trebuie să conectam linkul de primul port liber
- Dacă este un router atunci rugam utilizatorul să selecteze placa de rețea

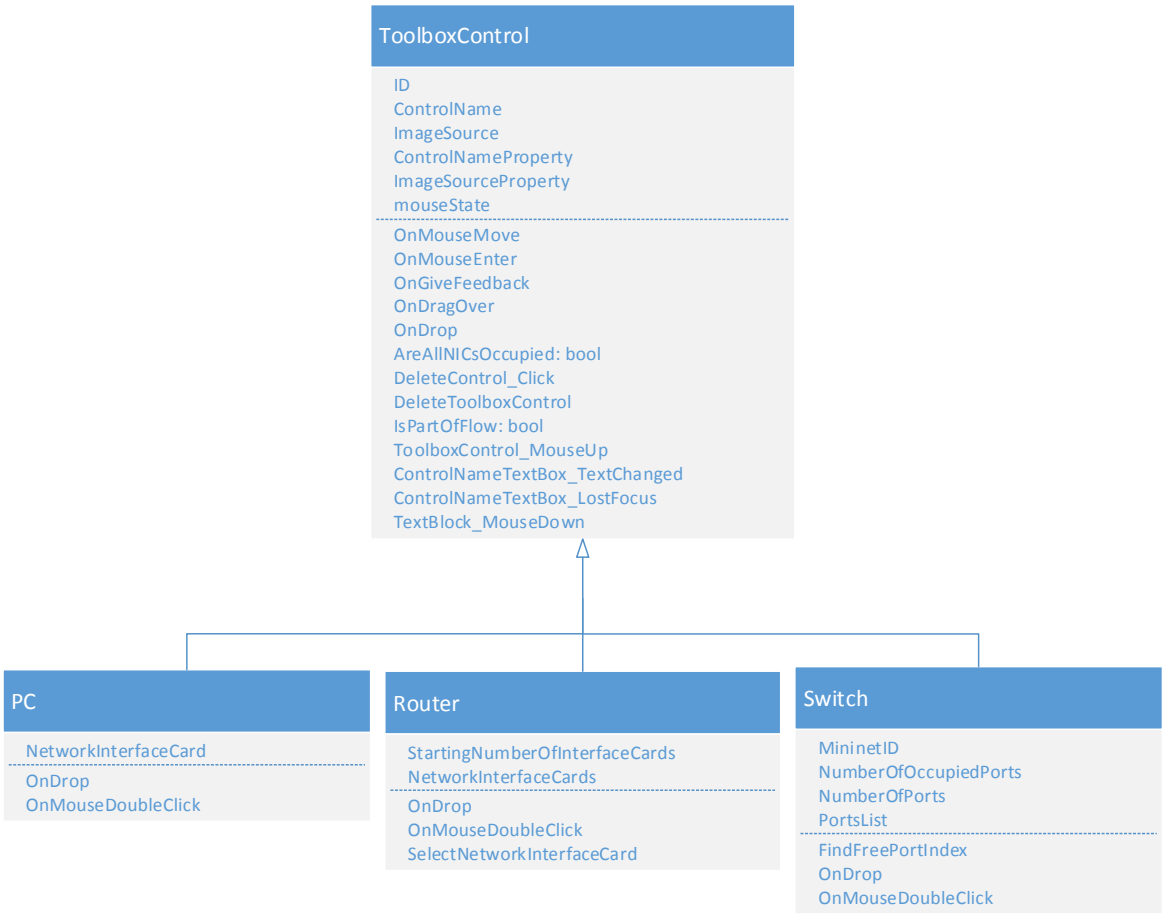

Figura 3.5 Echipamentele de rețea

Echipamentele de rețea moștenesc toate clasa ToolboxControl, care la rândul ei moștenește clasa UserControl. Aceasta din urma permite crearea elementelor personalizate, complexe. Un ToolboxControl are următoarea forma:

```
<UserControl x:Class="NetworkConfigurator.ToolboxControl"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:NetworkConfigurator"
             Height="60"
             Width="60"
             Name="toolboxControl" Margin="5, 0, 5, 0">
    <!-- Aici scrii elementele -->
</UserControl>
```

Pentru a edita un echipament, se face dublu click pe acesta, în fereastra care apare ii putem modifica proprietățile. Pentru a edita numele echipamentului se poate face dublu click pe

numele acestuia, iar TextBlock-ul dispare și iasă la iveala un TextBox. Când s-a schimbat numele se face click oriunde altundeva și atunci se cheamă metoda OnLostFocus care face să reapară TextBlock-ul, iar TextBox-ul să dispară.

Dacă linkuri sunt conectate la echipament, când conectam un altul verificăm dacă acesta mai are porturi sau placi de rețea libere. Evenimentul de OnDrop este tratat și în ToolboxControl și în fiecare tip particular de echipament, evenimentul este suprascris. In ToolboxControl, evenimentul OnDrop, are rolul de a lega coordonatele capătului care a fost aruncat pe echipament de echipament. In următoarele descriem metoda OnDrop din ToolboxControl:

- Luăm linkul din obiectul aruncat pe echipament
- Dacă capătul care a fost aruncat pe echipament este primul capăt atunci:
    - Legăm coordonatele primului capăt de mijlocul echipamentului
    - Marcam faptul ca acest capăt este conectat
    - Setăm în link ID-ul echipamentului la care tocmai s-a conectat
    - Dacă al doilea capăt este și el conectat atunci setăm offsetul acestui link

Offset-ul unui link reprezintă distanta față de mijlocul liniei care trece prin echipamentele la care este conectat. Dacă exista mai multe link-uri între aceleași echipamente atunci pe măsura ce adăugăm linkuri le punem tot mai îndepărtat de mijloc pentru a putea fi vizibile. Imaginea următoare ilustrează cuvintele citite anterior.

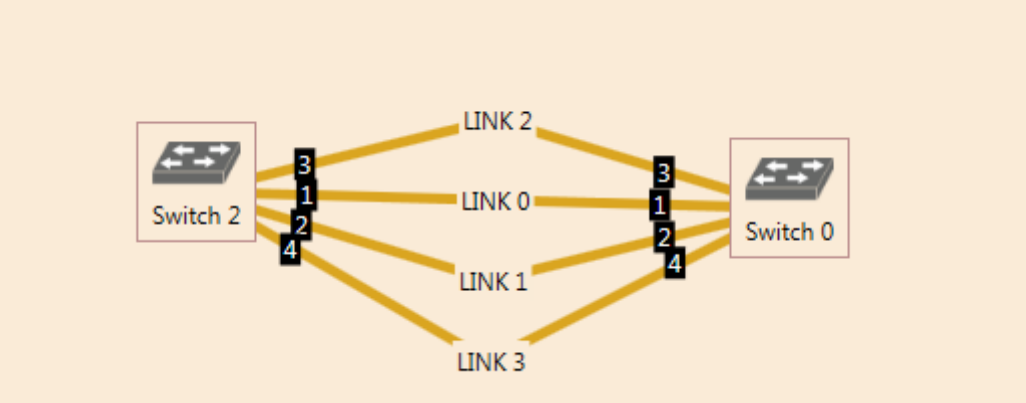

Figura 3.6 Două echipamente cu mai multe linkuri între ele.

Pentru a trage un echipament verificăm în metoda OnMouseMove dacă click-ul stâng este apăsat și nu a intrat pe suprafața echipamentului fiind apăsat. Dacă aceste condiții se îndeplinesc atunci împachetăm echipamentul într-un obiect care urmează să fie prelucrat în metoda OnDrop a elementului pe care aruncam echipamentul.

Pentru un exemplu complet de drag&drop vizitați [10], sau citiți capitolul 6 din [4].

Ștergerea unui echipament de pe suprafața de configurare se face cu click dreapta pe echipament și din meniu selectați opțiunea delete, sau click pe echipament și apăsăm tasta Delete de pe tastatura. Pentru a putea șterge apăsând tasta Delete trebuie să setăm proprietatea Focusable pe true în clasa ToolboxControl. Astfel atunci când dam click pe un echipament în metoda ToolboxControl_OnMouseUp vom seta elementul focusat. Pasul următor este ca să definim evenimentul OnKeyUp, care atunci când este declanșat, verificăm dacă tasta Delete este apăsată. Pentru a atașa un meniu de un UserControl, definim un meniu în proprietatea ContextMenu a acestuia.

```
<UserControl.ContextMenu>
    <ContextMenu>
        <MenuItem Header="Delete"
            Click="DeleteControl_Click">
        </MenuItem>
    </ContextMenu>
</UserControl.ContextMenu>
```

Când un echipament se șterge trebuie să verificăm dacă acesta face parte dintr-un flux, sau dacă are linkuri conectate. In caz ca are fluxuri care trec prin el atunci notificam utilizatorul de aceasta constatare și îl lăsăm să aleagă, să șteargă sau nu echipamentul. Dacă se șterge un echipament toate linkurile conectate se vor deconecta capetele lor se vor pune undeva într-o zona sub echipamentul șters, iar fluxurile care trec prin echipament vor fi și acestea șterse. O îmbunătățire care s-ar putea aduce aplicației este aceea de a încerca să se restabilească fluxul pe o alta ruta dacă un echipament din flux este șters. Pentru a determina la momentul ștergerii dacă un echipament face parte dintr-un flux, parcurgem toate căile din toate fluxurile și verificăm dacă în calea fluxului găsim ID-ul echipamentului.

3.5. LINKUL

Linkul are rolul de a lega echipamentele între ele. La fel ca oricare alt echipament, acesta poate fi luat din lista de echipamente și pus pe suprafața de configurare prin procedeul de drag&drop. Acolo poate fi mutat tot prin drag&drop, făcând click pe orice parte a suprafeței linkului, în afara de capete. Când se face drag&drop apucând unul dintre capete, celălalt capăt va rămâne în loc. Pentru a conecta linkul de un echipament trebuie să aruncam unul dintre capete pe echipament. În aplicație linkul este clasa LinkControl, este un UserControl și implementează interfața INotifyPropertyChanged. Aceasta interfața conține evenimentul PropertyChanged. Rolul acestui eveniment este de a notifica elementele din interfața grafică, de faptul ca o schimbare a apărut în date și trebuie să se actualizeze. De obicei datele care se leagă de interfața grafică sunt proprietățile unui obiect. Când o proprietate își

schimbă valoarea aceasta schimbare nu se reflecta pe interfața grafică. Pentru a realiza aceasta reflexie în metoda "set" a proprietăți declanșăm evenimentul PropertyChanged.

```
public string LinkName
{
    get
    {
        return linkName;
    }

    set
    {
        linkName = value;
        OnPropertyChanged();
    }
}
```

Proprietatea de mai sus în "set", cheamă metoda OnPropertyChanged, care nu este decât un wrapper al evenimentului PropertyChanged.

```
private void OnPropertyChanged([CallerMemberName]string property = "")
{
   PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(property));
}
```

Atributul "CallerMemberName" are rolul de a obține numele proprietății care a chemat metoda. In acest caz proprietatea fiind "LinkName".

Deconectarea capătului unui link are loc prin dublu click pe jumătatea din care capătul face parte. Când capătul unui link se deconectează, verificăm dacă acest link face parte dintr-un flux, în caz afirmativ notificam utilizatorul și îl lăsăm pe el să decidă dacă va continua sau va anula procesul de deconectare.

3.6. TRIMITEREA FLUXURILOR

Trimiterea fluxurilor consta în construirea unor obiecte JSON care vor avea rolul de stabili în fiecare switch dintr-un flux porturile de intrare și ieșire pentru anumite adrese IP. După cum am explicat în capitolul (scrie aici unde ai chestia cu ARP-ul), pentru a stabili un flux între două gazde trebuie să rutăm pe lângă traficul IP, și traficul ARP. In aplicația noastră ruta pe care o parcurge un flux este memorata într-un sir de string-uri, care conține elemente de forma "ID:port". Unde "ID" este ID-ul echipamentului, iar "port" este evident, portul prin care trece. Menționez ca ID nu este ID-ul din Mininet al switch-ului, este un câmp din clasa ToolboxControl, și are rolul de a identifica un echipament, oricare ar fi tipul acestuia. Pentru fiecare echipament de rutare, în cazul nostru avem doar switch-uri, care face parte din ruta, trimitem două obiecte JSON.

Pentru traficul ARP trimite următorul obiect JSON:

{

"switch": "00:00:00:00:00:00:00:01",

"name": "flow_1_ARP_01",

"cookie": "0",

"priority": "32765",

"active": "true",

"in_port": "5",

"actions": "output=3",

"arp_tpa": "10.0.0.7",

"eth_type": "0x806"

}

- “switch” – ID-ul din Mininet al switch-ului
- “name” – ID-ul global prin care se identifică acest obiect JSON. Ca și convenție pentru aceste ID-uri am folosit următoarea forma:”flow_indexul fluxului_tip trafic_ID switch”. Aceasta formă asigură unicitate pentru fiecare obiect trimis
- “in_port” – portul pe care traficul intră în switch
- “actions” – specifica portul pe care să iasă traficul
- “arp_tpa” – adresă IP destinație
- “eth_type” – specifica tipul traficului, 0x806 fiind ARP și 0x800 fiind IP

Câmpurile cookie, priority și active sunt la fel pentru fiecare obiect JSON.

Pentru traficul IP se trimite următorul obiect JSON:

{

"switch": "00:00:00:00:00:00:00:01",

"name": "flow_1_IP_01",

"cookie": "0",

"priority": "32765",

"active": "true",

"in_port": "5",

"actions": "output=3",

"ipv4_src": "10.0.0.1",

"ipv4_dst": "10.0.0.7",

"eth_type": "0x800"

}

Unele dintre aceste câmpuri sunt explicate la traficul ARP, în continuare le descriu doar pe cele noi.

- “ipv4_src” – adresă IP sursă
- “ipv4_dst” – adresă IP destinație

Dacă traficul este bidirecțional trebuie să configuram și acest lucru în switch-uri. Pentru asta trebuie să trimitem fiecare obiect încă odată, dar interschimbăm porturile, adresa IP sursă cu cea destinație și la atributul “name” adăugăm string-ul “_rev” pentru a nu suprascrie obiectul copiat.

In final testam dacă fluxurile funcționează între gazde. Intrăm în consola Mininet și acolo scriem următoarea comanda: “nume_sursă ping nume_destinatie”. Fluxul a fost stabilit cu succes dacă destinația ne răspunde.

3.7. CITIREA ȘI SCRIEREA FIȘIERELOR

Fisierul care conține output-ul comenzi net din consola Mininet are următoarea forma:

h1 h1-eth1:CS4-eth7

- h1 – numele gazdei
- eth1 – portul 1
- CS4 – switch-ul cu ID-ul 4

După ce aceasta conversie are loc se construiește topologia. Switch-urile și gazdele sunt puse pe două coloane, fiind necesară o aranjare a lor.

In ceea ce privește salvarea unei configurații, am construit clasa Configuration care reprezintă structura care se salvează în fișier, vedeți figura 3.7 de mai jos. Când scriem configurația în fișier avem grija să respectam structura acestei clase. Citirea fișierului este simpla deoarece tot ceea ce trebuie să facem este să deserializăm fișierul într-un obiect de tipul Configuration.

```
private Configuration ReadXMLConfiguration(string fileName)
{
    XmlSerializer serializer = new XmlSerializer(typeof(Configuration));
    using (TextReader reader = new StringReader(File.ReadAllText(fileName)))
    {
        try
        {
            return (Configuration)serializer.Deserialize(reader);
        }
        catch (InvalidOperationException e)
        {
            throw new InvalidOperationException(e.Message);
        }
        catch (Exception e)
        {
            throw new Exception(e.Message);
        }
    }
}
```

Metoda ReadXMLConfiguration citește tot fișierului și pune conținutul acestuia într-un obiect Configuration. Ceea ce rămâne de făcut este să reconstruim topologia și setările din datele culese, ceea ce este destul de simplu având în vedere faptul ca structura obiectului și conținutului acestuia se aseamănă foarte mult cu clasele care reprezintă echipamentele, linkurile și fluxurile.

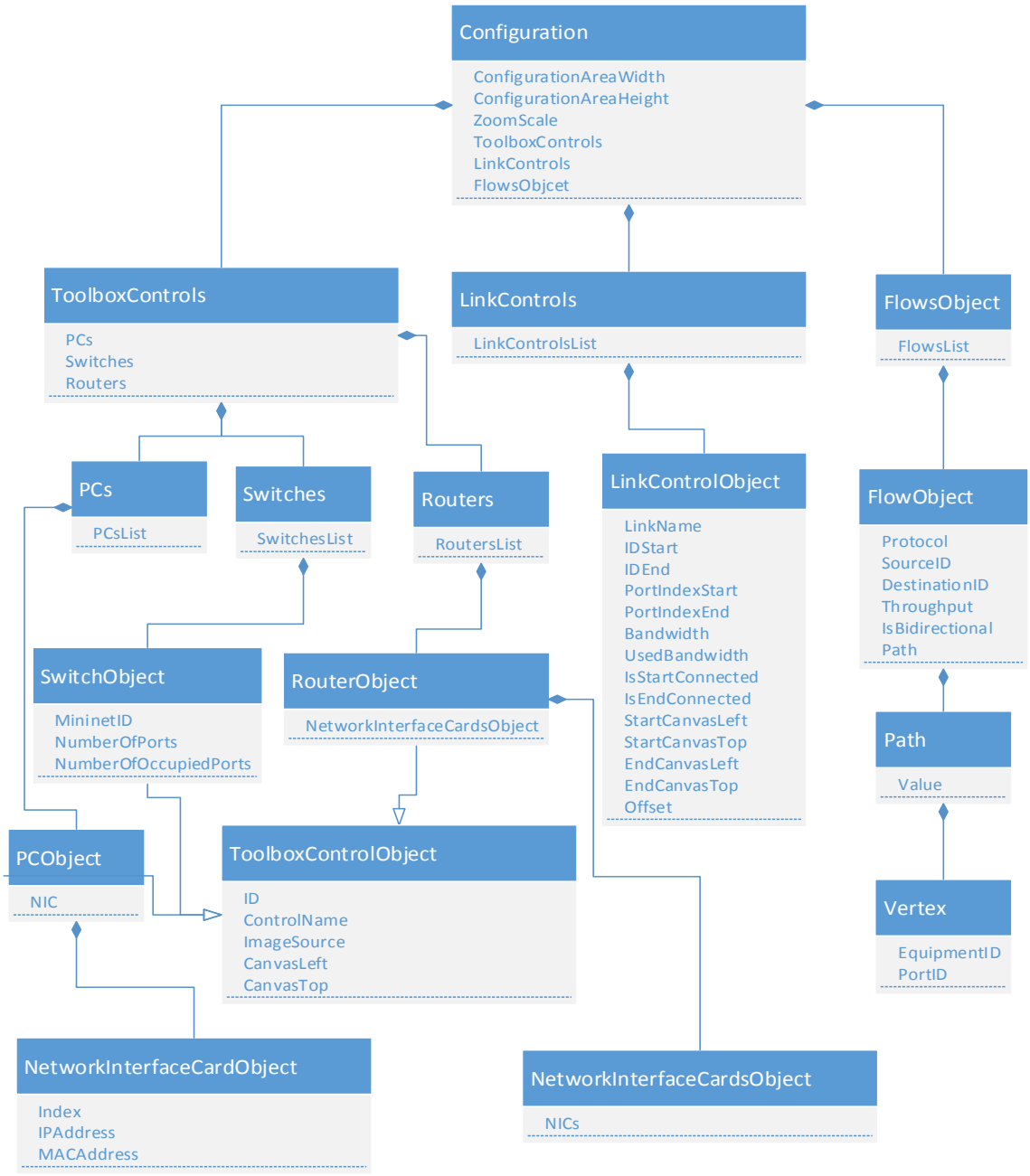

Figura 3.7 clasa Configuration

Pentru mai multe detalii despre WPF vedeți [4] și [7].

CONCLUZII

Aceasta aplicație a reprezentat pentru mine o provocare din punct de vedere al cunoașterii tehnologiei. Înainte de asta nu aveam experiența în construirea aplicațiilor desktop. Din punct de vedere al noțiunilor teoretice pe care a trebuit să le cunosc despre rețele și rutare nu a trebuit să depun un efort semnificativ, trebuia doar să cunosc pana la un nivel abstract modul de operare al rețelelor, cum ar fi protocolul ARP, iar algoritmi de găsire a rutelor cele mai optime deja există. O provocare a constat în transpunerea rețelei într-un graf pentru a putea aplica acești algoritmi. Cea mai mare provocare pentru mine a fost să mă familiarizez cu WPF. Un subsistem al framework-ului .NET pe care am ajuns să îl apreciez pentru puterea sa. Mai ales pentru modalitățile simple și directe pe care le are pentru a rezolva o anumită problemă de interfața grafică.

In urma muncii depuse pentru realizarea acestei lucrări am aflat ca modul de administrare al rețelelor trebuie să se schimbe pentru a putea face față cu ușurință noilor probleme din domeniu. O soluție este SDN, aceasta separă controlul rețelei, creierul, de redirecționare, mușchii rețelei, și oferă o vizualizare centralizată a rețelei pentru o administrare și automatizare eficientă a serviciilor de rețea.

În ceea ce privește îmbunătățirea aplicației, o trăsătură pe care aș implementa-o ca un următor pas, ar fi recalcularea rutei după ce un echipament sau link este șters, respectiv deconectat.

Zona topologiei care poate fi vizualizată depinde de nivelul de zoom, distanța dintre echipamente și dimensiunea ecranului. În cazul rețelelor cu un număr foarte mare de echipamente, găsirea unuia ar fi un proces posibil îndelungat și epuizant. O soluție care ar rezolva această problemă ar consta în implementarea unei trăsături care are rolul de a căuta un echipament după nume, adresă IP, sau orice altă trăsătură a acestuia și ca rezultat să centreze acel echipament pe ecran.

Timpul necesar determinării rutei optime crește cu numărul de porturi ale switch-urilor. Implementarea curentă transpune toate porturile unui switch în noduri din graf, inclusiv cele care nu sunt conectate la vreun link. O optimizare în acest sens ar consta în eliminarea din graf a porturilor goale. Asta ar rezulta în micșorarea numărului de noduri și muchii din graful creat și un timp mai scurt necesar calculării rutei. În stadiul actual al aplicației această problemă s-ar putea rezolva prin setarea numărului de porturi ale switch-ului să fie egal cu numărul de linkuri la care este conectat.

BIBLIOGRAFIE

[1] Reaz Ahmed, Raouf Boutaba, Design considerations for managing wide area software defined networks, IEEE Communications Magazine, 15 iulie 2014

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms (3rd Edition), The MIT Press, 31 iulie 2009

[3] Daniel F. Machedo, Dorgival Guedes, Luiz F. M. Vieira, Marcos A. M. Vieira, Michel Nogueira, Programmable Networks - From Software-Defined Radio to Software-Defined Networking, IEEE Communications Surveys & Tutorials, 11 februarie 2015

[4] Adam Nathan, WPF 4.5 Unleashed, Sams Publishing, 9 august 2013

[5] Chris Sells, Ian Griffiths, Programming WPF, O'Reilly Media; 2nd Edition, 7 septembrie 2007

[6] Andrew S. Tanenbaum, David J. Wetherall, Computer Networks (5th Edition), pag. 465-487, Pearson, 7 octombrie 2010

[7] RFC 7426, https://tools.ietf.org/html/rfc7426, ISSN: 2070-1721, ianuarie 2015

[8] https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Floodlight+Documentation

[9] http://mininet.org/

[10] https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/drag-and-drop-overview