University POLITEHNICA of Bucharest
Faculty of Automatic Control and Computer Science

REZUMAT TEZĂ DE DOCTORAT

Optimizarea Rețelelor Definite Software În Centre de Date Moderne

Optimization of Software Defined Networks In Modern Data Centers

Autor: Ing. Ovidiu Mihai Poncea

Coordonator științific: Prof. Dr. Ing. Florica Moldoveanu

COMISIE DOCTORAT

Preşedinte	Prof. Dr. Ing. Theodor Borangiu	Universitatea POLITEHNICA din București
Coordonator științific	Prof. Dr. Ing. Florica Moldoveanu	Universitatea POLITEHNICA din București
Referent	Prof. Dr. Ing. Victor Valeriu Patriciu	Academia Tehnică Militară
Referent	Prof. PhD. Eng. Alexandru Soceanu	Munich University of Applied Sciences
Referent	Prof. Dr. Ing. Nicolae Ţăpuș	Universitatea POLITEHNICA din București

București, 2016

CUPRINS

1 INTRODUCERE

În ultimii 25 de ani am văzut ascensiunea Internetului de la o simplă rețea ce interconecta câteva sisteme guvernamentale, cadre universitare și entuziaști la statutul de "serviciu de utilitate publică", prezent peste tot și folosit de către toată lumea. În acest timp, viteza la care utilizatorii se conectează la Internet a crescut de la câteva kilobiți/s la gigabiți/s iar numărul de utilizatori a ajuns la 3,5 miliarde (estimare - iulie 2016 [1]) și se estimează că traficul pe internet se va dubla în doar 3 ani (2016-2019) [2]. Această creștere rapidă a cererii pune o presiune constantă asupra infrastructurii de rețea făcând-o să se extindă foarte rapid. Acest fapt ducând la creșterea costurilor și complexității. De asemenea, cantitatea tot mai mare de date transferate are nevoie de prelucrare și de stocare. Aceste sisteme, *rețea*, *prelucrare* și *stocare* au, la rândul lor, nevoie de upgrade-uri constante. Și nu se prevede un sfârșit pentru această nevoie exponențială de creștere. De aceea, orice inovație ce poate contribui fie la reducerea complexității fie a costurilor este binevenită.

O astfel de inovație este ceea ce noi numim procesare in *Cloud*. Un *cloud* asigură o capacitate enormă de procesare și de stocare la un cost mai mic decât soluțiile anterioare (adică *mainframe*-uri proprietare, computere de sine stătătoare sau Centre de Date private mici). *Cloud*-ul, pentru a reduce costurile, atât CAPEX cât si OPEX, aduce în prim plan proiectarea modulară și partajarea resurselor dintr-un Centru de Date cu mai mulți utilizatori (*multitenancy*).

O altă îmbunătățire importantă provine din utilizarea unor procese *agile* în contrast cu procesele tradiționale, mult mai riguroase. Agilitatea noilor procese oferă companiilor posibilitatea de a se adapta rapid la schimbare și de a aduna *feedback*-ul esențial în primele stadii ale dezvoltării. În acest mod produsele sunt lansate pe piață mai repede, riscurile sunt reduse la minimum, iar produsele au funcționalitatea cea mai apreciată de utilizatori.

Agilitatea, ce în general s-a dovedit benefică pentru industrie, s-a lovit însă de limitările IT-ului: costul ridicat și ciclul lent de dezvoltare al *hardware*-ului. Deși caracteristicile oferite de *hardware* țin în general pasul cu necesitățile *software*-ului, totuși ciclurile lente de dezvoltare impactează timpul de lansare pe piață al produselor. Motivul principal este costul, companiile fiind nevoite să folosească echipamentele deja achiziționate o perioadă lungă de timp (6 luni poate fi o perioadă lungă într-o piață în continuă schimbare). Până când va fi descoperită o modalitate mai rapidă și mai ieftină de a crea și livra *hardware*-ul, industria are nevoie să limiteze efectele ciclurilor lente de dezvoltare ale acestuia cât mai mult posibil. O modalitate de a face acest lucru este prin adăugarea unui nivel de abstractizare între *hardware* și *software*. Acest nivel ar permite software-ului să evolueze independent de hardware. Această abordare a dat naștere unui nou concept: Totul Definit prin Software (*Software Defined Everything* - SDX sau SDE).

În această nouă abordare, Rețelele Definite Software (SDN) ocupă un loc important deoarece asigură infrastructura de comunicații a întregului centru de date și conexiunea acestuia cu exteriorul. Principalele preocupări ale SDN sunt simplificarea, reducerea costurilor, programabilitate și agilitate în rețelele de comunicație de date.

Chiar dacă SDN promite să fie o tehnologie care să redefinească modul de funcționare al rețelelor de comunicații, este încă departe de a ajunge în acel punct. Pentru a aduce SDN-ul la scalabilitatea, performanța și securitatea din rețele tradiționale este nevoie de mult efort. Rețelele tradiționale au atins acest nivel prin mulți ani de dezvoltare lentă, dar solidă.

2 REȚELELE DEFINITE SOFTWARE

În rețelele de calculatoare, există de obicei două tipuri de noduri: cele care creează și consumă pachete și cele care direcționează traficul. Vom numi primul tip de noduri *gazde* sau *servere* iar al doilea tip *switch*-uri. Gazdele există la periferia rețelei iar *switch*-urile formează nucleul rețelei. Interconectarea diferitelor rețele se face prin *switch*-uri sau *router*-e periferice. Rețeaua poate fi, de asemenea, văzută ca un graf de noduri unde gazdele sunt frunzele grafului, majoritatea switch-uri sunt noduri interioare și doar câteva switch-uri, cele de la graniță, sunt frunze.

2.1 Definiția SDN

De-a lungul timpului, SDN a fost definită de mai multe entități, atât academice cât și din industrie, după cum au înțeles conceptul și de interesele pe care le-au avut. SDN își are originea în mediul academic iar primele definiții legau conceptul de protocolul OpenFlow și de separarea dintre planurile de date și planul de control. La acel moment SDN și OpenFlow erau aproape sinonime.

Definiția cea mai cuprinzătoare și corectă este cea dată de Fundația pentru Rețele Deschise (*Open Networking Foundation* – ONF):

> *"Rețeaua Definită Software (SDN) este o arhitectură emergentă, dinamică, ușor de gestionat, eficientă din punct de vedere al costului și adaptabilă. Datorită acestor caracteristici este ideală pentru aplicațiile dinamice din ziua de astăzi ce folosesc lățime mare de bandă. Această arhitectură decuplează planul de control al rețelei de funcțiile de expediere a traficului ceea ce permite planului de control să devină direct programabil dar și abstractizarea infrastructurii de bază pentru aplicații și servicii de rețea."* [3]

Rețelele Definite Software au 3 caracteristici importante:

1. Control centralizat,
2. Separarea planului de date (i.e. deciziile de direcționare) de control și aplicații,
3. Programarea comportamentului rețelelor printr-un set de API-uri (*Application Programming Interfaces*).

2.2 Separarea planului de Control de cel de Date

Cea mai importantă diferență dintre SDN și modelul tradițional de rețea este separarea dintre planul de date si planul de control.

Planul de control este reprezentat de *controler*-ul SDN ce poate fi centralizat – o singură entitate – sau distribuit – mai multe entități care cooperează pentru a oferi o imagine centralizată, unificată și transparentă a rețelei. Planul de control este considerat a fi centralizat, deoarece vederea asupra rețelei este unică, chiar dacă acest controler este distribuit. Un controler distribuit oferă

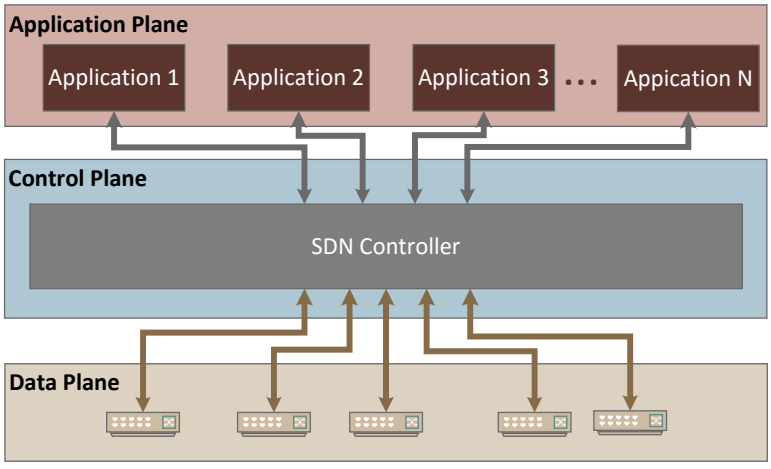


Figura 1: Separarea dintre Date, Control & Aplicații

avantaje, cum ar fi o disponibilitate ridicată și o performanță mai bună în detrimentul unei complexități crescute.

Principalele avantaje ale controlului centralizat îl reprezintă imaginea unificată asupra rețelei oferită aplicațiilor și simplificarea modelului de programare. Pentru a efectua modificări într-o rețea, aplicațiile nu mai trebuie să își creeze propriile lor opinii despre cum arată rețeaua prin accesarea fiecărui element în parte, ele pot accesa pur și simplu informațiile din controler prin interfețele acestuia (API-uri).

2.3 Cerințe și avantaje

Cele mai importante cerințe ale rețelelor definite software sunt:

- **Automatizare** - fluxurile de lucru manuale trebuie să fie automatizate
- **Virtualizarea rețelei** – mai mulți utilizatori beneficiază de aceeași infrastructură. Utilizatorii sunt izolați unul de altul
- **Securitate mai bună** - trafic izolat între utilizatori
- **Reducerea complexității** - centrele de date sunt complexe, orice reducere este binevenită
- **Reducerea costurilor**
- **Reducerea timpului de implementare.**

Avantajele SDN în comparație cu rețelele convenționale:

- **Ușurință la:**
  - Dezvoltarea software.
  - Mentenanță - operațiile sunt automatizate, nu este nevoie de scripturi complexe.
  - Înlocuirea echipamentelor.
  - Integrarea cu mașini virtuale
  - Virtualizarea rețelei.
  - Mai mulți utilizatori folosesc în comun aceeași infrastructură - producție, testare, dezvoltare, de cercetare utilizează aceeași infrastructură fizică;
  - *Upgrade in-service* - upgrade-urile sunt mai rapide și mai sigure.
  - Reducere a congestiei de date - aceasta este o problemă complexă, atât în rețelele convenționale cât și în SDN, dar noua arhitectură oferă instrumente mai bune pentru a le gestiona.
- **Planul de Management permite:**
  - Coordonarea între comportamentul diferitelor tipuri de servicii (de exemplu, un router + firewall + IDS coordonate de aceeași aplicație).
  - Configurație și management integrat între diferite tipuri de dispozitive - nu numai *switch*-uri și rutere, dar și servere, IDS, firewall-uri si funcții virtuale.
  - Aplică metodele din Știința Calculatoarelor la rezolvarea problemelor din SDN.
  - Orchestrarea realizată cu ajutorul unor scripturi personalizate scrise în limbaje de nivel înalt (de exemplu, Python).
  - Ingineria software aplicată la rețea: agilitate (dezvoltare iterativă), discipline (de exemplu, analiza, dezvoltare, testare, instalare)

2.4 Implementări *Open Source* de controlere

De-a lungul timpului au fost dezvoltate mai multe controlere SDN. În Tabelul 1 este prezentată starea celor mai cunoscute. S-a analizat dimensiunea programelor (număr linii de cod) și al activității (modificări pe lună).

Tabelul 1: Starea implementării controlerelor SDN

Nume	Sursă	Limbaje de programare	LOC	Activitate/lună	Observații
NOX	ICSI	C++ 64%, Python 27%, Altele 9%	81,980	Inactiv	Ultima contribuție a fost în Septembrie 2010
POX	ICSI	Python 96%, Altele 4%	20,928	Inactiv	Ultima contribuție a fost în Iulie 2012
Ryu	NTT Comm.	Python 87%, Erlang 9%, Altele 4%	115,000	33 commit-uri, 8 contributori	Proiect vechi cu activitate redusă dar stabilă
Beacon	Stanford University	Java 85%, Altele 15%	46,000	Inactiv	Beacon devine Floodlight deci ne așteptăm să fie inactiv
Floodlight	Big Switch Networking	Java 98%, Altele 2%	98,000	20 commit-uri, 7 contributori	Controler mai vechi cu activitate redusă, devoltarea a încetat
Pyretic	Mediu universitar (mai multe)	Python 80%, Erlang 9%, Altele 11%	188,000	40 commit-uri, 2 cont	Activitate foarte redusă
Trema	NEC	Ruby, C	60,000	Inactiv	2 commit-uri în ultimele 6 luni
OpenMUL	OpenMUL Foundation	C 86%, Shell script 7%, Altele 7%	213,741	Inactiv	Ultima contribuție a fost în Septembrie 2015
OpenDaylight	OpenDaylight Foundation	Java 55%, C++ 16%, JavaScript 9%, Altele 20%	2.6 Mil	1000 commit-uri 120 cont	Activitate foarte mare, numărul de linii de cod crește foarte repede!
ONOS	ON.Lab	Java 80%, JavaScript 14%, Altele 6%	450,000	500 commit-uri, 50 cont	Al doilea proiect cu o creștere rapidă

Din tabel observăm că, de departe, cele mai dezvoltate sunt OpenDaylight și ONOS.

3 CENTRE DE DATE DEFINITE SOFTWARE SI PROCESAREA ÎN CLOUD

3.1 Introducere

Într-un centru de date definit software (*Software Defined Data Center* – SDDC) resursele de procesare (*compute*), cele de rețea și cele de stocare sunt abstractizate. Procesarea și rețelele sunt virtualizate iar stocarea este partiționată și distribuită. Într-un SDDC fluxuri de date pot fi migrate de la o mașină la alta fără întreruperi[1]. Diferențele de hardware și complexitatea configurării este ascunsă utilizatorilor. Resursele sunt gestionate de o Platformă de Management a *Cloud*-ului (*Cloud Management Platform* – CMP) și sunt prezentate utilizatorului într-un mod abstract. Accesul la aceste resurse se face prin intermediul unei interfeţe în linie de comandă (CLI), a unei interfeţe grafice (GUI) - de obicei bazată pe Web și / sau a unui API (de obicei, REST).

3.2 Clasificarea și Caracterizarea Centrelor de Date

Centrele de date sunt facilități (clădiri, încăperi) utilizate de companii pentru a ține servere, echipamente de stocare și de rețea pentru nevoile unei singure sau a mai multor entități (de exemplu companii, departamente). Acest lucru implică stocarea, procesarea și difuzarea unor cantități mari de date clienților ce folosesc aplicații implementate cu arhitecturi client-server.

Cerințele unui Centru de Date sunt diverse și depind în mare măsură de clasificarea Centrului de Date. Așa că, doar prin a înțelege ce *tip* de DC analizăm, putem estima cerințele de nivel înalt și restricțiile impuse de acestea la proiectare și implementare.

Această teză propune următoarele criterii pentru clasificarea centrelor de date:

1. **Dimensiunea și disponibilitatea** – acesta este modul în care standardul ANSI/TIA-942 clasifică centrele de date. Această clasificare reprezintă o simplificare a mai multor caracteristici: putere și redundanță de răcire, redundanța componentelor, toleranță la erori, putere, intervalul maxim de întrerupere în funcționare permis.
2. **Scop** – tipul de aplicații pe care îl servește:
    - *Construite în jurul unei aplicații* - special construite pentru una sau mai multe aplicații.
    - *Generic – astfel de* centre de date se concentrează pe furnizarea de suport pentru orice aplicație.
3. **Modelul de Orchestrare** – este definit de modelul de gestionare și partajare a resurselor între utilizatori. Acest model influențează multe alte caracteristici, cum ar fi topologia de rețea, stocare și implementare:
    - *Management al unui Cluster ce folosește o aplicație Middleware* – un cluster este un set de noduri ce oferă capabilități de procesare similare sau chiar identice ce sunt conectate între ele printr-o rețea de mare viteză. Acestea au obiectivul de a finaliza sarcini specifice într-un mod distribuit;
    - *Folosind o platformă de management al Cloud-ului* (Cloud Management Platform – CMP) – CMP-ul este o aplicație special concepută care gestionează resursele de tip *cloud* și le oferă clienților;

[1] Sau cu întreruperi minore dacă migrarea *live* a mașinilor virtuale nu este suportată.

- *Management specific unei aplicații* - centre de date construite pentru o anumită aplicație sau un set de aplicații.

4. **Modelul Serviciilor**:
    1. *IaaS*: Infrastructura ca serviciu - oferă mașini virtuale, containere sau servere *bare metal* (goale, fără software pe ele) pentru utilizatori ai *cloud*-*ului*;
    2. *PaaS*: Platforma ca serviciu - oferă diferite medii de execuție pentru utilizator;
    3. *SaaS*: *Software*-ul ca serviciu - utilizatorul, de obicei, se poate conecta într-o aplicație software furnizată de către operatorul *cloud*-ului SaaS.
5. **Modelul de funcționare** - Cine deține și cine utilizează un Centru de Date:
    - *Privat* – *cloud*-ul este deținut de aceeași entitate care îl și utilizează;
    - *Comunitate* - mai multe entități cu interes comun utilizează același *Cloud*;
    - *Public* – cu mai mulți utilizatori (*multitenant*), entități externe pot închiria resurse din aceste *cloud*-uri. Centrul de Date are un singur proprietar (operator), celelalte entități sunt externe dar au acces la el;
    - *Hibrid* – *cloud* privat, care se extinde în cloud-uri publice atunci când propriile resurse nu sunt suficiente.
6. **Topologia de rețea** – În *cloud*-uri cele mai utilizate topologii sunt cele din categoria Clos: *Fat-tree* și *leaf-spine*.
7. **Modelul de stocare** - Modelul de stocare a evoluat în timp de la stocare locală la Rețele de Stocare Zonale (Storage Area Network – SAN) și ulterior la stocare distribuită:
    - *Rețea de stocare independentă*
    - *Rețele de stocare de date parțial convergente*
    - *Convergență a rețelelor de stocare peste rețeaua de date*
    - *Hiper-convergență*
    - *Stocarea locală* – stocarea este accesibilă la nivel local pe fiecare server, nu este nevoie de rețea.
8. **Modelul de Procesare** - clasifică Centrul de Date pe baza modului în care sunt gestionate resursele de procesare. Modelele folosite sunt:
    - *Bare-metal* – servere ce nu au sistem de operare; sunt instalate de la distanță de operator.
    - *Virtualizare Hardware* – serverele folosesc virtualizare hardware pentru a partiționa resursele.
    - *Virtualizarea la nivel de Sistem de Operare (containere sau jails)* - sistemul de operare permite existența mai multor medii de execuție izolate; izolarea se face prin mecanisme dedicate în *kernel*-ul serverelor.

3.3 Clasificarea traficului în Centrele de Date

Profilul de trafic al aplicațiilor ce rulează într-un Centru de Date este utilizat de către administratori pentru optimizarea rețelei în vederea îmbunătățirii performanței aplicațiilor.

Informațiile necesare creării pofilelor de trafic pot fi obținute prin eșantionarea și analizarea pachetelor din rețea după diferite criterii.

Analiza se referă la determinarea claselor corecte de trafic și debitul fiecărei clase, fie prin valori relative (de exemplu, procentul din totalul traficului, procent pe o perioadă de timp) sau valori absolute (de exemplu MB/s, secunde sau pps - pachete pe secundă).

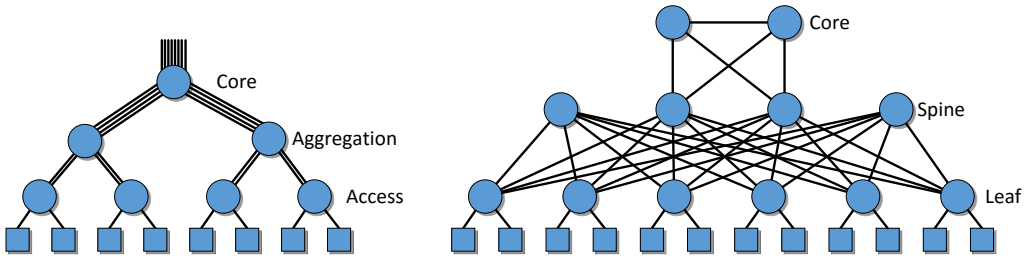
Această teză propune următoarele criterii de clasificare:

1. **După direcție** – direcția în care curg fluxurile de trafic printr-o rețea. Din exteriorul rețelei spre interiorul centrului de date (*Nord-sud*) sau generate și consumate în interiorul centrului de date (*Est-vest*). Cantitatea de trafic în fiecare direcție este de obicei măsurată în procente din total și este importantă în alegerea topologiei și capacității conexiunilor de date.
2. **După durata de viață a unui flux** – scurtă (*dragonflies*), normală și lungă (*toirtoise*).
3. **După impactul asupra lățimii de bandă utilizate** – unele fluxuri au un impact mai mare asupra lățimii de bandă utilizată decât altele.
4. **În funcție de tipul de tranzacție** – o tranzacție de rețea este o secvență de schimb de informații necesare pentru a finaliza o anumită sarcină. O tranzacție poate avea o singură sursă și o singură destinație (unu-la-unu), o singură sursă și destinații multiple (unu-la-multe) sau surse multiple și o singură destinație (multe-la-unu).
5. **După dimensiunea pachetului** – fluxuri cu pachete mari sunt procesate aproape la fel de rapid de *switch*-uri și dispozitive speciale ca cele cu pachete mici. Motivul este că majoritatea procesării este de obicei folosită la manipularea antetului. Optim este ca pachetele sa fie cât mai mari și mai puține**.**
6. **După aplicație** – această clasificare este importantă pentru monitorizare și stabilirea QoS per aplicație. Exemple de aplicații ce pot face parte din această clasificare: Distribuit de stocare, HTTP, FTP, voce (VoIP), video, Torrent etc.
7. **După prioritate** – o prioritizare strictă poate fi suficientă în unele cazuri, dar de obicei se utilizează o abordare mai fin granulată.

3.4 Topologii de Rețele în Centre de Date

Topologiile rețelelor de calculatoare din Centrele de Date sunt influențate în principal de capacitate, scalabilitate, costuri și, într-o anumită măsură, de cablarea rețelei. În practică, datorită limitărilor de scalabilitate a majorității topologiilor (mai ales la cablare) în majoritatea cazurilor sunt folosite variații ale a topologiei de tip *Clos*. Din aceste variații două dintre ele ies în evidenţă: *Fat-tree* și *leaf-spine* (Figura 2). Acestea două reprezintă nucleul celor mai multe Centre de Date.

Fat-tree este un *arbore* în care rădăcina este conexiunea *backbone* cu exteriorul iar serverele sunt frunze. În forma sa cea mai pură (Figura 2-1) nu prevede nici o suprasubscriere (*oversubscription*) astfel încât, la fiecare nod, conexiunea către nodul părinte reprezintă suma conexiunilor către nodurile copil. În mod implicit nu prevede nici redundanță, în cazul în care un nod eșuează pe toți copiii săi vor fi deconectate de la restul rețelei. De notat că, pentru o scalabilitate mărită, înălțimea arborelui poate fi mai mare decât cea a exemplului din Figura 2-1.



1. Fat-tree **2. Leaf-spine**

Figura 2: Variantele cele mai folosite ale topologiei de tip *Clos*

Diametrul (cea mai lunga distanță calculată pe drumul cel mai scurt între oricare două noduri din arborele rețelei) depinde de înălțimea de rețea. Diagrama din exemplu are un diametru de 5, ceea ce înseamnă că un pachet ce va traversa rețeaua de la un capăt îndepărtat la celălalt necesită parcurgerea a 5 noduri înainte de a ajunge la destinație. Prin urmare crește latența pachetelor. Acesta și este de obicei diametrul maxim acceptat al unei rețele dintr-un centru de date. Motivul este ca întârzierea introdusă să aibă un impact cât mai redus asupra performanței aplicațiilor.

Această topologie este foarte bună pentru trafic de tipul *Nord-Sud* (adică traficul ce părăsește Centrul de Date), deoarece nu oferă nici o suprasubscriere[2], dar poate fi mai rău pentru *est-vest* (adică traficul dintre serverele Centrului de Date) datorită diametrului mare.

Cea de a doua topologie utilizată, *leaf-spine*, are o înălțime maximă de 3, și are trei niveluri (*leaf*, *spine, core*) și nu oferă nici o suprasubscriere între *leaf* și *spine* (Figura 2-2). Caracteristica sa principală este aceea că fiecare *switch leaf* se conectează la fiecare *switch spine* (asta în forma ideală).

Această topologie este foarte bună pentru trafic *est-vest*, datorită diametrului care este egal cu 3. De asemenea este tolerant la defecțiuni ale *switch*-urilor *spine*. Această toleranță se datorează faptului că fiecare *leaf* este conectată la toate *spine*-urile din topologie. Deci pot fi formate mai multe căi redundante între sursă și destinație. Toleranța la defecțiuni ale *switch*-urilor *leaf* poate fi ușor adăugată prin conectarea fiecărui server la mai multe *switch*-uri *leaf* (a se vedea Figura 6 in 4.2 pentru un exemplu de *leaf-spine* tolerant la erori). *Leaf-spine* este avantajată în contextul SDN datorită conexiunilor redundante ce pot fi folosite pentru transmiterea datelor pe mai multe căi în același timp.

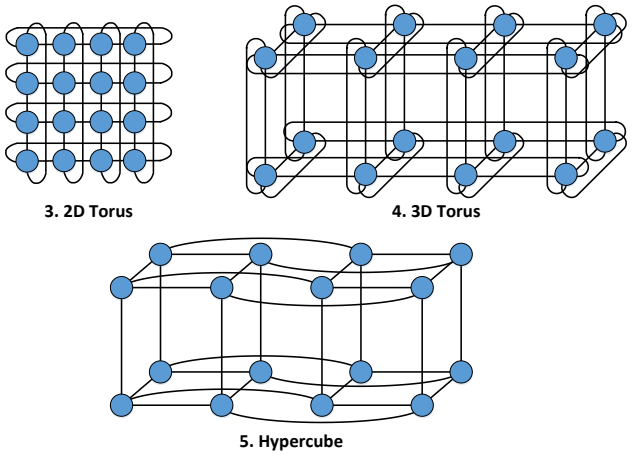


Figura 3: Topologiile *torus* și *hypercube*

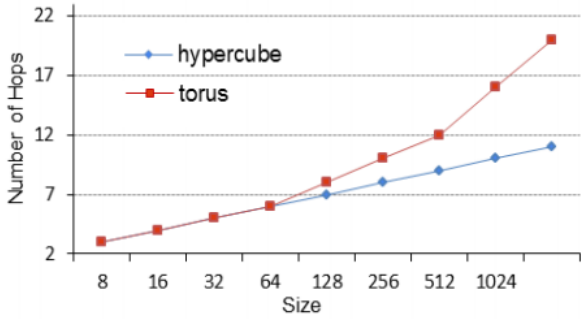


Figura 4: Diametrul rețelei la *torus 3D* și *hypercube* [4]

[2] În varianta ideală.

În afara celor două topologii prezentate mai sus, în unele centre de date, încă două topologii mai sunt uneori folosite: *Torus* și *Hypercube* (Figura 3).

Atât *torus* cât și *hypercube* sunt optimizate pentru traficul *est-vest*. Ele sunt de obicei utilizate în anumite centre de date pentru aplicații specifice (de exemplu, pentru HPC) la prelucrarea unor cantități mari de date de la nodurile apropiate (de exemplu la analiză de date - *data analytics* sau *MapReduce*). Diametrul lor crește odată cu numărul de noduri. Pentru *torus* crește exponențial în timp ce pentru *hypercube* crește liniar. De asemenea, lățimea de bandă de bisecție este mai bună pentru *hypercube* în topologii cu mai mult de 64 de noduri. Acestea sunt rezultatele analizei noastre din [ANDR 2015].

În general, atunci când se implementează topologii cu multe noduri (mai mult de 64) *hypercube* este avantajat în comparație cu *torus* atât din punct de vedere al performanței cât și din perspectiva costurilor. Pe topologii mai mici de 64 de noduri *torus* este mai bună deoarece are mai puține conexiuni. Prin urmare costul de implementare este mai mic iar performanța la același număr de noduri este similară.

Aceste arhitecturi de rețele constituie baza de la care pleacă proiectarea. Însă acestea sunt adaptate, de la centru de date la centru de date și de la aplicație la alicație. De exemplu pentru un centru de date dedicat aplicațiilor de e-learning traficul est-vest al unei topologii *Fat-tree* poate fi îmbunătățit prin realizarea de conexiuni suplimentare între nodurile vecine. Desigur, această îmbunătățire este un caz special și rezolvă problema aplicației noastre, deci nu poate fi generalizată [PIST 2014].

4 PROIECTAREA ȘI IMPLEMENTAREA CONTRON, CONTROLERUL OPENFLOW PENTRU SIMULATORUL DE REȚELE NS-3

În acest capitol propunem un nou controler pentru OpenFlow, construit de la zero, numit Contron. Acest controler funcționează cu simulatorul de rețele NS-3 și oferă servicii de bază pentru gestionarea de *switch*-uri OpenFlow dar și API-uri ce permit: definirea *acțiunilor*, *match*-urilor și programarea fluxurilor, citirea și actualizarea topologiei, trimiterea și primirea de notificări la evenimente generate de schimbări ale topologiei etc. Controlerul este extensibil - multe servicii pot fi ușor înlocuite sau personalizate **[PONC, 2016/1]**.

4.1 Arhitectura Contron

Arhitectura Contron este structurată pe trei niveluri: un *nivel de bază* (Core) ce furnizează funcționalități de care alte componente depind, un *nivel de servicii* și un nivel de *aplicații* independente unele de altele ce oferă funcționalitate *end-to-end*. Arhitectura este modulară iar componentele pot fi dezactivate în cazul în care nu sunt utilizate. Componentele nivelului de bază sunt întotdeauna necesare, în timp ce serviciile pot fi alese în funcție de nevoile aplicației. Arhitectura Contron cu un set complet de servicii activate este prezentată în Figura 5.

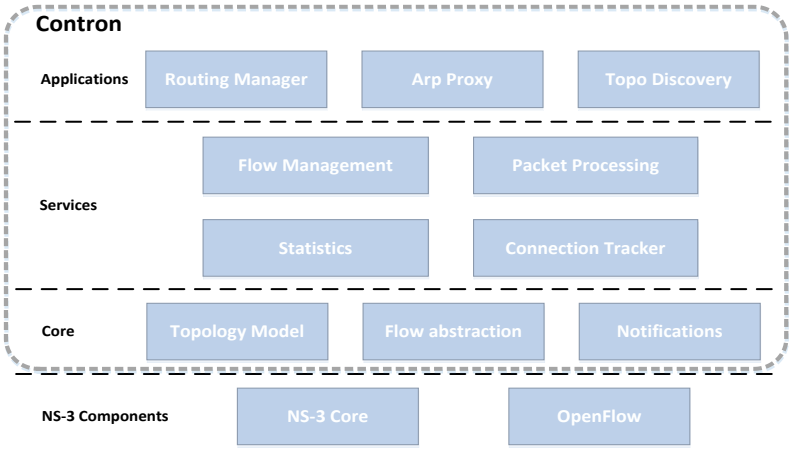


Figura 5: Arhitectura Contron

Componentele Contron sunt:

- **Notificări (*Notifications*)**. Notificările sunt generate la modificări în rețea. Aplicațiile și serviciile ce trebuie să reacționeze la schimbări din rețea se înregistrează la acestea și primesc notificări.
- **Modelul Topologiei (*Topology Model*)**. Reprezintă imaginea centralizată pe care o are controlerul asupra rețelei. Ea poate fi văzută ca o bază de date în memorie (*in-memory-database*) și conține toate *switch*-urile și proprietățile gazdelor dar și fluxuri și statistici.
- **Abstractizarea Fluxurilor (*Flow Abstraction*).** Folosită pentru definirea fluxurilor de date OpenFlow și pentru adăugarea sau scoaterea lor din tabele de comutare. Pentru fiecare flux, utilizatorul poate defini *acțiunile*, *match*-ul, *switch*-ul unde fluxul va fi adăugat și prioritatea.
- **Managementul fluxurilor (*Flow Management*).** Această omponentă ține evidența tuturor fluxurilor din topologie. Sunt suportate două categorii de fluxuri: *normale* și *de management*. Cele de management reprezintă intrări ce trebuie să fie prezente în toate *switch*-urile.

- **Statistici (*Statistics*)**. Această componentăcompletează modelul topologiei cu statistici colectate din toate switch-urile prin intermediul protocolului OpenFlow sau direct prin intermediul mecanismului NS-3 de urmărire; a se vedea [5] pentru mai multe detalii.
- **Procesare de Pachete (*Packet Processing*).** Pachetele primite de către controler sunt clasificate și transmise serviciilor pentru prelucrare. Fiecare serviciu primește pachetele pentru care s-a înregistrat.
- **Descoperirea Topologiei (*Topology Discovery*).** Oferă un serviciu simplu pentru descoperirea topologiei. Această componentă iterează toate nodurile definite în NS-3, identifică tipul lor (adică *switch* sau *gazdă*) și conexiunile dintre ele apoi actualizează Modelul Topologiei cu aceste informații.
- **Proxi Arp (*Arp Proxy*).** Oferă suport ARP pentru gazde, fără necesitatea de a difuza cererile ARP prin toată rețeaua. Această componentă primește și răspunde la cereri în numele gazdelor.
- **Supravegherea Conexiunilor (*Connection Tracker*).** Ține evidența tuturor conexiunilor din rețea. Conexiunile sunt create atunci când primul pachet dintr-un flux nou este trimis de sursa și ajunge la controler. Acest pachet este analizat și se adaugă o intrare unică bazată pe adresa sursă și destinație din antet. Serviciul trimite notificările când sunt detectate noi conexiuni.
- **Rutare Shortest Path (*Shortest Path Routing*).** Folosește algoritmul lui Dijkstra pentru determinarea celei mai scurte căi prin graful cu nodurile rețelei pentru a calcula și configura rutele optime.

4.2 Studiu de caz: observarea dimensiunii cozilor și profilul traficului la recepție pe conexiuni congestionate când rutele sunt actualizate dinamic

Să presupunem că, în rețele SDN, vrem să validăm un nou mecanism de rutare ce folosește informații de congestie. Pentru aceasta, alegem o topologie comună într-un Centru de Date, *leaf-spine*, și începem să simulăm.

De asemenea, să presupunem că în topologia noastră avem trei rack-uri (R_1, R_2, R_3), fiecare cu trei servere (S_{x1}, S_{x2}, S_{x3} unde x este numărul *rack*-ului). Serverele sunt conectate la *switch*-uri *Top of Rack* (Tor_1, Tor_2, Tor_3, Tor_4) prin două conexiuni redundante. Pe servere se execută mașini virtuale (VM_1, VM_2, VM_3). Pentru a simplifica simularea toate conexiunile sunt de 10 Gbps. Topologia este reprezentată în Figura 6.

Dimensionarea *buffer*-elor de pe porturi este o problemă complexă [6]. Pentru studiul nostru este suficientă o singură coadă la transmisie. Dimensiunea acesteia a fost aleasă la 1.43MB (1500B / pachet × 1000 pachete). 1.43MB se încadrează în dimensiunile cozilor pentru *switch*-urile de 10 Gbps [7].

Transfer de date se face prin UDP, mașinile virtuale (VM) din R_3 solicită blocuri de date cu dimensiunea de 32MB de la mașinile virtuale din R_1 (Figura 6). Cererile se întâmplă aproape în același timp, iar răspunsurile creează congestii în rețea. Acest model de trafic este similar cu cel al infrastructurilor de stocare Ethernet și IP (de exemplu, FCoE) și protocoalele de sisteme de fișiere distribuite (de exemplu NFS peste UDP). Am limitat simularea la 2 cereri, astfel încât să putem observa mai bine comportamentul cozilor și impactul modificării rutelor asupra traficului.

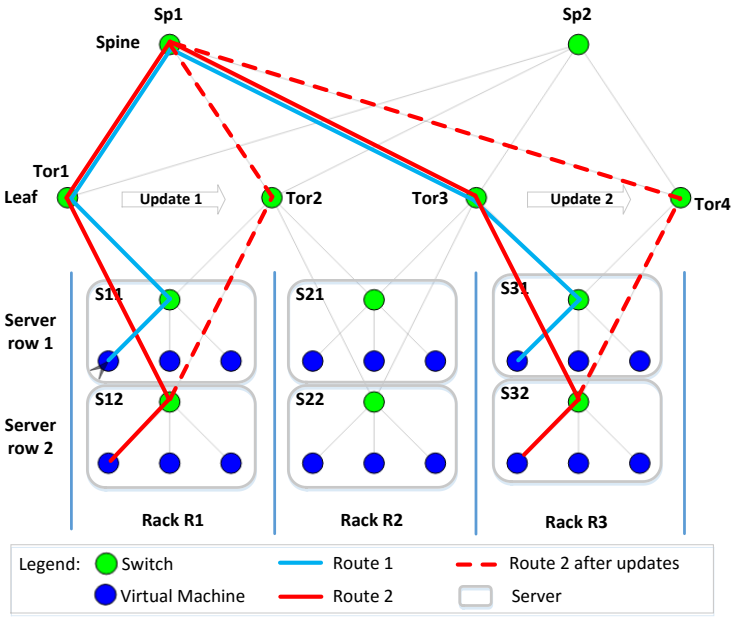


Figura 6 Topologia și rutele pentru studiul de caz

4.3 Descrierea simulării

Simularea începe când o mașină virtuală ce rulează pe serverul S_{11} răspunde la solicitarea venită de la o altă mașină virtuală dar de pe serverul S_{31}. Vom numi acest flux $Flow_1$. Cinci milisecunde mai târziu o mașină vrtuală de pe serverul S_{12} începe și ea să răspundă la o cerere venită de pe serverul S_{32} ($Flow_2$). Răspunsurile primite sunt trimise la 7 Gbps. Această valoare reprezintă aproximativ 700 MB/s de date, o valoare ușor de atins de matricele de stocare (*storage arrays*) de astăzi (Figura 7).

După cum se vede în Figura 6 rutele celor două fluxuri se suprapun în două locuri: pe conexiunea $Tor_1 \rightarrow Sp_1$ și pe conexiunea $Sp_1 \rightarrow Tor_3$. Debitul însumat al celor două fluxuri este de 14Gbps, mai mult decât capacitatea unui singure conexiuni (10Gbps). Prin urmare, coada de la Tor_1 începe să se umple (Figura 7 *dreapta*) iar conexiunea $Tor_1 \rightarrow Sp_1$ se saturează. Controlerul detectează acest lucru și redirecționează traficul în jurul conexiunii congestionate: de la $S_{12} \rightarrow Tor_1 \rightarrow Sp_1$ la $S_{12} \rightarrow Tor_2 \rightarrow Sp_1$ (*Update* 1 în Figura 6).

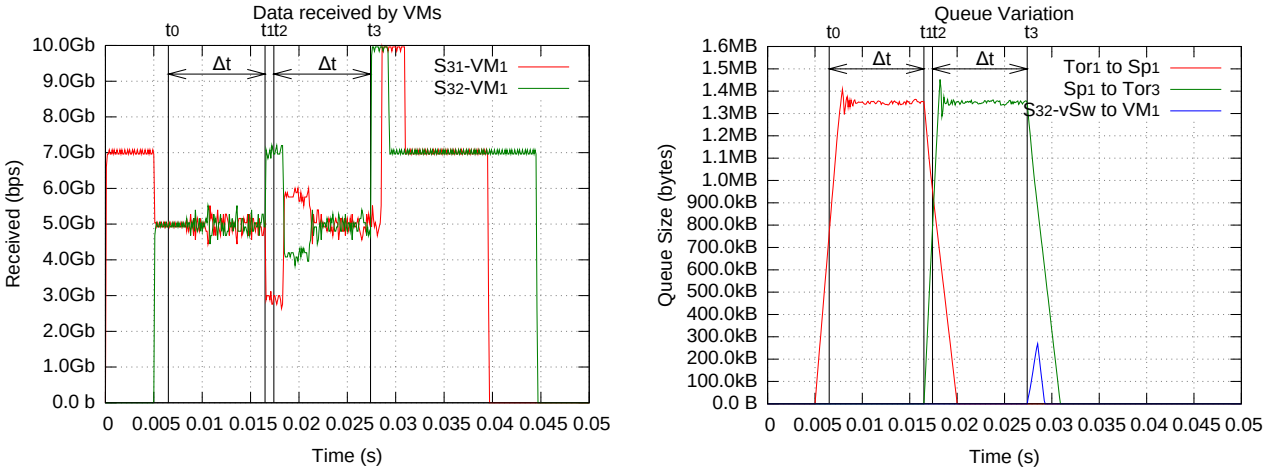


Figura 7 *Stânga*: Debitul la destinație; *Dreapta*: Variația cozii

După actualizarea primei rute (la t_1), o altă conexiune se saturează ($Sp_1 \rightarrow Tor_3$) iar controlerul decide rerutarea $Flow_2$ a doua oară (în t_2 detectează iar în t_3 finalizează rerutarea): de la $Sp_1 \rightarrow Tor_3 \rightarrow S_{32}$ la $Sp_1 \rightarrow Tor_4 \rightarrow S_{32}$ (*Update* 2 in Figura 6).

Intervalul dintre emiterea unei notificări de congestie și programarea unei noi rute în *switch*-uri a fost configurat la 10ms (Δt). În Centre de Date reale acest interval depinde de viteza de procesare a controlerului și viteza de actualizări a fluxurilor în *switch*-urile OpenFlow.

4.4 Analiza rezultatelor și concluzii

Analizând informațiile primite (Figura 7 *stânga*) vedem că, între 5ms și t_1, ambele fluxuri folosesc câte 50% din capacitatea legăturii saturate. În acest interval se pierd pachete deoarece coada dintre $Tor_1 \rightarrow Sp_1$ se umple.

La t_1, prima actualizare a rutei, se produce un efect interesant: debitul primit de $S_{32} - VM_1$ (adică $Flow_1$) ajunge brusc la maxim, în timp ce debitul la $S_{31} - VM_1$ (adică $Flow_2$) scade! Ne-am aștepta ca ponderea celor două fluxuri să fie aceeași (50%). După ≈ 0.2ms, chiar mai interesant, graficul arată un revers, cu S_{31} mai mare decât S_{32}.

Situația de mai sus se datorează faptului că Sp_1 primește pachete simultan din două surse:

1. La rata conexiunii (10Gbps) încă mai primește pachete vechi de la Tor_1. Acestea erau deja în coada dintre $Tor_1 \rightarrow Sp_1$ când s-a făcut actualizarea rutei.
2. Din cauza actualizării rutei primește noi pachete prin Tor_2, la 7Gbps.

Prin urmare, în timp ce Tor_1 continuă să transmită 5Gbps de traficul lui $Flow_1$ (acestea sunt pachete rămase în coadă) și 5Gbps din traficul lui $Flow_2$, noua rută actualizată de către controler va aduce trafic proaspăt de la S_{12} ($Flow_1$) prin Tor_2 direct în coadă lui $Sp_1 \rightarrow Tor_3$ la încă 7Gbps. Rezultatul este că traficul real de la S_{12}($Flow_1$) este de 12Gbps, în timp ce de la S11 ($Flow_2$) mai avem încă 5Gbps. Totalul este de 17Gbps! Acest *burst* practic umple următoarea coadă.

După un timp (≈20µs), *burst*-ul primit de la congestia anterioară în $Sp_1 \rightarrow Tor_3$ se termină dar, până ce se întâmplă acest lucru, pachetele fluxului $Flow_1$ vor continua să se acumuleze peste pachetele din coadă. Prin urmare, atunci când se termină *burst*-ul coada nu va fi goală, va conține pachetele fluxului $Flow_1$. Aceste pachete vor fi transferate la Sp_1 la viteza maximă a conexiunii. Acest lucru explică reversul ponderii fluxurilor din profilul de trafic (Figura 7).

După cea de a doua rerutare de la momentul t_3, ambele fluxuri urcă la viteza de 10Gbps, pentru un interval scurt, în timp ce cozile sunt golite. De asemenea, există un scurt *burst* în coada de transmisie a switch-ului de pe serverul S_{32} (în Figura 7 *dreapta* este graficul cu albastru).

În concluzie, dacă ne uităm la grafice putem observa că:

3. Rezolvarea congestiei pe o conexiune tinde să mute congestia pe conexiunea următoare din rută.
4. La congestie pachetele se acumulează în cozi și, atunci când congestia este eliminată, acestea continuă să fie transmise din vechea coadă provoacând un scurt *burst* de trafic ce umple coada următoare.
5. Pachetele care sosesc la destinație prin intermediul unor legături congestionate sunt întârziate din cauza timpului de așteptare în cozi. În același timp, pachetele ce vin pe noua rută ajung la

aceeași destinație înaintea celor vechi. Acest lucru strică ordinea pachetelor ceea ce cauzează probleme anumitor protocoale.

6. Timpul de reacție al controlerului la congestii ($\Delta t = 10ms$) este prea mare dacă intenționăm să eliminăm complet pierderile de pachete.

O soluție pentru reducerea Δt ar putea fi creșterea vitezei de reacție a planului de control astfel încât procesarea să fie mai scurtă decât intervalul necesar pentru ca o coadă pentru să se umple la peste 90%. Acest lucru înseamnă că planul de control trebuie să poată procesa un pachet și să ia o decizie în mai puțin de 1-2ms. Acest interval scurt se poate dovedi dificil de atins în condiții reale, în special pentru Centrele de Date cu topologii compuse din mii de switch-uri.

O altă soluție la problema de mai sus ar putea fi mărirea dimensiunii cozilor pentru a compensa întârzierea. Prin urmare, *switch*-urile vor trebui să aibă cozi de zece ori mai mari. Acest lucru pare să rezolve problema, dar, în cazul în care există neconcordanțe de viteză în rețea (de exemplu, trecerea de la 10Gbps la 1Gbps) și multe topologii au această problemă, atunci 10ms de coadă poate duce la întârzieri de pachete mult mai mari pe conexiunile de viteză mică (10ms la 10Gbps înseamnă 100ms la 1Gbps).

O soluție mai inteligentă este de a utiliza un mecanism similar cu QCN pentru rezolvarea congestiilor în timp real [PIST, 2015].

Recepția neordonată de pachete poate fi rezolvată prin discipline QoS inteligente care șterg pachete mai vechi ale aceluiași flux în momentul în care se face mutarea fluxurilor.

Concluzia finală a studiului de caz este faptul că, chiar dacă algoritmul nostru de congestie SDN pare să rezolve problema, mai este încă un drum lung de parcurs. Și acest lucru a fost dovedit pe un simulator cu ajutorul unui controler OpenFlow.

5 QCN-WFQR & SDN: MAXIMIZAREA TRAFICULUI ÎN REȚEA ȘI REDUCEREA PIERDERILOR DE PACHETE

Ethernet-ul a devenit principalul mediu de comunicație în rețelele de calculatoare datorită avantajelor sale incontestabile, cum ar fi: costuri reduse, viteze mari sau ușurința în administrare. Totuși acesta este un mediu *best-efort* unde transmisia pachetelor nu este garantată. Pentru a rezolva această problema, *IEEE Data Center Bridging Task Group* a dezvoltat o serie de îmbunătățiri ce includ *Quantized Congestion Notification* (QCN) [8]. Acest protocol a permis dezvoltarea unui mediu fără pierderi.

QCN este definit ca un protocol standard IEEE pentru rețelele tradiționale, totuși principiile sale pot fi adaptate la SDN iar modelul centralizat al controlerului poate fi folosit pentru a extinde și mai mult funcționalitatea acestuia.

În următoarele secțiuni, teza propune o metodă pentru a maximiza randamentul rețelei și a crea un mediu fără pierderi, prin reducerea pierderii de pachete în *centrul* rețelei. Acest lucru se realizează în trei moduri:

1. Reducerea traficului la sursă prin monitorizarea dimensiunii cozii.
2. Distribuirea încărcării traficului între mai multe servere ce găzduiesc aceeași aplicație. Distribuirea folosește informații despre congestie.
3. Migrarea fluxurilor deja stabilite pe rute alternative și mai puțin aglomerate, reducând astfel nevoia de a încetini traficul la sursă.

Implementarea pentru 1 și 3 s-a făcut în NS-3 și s-a simulat folosind *Contron* (prezentat în capitolul 4).

Propunerea se bazează pe QCN *Weighted Flow Queue Ranking* (QCN-WFQR). Algoritmul QCN-WFQR oferă mai mulți indicativi de congestie: per nod (locali) și per sistem. Acești indicativi sunt folosiți pentru decizii la nivelul controlerului și realizează o echilibrare a traficului în rețea oferind în același timp o performanță globală crescută a sistemului.

Mai multe detalii despre indicativii de congestie QCN-WFQR și aplicații în rețelele de calculatoare tradiționale sunt prezentate în [PIST, 2015].

5.1 Notificările de congestie cuantizate și SDN

După cum am prezentat mai devreme, informații despre congestiile conexiunilor de date sunt date de utilizarea cozilor de transmisie. O utilizare redusă a cozii înseamnă că nu avem nici o congestie în timp ce o utilizare care depășește un anumit prag (Q_{eq}) ne semnalează existența unei congestii. În QCN aceste informații ajung la sursă numai prin Mesaje de Notificare a Congestiei (CNMs), iar sursele de pachete (serverele) reacționează la aceste mesaje prin reducerea traficului.

În contrast, rețelele SDN au avantajul ca folosesc un controler centralizat, prin urmare, datele de congestie pot ajunge la sursă prin mecanisme diferite, sau nu ajung deloc dacă controlerul decide să renunțe la notificări. Prin urmare, cercetarea noastră propune trei modele de comunicare care pot fi aplicate pentru gestionarea congestiei: model de integrare minimală, model de integrare parțială și model de integrare completă. Aceste modele sunt descrise în detaliu în secțiunile următoare.

5.1.1 Indicativi de congestie

Weighted Flow Queue Ranking - WFQR reprezintă un set de indicativi care prelucrează informațiile de congestie brute și le convertește într-un format compact, mai potrivit pentru deciziile de congestie.

În rețelele SDN sunt utilizați următorii indicativi de congestie din [PIST, 2015]:

A. Indicativi locali - caracterizează un switch:

1. Rangul cozii (R_{queue}) – numărul de fluxuri dintr-o coadă:

$$R_{queue} = COUNT\langle flows\rangle \tag{5.1}$$

2. Ponderea cozii ($Share_{flow}$) – măsura congestiei unui flux dintr-o coadă:

$$Share_{flow} = F_{b:EMA_t} \times R_{queue} \tag{5.2}$$

unde $F_{b:EMA_t}$ reprezintă media feedback-ului transmis prin mesaje de congestie folosind funcția de Mediere Mobilă Exponențiala (*Exponential Moving Average* – EMA):

$$EMA_t = \alpha_t Y_t + (1 - \alpha_t) EMA_{t-1}, \text{ and} \tag{5.3}$$

$$\alpha_t = 1 - e^{-\frac{\Delta t}{T}}, \text{ therefore} \tag{5.4}$$

$$F_{b:EMA_t} = \begin{cases} \left(1 - e^{-\frac{\Delta t}{T}}\right) \cdot F_b + e^{-\frac{\Delta t}{T}} \cdot F_{b:EMA_{t-1}}, F_b \in flow \\ e^{-\frac{\Delta t}{T}} \cdot F_{b:EMA_{t-1}}, F_b \notin flow \\ 0, if\ t = 0 \end{cases} \tag{5.5}$$

unde T este intervalul de masurare iar Δt este intervalul de eșantionare;

B. Indicativi de sistem – calculați la nivelul controlerului:

3. Ponderea fluxului ($Weight_{flow}$) – măsura congestiei unui flux din punctul de vedere al sistemului. Se calculează ca suma tuturor ponderilor unui flux în toată topologia:

$$Weight_{flow} = \sum_1^{N-nodes} Share_{flow} \tag{5.6}$$

4. Ponderea Punctului de Reacție ($Weight_{RP}$) – măsura congestiei unui punct de reacție (server) Se calculează ca suma tuturor ponderilor fluxurilor specifice acelui punct de reacție:

$$Weight_{RP} = \sum_1^{N-flows/RP} Weight_{flow} \tag{5.7}$$

5.1.2 Modelul de integrare minimă

Acest model este similar cu cel din standardul QCN. Mesajele de congestie (CNM-urile) își păstrează formatul și trec prin rețea nemodificate. Serverele (Puncte de Reacție – RP-uri) nu au nevoie de nici o modificare, atâta timp cât acestea suportă standardul QCN, dar switch-urile (Puncte de Congestie – CP-uri) trebuie să păstreze tabele cu indicativii de congestie locali.

În acest model, indicativii de congestie ajung la controler prin intermediul *OpenFlow*, folosind noi mesaje de protocol pentru cozi și statistici. Prin urmare, ceva efort este necesar pentru definirea și implementarea acestor noi extensii în OpenFlow.

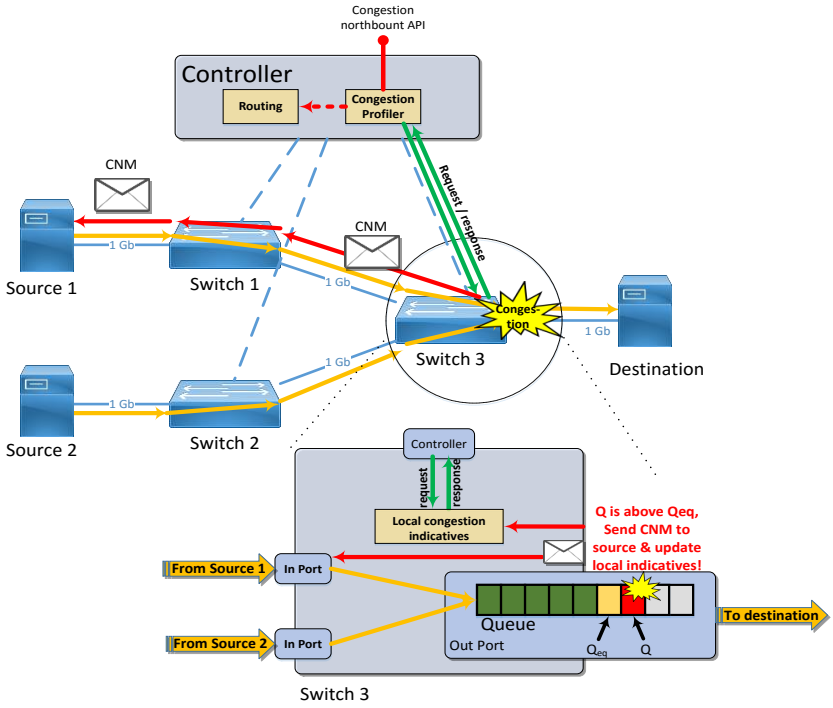


Figura 8: QCN și SDN – Modelul de integrare minimală

Controlerele SDN au nevoie de un nou serviciu. Îl vom numi *Congestion Profiler.* Acest serviciu trebuie să fie rapid, prin urmare, ar trebui să ruleze în interiorului controlerului nu ca o aplicație separată[3]. Rolul său este de a aduna și de a centraliza indicativii locali WFQR din *switch*-uri. Pe baza acestora calculează diferiți indicativi de congestie ai sistemului. Cu aceste informații este apoi alimentat un algoritm de rutare ce decide calea fiecărui flux (e.g. folosind algoritmul de calcul al drumului minim al lui Dijkstra).

Această abordare este simplă și are avantajul că utilizează aceleași mecanisme, dovedite în timp, din standardul QCN. Datorită acestui lucru întârzierea din controler afectează în mică măsură funcționalitatea sistemului, limitarea traficului având loc indiferent de deciziile controlerului. Dezavantajele acestui model sunt complexitatea ridicată a software-ului din switch-uri și necesitatea de a extinde OpenFlow. Toate acestea diminuează compatibilitatea cu standardul OpenFlow, ceea ce este de evitat.

5.1.3 Modelul de integrare parțială

In comparație cu modelul minim, în acest model notificările de congestie (CNM-urile) sunt trimise direct controler-ului, prin urmare vor fi necesare modificări minore în software-ul *switch*-urilor. Indicativii locali nu mai sunt păstrați în switch-uri, fiind calculați direct de către controler. Switch-urile vor monitoriza în continuare cozile și vor genera mesaje de congestie. De asemenea, nu sunt necesare modificări în OpenFlow, deoarece mesajele de congestie pot fi transmise ca mesaje de tip *packet-in* către controler și înapoi din controler către servere (CP-uri) sub formă de mesaje de tip *packet-out*.

[3] Aplicațiile separate sunt mai lente decât serviciile

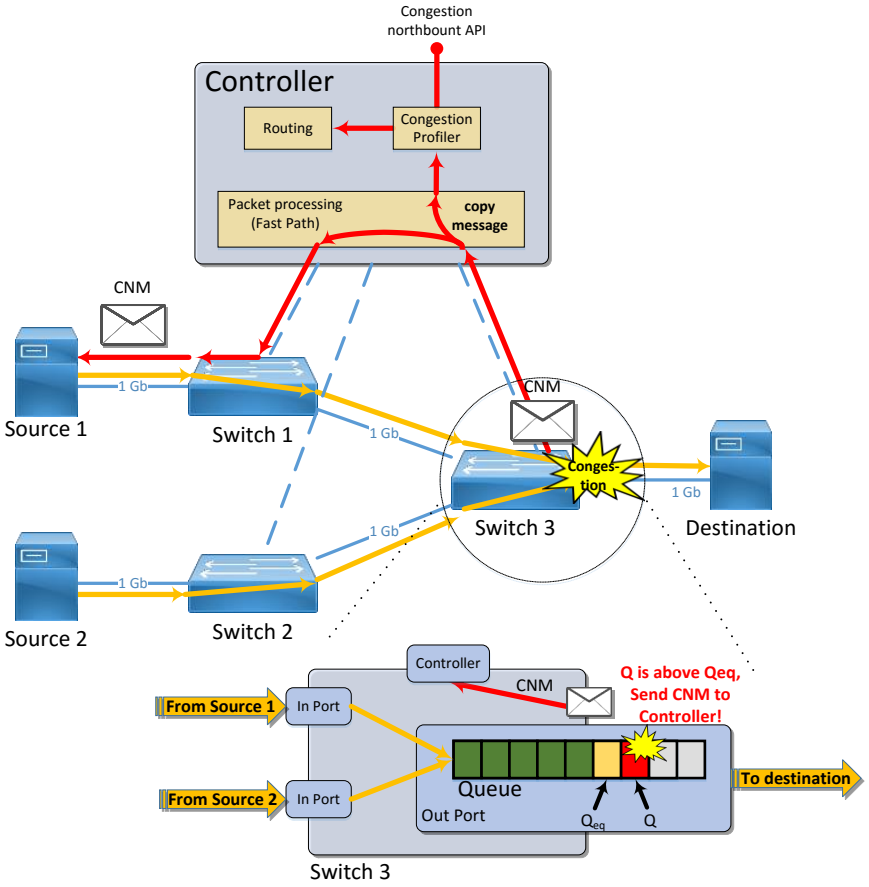


Figura 9: QCN and SDN – Modelul de integrare parțială

Această abordare are mai multe probleme, însă fiecare dintre ele poate fi rezolvată:

- CNM-urile trebuie să ajungă înapoi la sursă (server sau RP) cu întârziere minimă. Orice întârziere va determina acumularea traficului în switch-uri (în punctele de congestie - CP-uri). Acest lucru poate fi o problemă, deoarece controler-ele au întârzieri interne mari datorate căilor lungi de prelucrare software, așa cum am evidențiat anterior (a se vedea 4.3). Prin urmare, pentru a remedia acest aspect, este necesară existența la nivelul controler-ului a unei căi de prelucrare specială, foarte rapidă, astfel încât CNM-urile să poată fi transmise înapoi la sursă cu întârzieri minime.
- În cazul încărcării rețelei, controler-ul poate fi supra-încărcat cu CNM-uri. Acest aspect poate fi rezolvat cu un controler distribuit, în care componentele ce fac expediere CNM-urilor sunt pe noduri separate iar *profiler*-ul de congestie în sine este împărțit în mai multe componente pentru îmbunătățirea paralelismului.
- Mesajele de congestie (CNM-urile) sunt trimise prin rețeaua de management. În cazul în care această rețea este separată fizic de rețeaua de date, atunci poate apărea trafic considerabil peste ea, căci congestiile generează CNM-uri mici, dar suficient de multe încât să sufoce o rețea de management cu capacitate redusă (de exemplu, un *switch* cu multe porturi de 100Gb poate fi conectat la controler printr-un port de capacitate mult mai mică, de 1Gb).

5.1.4 Modelul de integrare completă

Integrarea deplină se îndepărtează de QCN până în punctul în care o implementare care să respecte standardul nici nu mai este necesara. Conceptele QCN rămân încă valabile însă compatibilitatea cu alte soluții nu mai este o necesitate. În acest model, CNM-urile[4] sunt trimise de la switch-uri la controler. *Profiler*-ul de congestie examinează aceste pachete, actualizează costurile legăturilor, trimite evenimente către serviciul de rutare și, dacă este necesar, limitează traficul în *switch*-urile periferice.

Soluția are două probleme majore:

1. Funcționează numai atunci când *switch*-urile periferice sunt switch-uri virtuale. Obiectivul nostru este de a obține un mediu fără pierderi. Prin urmare, atunci când limităm ratele de trafic din nodurile periferice (*switch*-uri virtuale), pachetele care sosesc trebuie să fie păstrate într-o coadă de mari dimensiuni. Această coadă poate stoca sute sau chiar giga octeți de trafic pentru a elimina pierderile de pachete. Toate astea sperând că protocoalele de nivel superior compensează încetinirea traficului..
2. Procesarea pachetelor se face în întregime de către controler. În acest caz, controlerul trebuie să reacționeze rapid, însă în caz de congestie ridicată, supraîncărcarea poate apărea foarte ușor (de exemplu, algoritmul de calcul al rutei minime al lui Djikstra este costisitor pe topologii de mari dimensiuni). Crearea unui *Profiler* de congestie distribuit și inteligent va atenua această problemă, dar crearea unei soluții scalabile pentru întregul Centru de Date este o provocare.

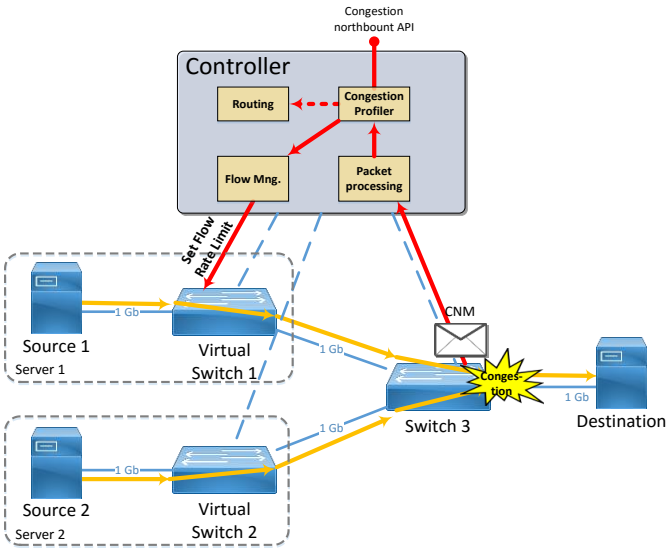


Figura 10: QCN și SDN – Modelul de integrare completă

5.1.5 Concluzii

Avantajul principal al ultimului model, care nu se bazează pe implementarea standardului QCN este umbrită de problemele sale. Acestea îl fac nefuncțional pentru implementările generice din centre de date, dar poate fi singura opțiune în cazul în care mașinile virtuale nu suportă limitatorul de trafic al

[4] Vom defini în continuare mesajele de congestie tot CNM-uri chiar dacă formatul acestor pachete poate fi complet diferit de cel din standardul QCN.

QCN[5]. Prin urmare, pentru a depăși limitările sale, modelul poate fi combinat cu cel de integrare minimală și/sau parțială.

5.2 Studiu de caz: echilibrarea încărcării *(Load Balancing)* bazată pe congestie

În secțiunea anterioară am prezentat modelele de implementare a managementului congestiei în SDN. În această secțiune voi prezenta un algoritm de echilibrare a încărcării traficului între multiple puncte de reacție QCN RPs (servere) într-un sistem de fișiere distribuit (Ceph).

În sistemele de fișiere distribuite, fișierele sunt împărțite în fragmente de date (sau obiecte) ce sunt replicate pe dispozitive de stocare multiple, prin urmare, un fragment sau un obiect poate fi transferat de la orice replică. De exemplu, în Ceph fiecare obiect are un PG (*Placement Group*) asociat. Fiecare PG conține lista de dispozitive de stocare (OSD-uri), care dețin acel obiect[6]. Prin urmare, folosind QCN-WFQR, replica poate fi aleasă în mod dinamic dintre OSD-urile disponibile în PG îmbunătățind astfel capacitatea de transfer a sistemului prin evitarea blocajelor de rețea.

În cazul unui controler centralizat, apare necesitatea unui Profiler de Sistem *(System Profiler)* pentru clienți, pentru a obține datele de care au nevoie. *Profiler*-ul este format din două module (Figura 11):

1. Un *Profiler de Congestie* (*Congestion Profiler*) prezentat în secțiunile anterioare.
2. O aplicație unică pentru fiecare serviciu ce beneficiază de această optimizare (*Service Manager*). Rolul său este de a direcționa clienții către cel mai puțin congestionat RP.

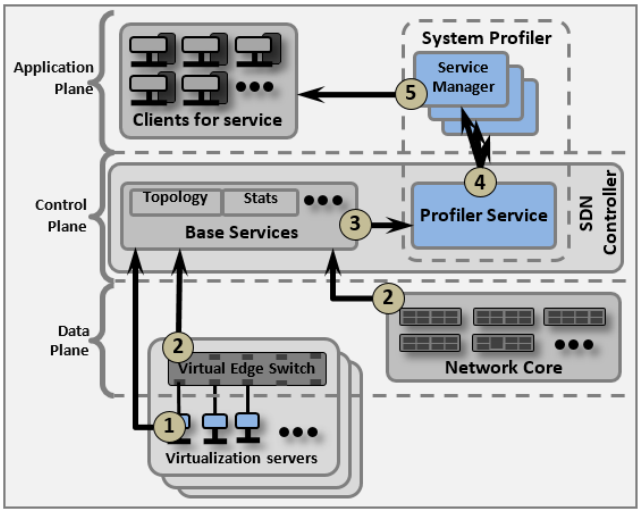


Figura 11: QCN-WFQR SDN echilibrarea traficului

În acest caz, pașii parcurși pentru alegerea celui mai bun RP sunt:

- **Pasul 1:** Serviciile se înregistrează la controler
- **Pasul 2:** Indicativii de congestie locali sunt colectați din *switch*-uri

[5] Mașinile virtuale sunt surse de trafic iar multe sisteme de operare nu suportă QCN pentru interfețe virtuale de rețea.

[6] OSD-ul este un RP din perspectivă QCN

- **Pasul 3:** *Service Profiller*-ul preia indicativii de congestie anterior stocați în bazele de date ale controler-ului pentru a îi calcula pe cei de sistem și pentru a decide, dacă este necesar, să migreze fluxurile existente pe alte rute.
- **Pasul 4:** Fiecare *Service Manager* solicită indicativii de congestie de la *Service Profiller* prin intermediul API-ul *Northbound* al controler-ului *API* și pregătește o listă ordonată de RP-uri bazate pe congestie.
- **Pasul 5:** Clientul solicită unul sau mai multe RP-uri de la *Service Manager*.

5.3 QCN-WFQR simularea migrării traficului în rețele SDN

În contextul SDN, migrarea fluxurilor bazată pe QCN-WFQR a permis obținerea unei încărcări a traficului mai bine echilibrată în topologia ilustrată în Figura 12.

În simularea noastră, topologia furnizează căi alternative astfel încât fiecare dintre cele patru fluxuri din sistem (de la $Flow_1$ până la $Flow_4$) să aibă două căi alternative.

În primele 5 secunde din timpul de simulare, punctul de congestie 3 transmite trei fluxuri (*Flow* 1, 2 și 3), în timp ce punctul de congestie 4 transmite un singur flux ($Flow_4$). Se poate observa din Figura 13 (stânga), că ponderea cozii (*Queue weight*) în CP_3 este semnificativ mai mare decât cea a CP_4, care inițial deservește doar un flux.

După 5 secunde, $Flow_1$ este migrat din punctul de congestie CP_3 către CP_4 de către controler-ul SDN. Figura 13 (dreapta) arată că ponderile cozilor pentru cele două puncte de congestie converg, de asemenea, ratele pentru toate cele patru fluxuri converg - Figura 13 (stânga) - atingând niște rate echitabile și încărcare mai bine echilibrată.

Toate fluxurile încearcă să ajungă la capacitate maximă - conexiunea dintre RP1 și CP1, RP2 și CP2 & CP3 și destinație au o capacitate infinită. Toate celelalte conexiuni au o capacitate limitată de 1.5Gb/s, prin urmare există posibilitatea apariției congestiilor.

Figura 13 (stânga) ilustrează că RP_1 a primit mai multe CNM-uri decât RP_2. La început (înainte de migrarea fluxului) RP_2 recepționează câteva CNM-uri deoarece fluxul 4 (singurul pe cale) încearcă să obțină o capacitate mai mare decât este disponibilă, în timp ce pe cealaltă cale 3 fluxuri concurează în același timp, pentru aceeași conexiune.

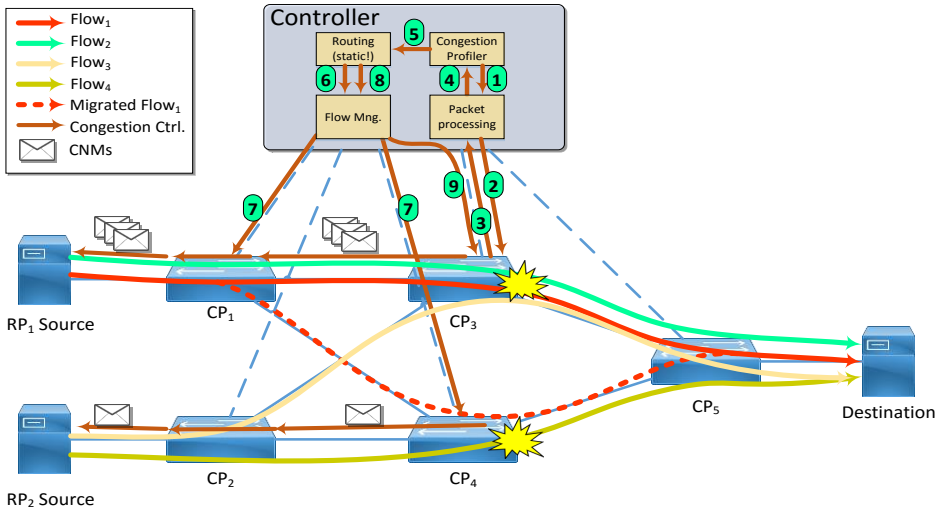


Figura 12: QCN-WFQR Migrarea rutei

Pentru o simulare simplificată, Profiler Service a fost implementat ca o aplicație de sine stătătoare și dovedește faptul că rerutare debitului se poate face în mod eficient cu QCN-WFQR. Nu au fost implementate interfețe *northbound* și nu s-a simulat un algoritm de rutare dinamică – rutele fiind modelate static. Migrarea unui flux către o cale alternativă a fost decisă în timpul rulării pe baza indicativilor de congestie colectați de *Profiler* din toate nodurile[7].

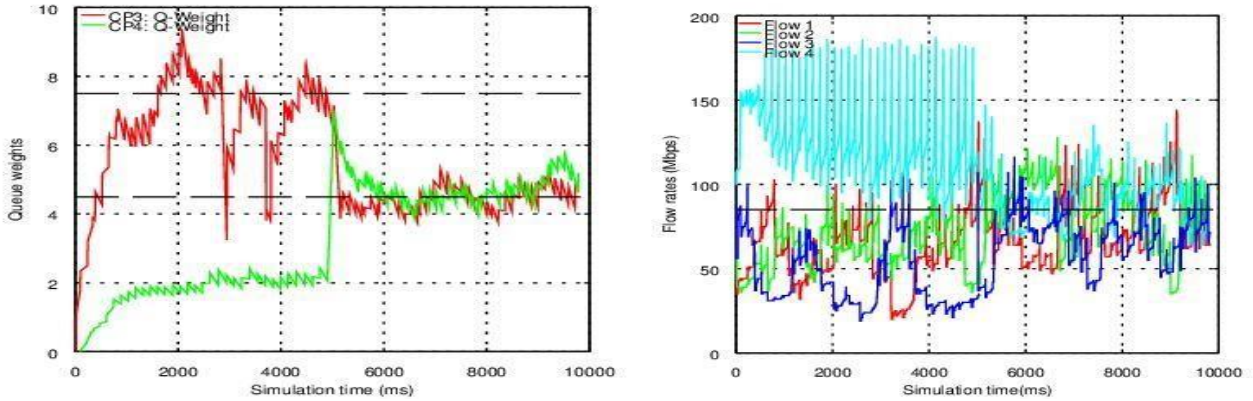


Figura 13: *Stânga* – Ponderile cozilor pentru CP3, CP4. *Dreapta* – variația ratei pentru cele două fluxuri

5.4 Concluzii

Algoritmul QCN *Weighted Flow Queue Ranking* (QCN-WFQR) calculează indicativii de congestie, care sunt măsuri ale încărcării generate de fluxuri în diferite puncte din rețea. Sistemul este capabil de a calcula contribuția unui flux, $Weight_{flow,}$ și contribuția unei surse de congestie (RP), $Weight_{RP}$, la încărcarea rețelei. Pe baza acestor indicativi de congestie clienții pot determina serverul cel mai potrivit (RP) pentru servicii, în scopul de a echilibra încărcarea rețelei.

Simulările arată modul în care un controler de SDN poate beneficia de WFQR când efectuează operații de optimizare a traficului într-un centru de date.

[7] Aceste simulări au fost efectuate înainte să existe suport pentru rutare dinamică în controler-ul SDN prezentat în Cap. 4.

6 VLAN-PSSR: RUTARE SURSA CU COMUTARE DE PORT CE FOLOSEȘTE ETICHETE VLAN IN CENTRE DE DATE SDN

6.1 Introducere

Rutarea Sursă permite unui nod din calea unui pachet să specifice parțial sau complet traseul urmat de acel pachet. Ruta este adăugată în antetul pachetului ca o listă de noduri de traversat, deciziile de direcționare a traficului in rețea fiind fixe (fără algoritmi de rutare) si simple. În acest capitol prezentăm o abordare inedită a Rutării Sursă în SDN, care utilizează etichete VLAN stivuite. Soluția noastră a fost validată în Mininet folosind controler-ul Ryu si s-a dovedit a avea mai multe avantaje față de alte metode de direcționare.

În general în rețelele din Centrele de Date și, în special în SDN, există trei tehnici principale pentru direcționarea pachetelor:

1. **Direcționare pe baza destinației** - pachetele sunt transmise pe baza informațiilor despre *destinație* din antet;
2. **Direcționare pe bază de etichetă** – pachetele care intră în rețea sunt clasificate și o etichetă este atașată fiecărui pachet, acesta fiind direcționat pe baza etichetei (e.g. ATM și MPLS);
3. **Direcționare bazată pe rutare sursă** – pachetele sunt transmise pe baza unei liste de noduri specificate în pachet.

Soluția de rutare sursă propusă se bazează pe etichete VLAN stivuite. Acestea sunt adăugate la pachet la periferia rețelei și sunt scoase la fiecare nod după ce este luată decizia de direcționare. O soluție similară folosind etichete MPLS este prezentată în [9]. Avantajul principal al stivuirii VLAN când o comparăm cu MPLS este faptul că suportul pentru MPLS este limitat în switch-urile din nucleul rețelei, în timp ce VLAN-ul este mult mai frecvent iar majoritatea switch-urilor MPLS suporta doar 3 niveluri de etichete. De asemenea, dimensiunile antetelor sunt mai mici când sunt folosite etichete VLAN - 2 octeți față de 4 octeți pentru MPLS (+2 octeți pentru tipul de antet, în ambele cazuri) **[PONC, 2016/2].**

6.2 Descrierea soluției VLAN-PSSR

În antetul Ethernet, etichetele VLAN stau între adresa MAC a sursei și antetele protocoalelor superioare, de obicei IPv4 sau IPv6. Etichete multiple sunt adăugate la pachet una după alta. În antetul Ethernet, aceste etichete sunt identificate printr-un *Ethertype* de 0x8100, prin urmare, fiecare etichetă conține acești doi octeți. Numai după Ethertype avem informații specifice VLAN:

- *Priority code point* (PCP), un câmp de 3 biți care mapează un pachet la o coadă de prioritate,
- *Drop Eligible Indicator* (DEI) – un câmp de un bit, care indică dacă se renunță la pachetul respectiv în cazul congestiilor și
- VLAN ID (VID) – un câmp de 12 biți care specifică VLAN-ul căruia îi aparține pachetul.

Soluția noastră, VLAN-PSSR refolosește VID pentru rutare sursă. Un bit este folosit pentru a specifica dacă eticheta este *multicast* sau *unicast*. Pentru *unicast* 8 biți specifică numărul de port al unui switch (0 la 255), în timp ce 3 biți nu sunt utilizați (Figura 14).

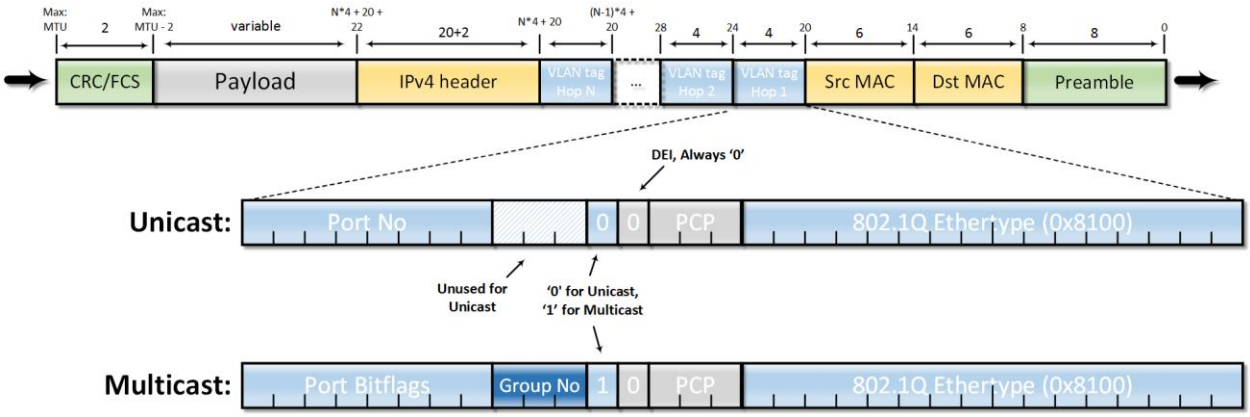


Figura 14: Formatul Pachetului în VLAN-PSSR

Pentru *multicast* vom împărți cei 11 biți ai VID în două părți:

1. *Numărul grupului* - un număr întreg fără semn care indică spre un subset de porturi. Dacă împărțim numărul total de porturi ale unui switch în grupuri, acest număr reprezintă o astfel de diviziune.
2. *Port bitflags* – un set de biți care reprezintă toate porturile dintr-un grup. Fiecare bit corespunde unui singur port. Porturi multiple pot fi selectate în același timp. Prin urmare, un pachet este transmis pe toate porturile care fac parte din acel grup și au biții lor setați la valoarea 1 în *Port bitflags*.

De exemplu, în Figura 14, 3 biți sunt rezervați pentru numărul grupului și 8 pentru *port bitflags*. În această configurație putem defini 8 grupuri, fiecare cu 8 porturi, pentru un total de 64 de porturi.

Rețineți că putem elimina bitul *multicast* și putem considera orice mesaj care are numărul grupului mai mare decât "0" ca fiind *multicast*. În acest caz, putem multiplexa (duplica) un pachet pentru 120 de porturi. În cazul în care avem nevoie de mai multe porturi, putem crește câmpul de grup la 5 biți și putem scădea *port bitflags* la 7 biți, rezultând un număr de 217 porturi. O multiplexare mai mare crește, de asemenea, numărul de etichete necesare. Aceasta duce și la creștea dimensiunii pachetului. Prin urmare, numărul optim de etichete trebuie să fie selectat în funcție de numărul de porturi al *switch*-ului.

Numărul maxim de porturi (*P*) și numărul maxim de grupuri (*G*) pot fi calculate astfel:

$$\begin{cases} P = (2^{Gs} - 1) * Ppg \\ G = 2^{Gs} \\ Gs + Ppg = 11 \end{cases} \quad (6.1)$$

Unde *Gs* este numărul de biți rezervat pentru numărul grupului, *Ppg* este numărul de biți din *Port bitflag* (i.e. porturi per grup) și 11 este numărul de biți din VID.

6.3 Validare funcţională în Mininet

Pentru a valida modelul nostru l-am implementat întâi în Mininet [10], cu o configurație mică (Figura 15) și am verificat că *ping*-ul funcționează iar conexiunile UDP și TCP pot fi stabilite cu succes. Apoi am validat conținutul mesajelor folosind Wireshark [11].

6.3.1 Soluția Unicast

Configurarea Mininet constă din trei gazde H_1, H_2, H_3 și 5 switch-uri: S_1, S_2 și S_3 la periferie și C_{11} și C_{12} în centru. Gazdele sunt instanțe de containere Linux[8] și *switch*-urile sunt instanțe Open vSwitch (*bridges*) conectate la controler-ul Ryu. Fluxurile sunt apoi gestionate de către VLAN-PSSR implementat peste Ryu.

Pentru o identificare mai ușoară am configurat atât adresa MAC Ethernet a gazdelor (*host*-urilor) cât și ID-urile *DataPath*-ului OpenFlow (dpid) la numărul host-ului/*switch*-ului (1, 2, 3, 11 = 0xb & 12 = 0xc). Comunicarea între H_2 și H_3 utilizează direcționare pe bază de destinație, fără rutare sursă, fiind la doar un salt distanță, în timp ce comunicarea între H1 ⇔ H2 și H1⇔ H3 utilizează rutare sursă. Configurația pe care s-a făcut validarea este prezentată în Figura 15.

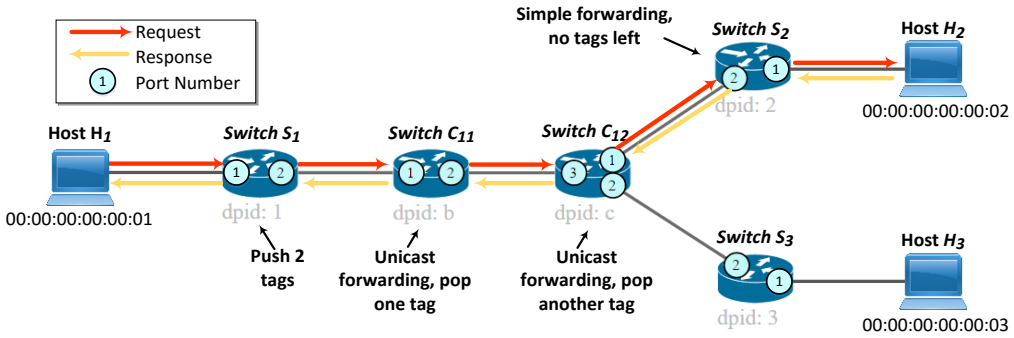


Figura 15: VLAN-PSSR Configurație validare soluție *unicast*

Etichetele PSSR sunt adăugate în switch-urile periferice (S_1, S_2 și S_3) și scoase (*popped*) în *switch*-urile centrale. Ca exemplu, în Figura 15, avem o secvență *cerere-răspuns* între sursa H_1 și destinația H_2. În acest caz două etichete sunt adăugate în S_1 și scoase, una câte una, în C_{11} și C_{12}.

6.3.2 Soluția Multicast

Pentru VLAN-PSSR pachetele *multicast* sunt transmise pe o cale *unicast* până la penultimul salt, unde pachetele sunt multiplicate și trimise la ultimul salt pentru direcționarea finală. Motivul este faptul că VLAN-PSSR poate face doar o singură multiplicare și aceasta trebuie să fie aproape de destinația pachetelor. În centrele de date, de obicei, penultimul salt este *switch*-ul din vârful Rack-ului (*Top of Rack* - ToR), în timp ce ultimul salt este *switch*-ul virtual din servere. Prin urmare, soluția noastră furnizează *multicast* în interiorul unui singur rack, dar, din moment ce vizează centrele de date *multitenant* cu *switch*-uri virtuale la periferie, *multicasting-ul* în interiorul aceluiași rack reprezintă majoritatea cazurilor de utilizare. Domeniile de *broadcast* sunt de obicei mici în Centrele de Date *multitenant* cu doar câteva mașini virtuale conectate la același domeniu (în jur de 10 - 20), care, pentru a reduce utilizarea lățimii de bandă a rețelei centrale, sunt rulate pe servere apropiate, rareori întinzându-se pe mai multe rack-uri.

Pentru *multicasting* între *rack*-uri este necesară trimiterea a câte unui flux către fiecare *rack* unde se dorește multiplicarea pachetelor. Această metodă crește traficul în rețea dar este o soluție mult mai bună decât trimiterea câte unui flux separat pentru fiecare destinație în parte.

[8] Aceasta este o soluție de virtualizare la nivel de sistem de operare furnizată de nucleul Linux. Ea oferă izolarea logică a resurselor de *proces*, *rețea* și *sisteme de fișiere* între containere, astfel încât un container să fie izolat de resursele altui container.

Pentru validarea abordării noastre, am folosit aceeași configurație ca și mai înainte, la soluția *unicast*. Diferența în acest caz este că un flux *multicast* care pleacă din H_1 și este trimis atât către H_2 cât și către H_3 (Figura 16). De la H_1 la C_{12} pachetele sunt transmise folosind *unicast* iar multiplicarea se face prin C_{12}.

În *switch*-ul S_1 se adaugă atât etichete *unicast* cât și *multicast*, mai întâi cele *unicast* apoi *multicast*, astfel încât, atunci când pachetul ajunge la C_{12}, acesta are numai etichete *multicast*. Apoi C_{12} multiplică pachetul și îl transmite către ambele switch-uri de la periferie. Este de remarcat faptul că C_{12} nu poate să scoată etichetele *multicast* din pachete astfel încât *switch*-urile de la periferie trebuie să scoată toate etichetele rămase înainte de a trimite pachetul către gazda destinație[9].

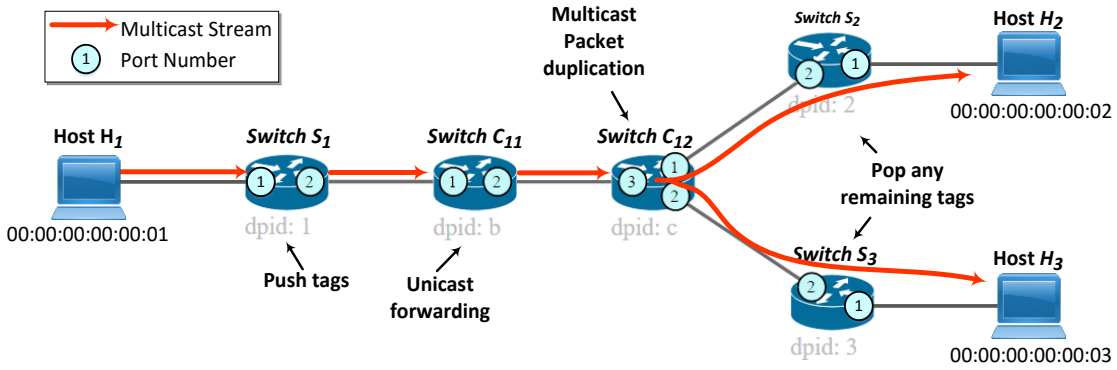


Figura 16: VLAN-PSSR în configurație Multicast

Tabelele de flux din nodurile ce fac direcționare *multicast* sunt complexe (Figura 17). Prelucrarea se face într-un *pipeline* și fiecare trecere procesează o singură etichetă *multicast*, prin urmare, în cazul în care mai multe etichete sunt prezente, sunt necesare mai multe treceri ale aceluiași pachet prin *pipeline*, ceea ce scade performanța atunci când soluția este utilizată în software.

În Figura 17 prelucrarea are următoarele etape (P este numărul de porturi dintr-un grup iar G numărul de grupuri):

1. În **tabela #0,** pachetele cu eticheta *multicast* sunt identificate. În cazul în care pachetul este etichetat *multicast* procesarea continuă în tabela 10.
2. În **tabela #10** pachetul este identificat de o singură intrare și trimis la un port altfel este trimis la tabelul următor.
3. De la **tabela #11 la #10 + (P-1)** pachetul este împerecheat cu un alt grup/port până la capătului *pipeline-ului*.
4. În tabelul **#10 + P**, în cazul în care pachetul conține încă etichete *multicast* este trimis înapoi la începutul *pipeline-ului* pentru a procesa încă o etichetă în caz contrar este considerat procesat și șters (*dropped*).

[9] Acest lucru este valabil numai pentru OpenFlow 1.3. Versiunile mai noi au două pipeline-uri, unul pe *ingress,* care procesează pachetele atunci când intră în *switch* și alta pe *egress*, care procesează pachetele după ce pachetul a fost direcționat către portul de ieșire. Prin urmare, multiplicarea se face pe *ingress* și eliminarea etichetelor pe *egress*. Am folosit capabilitățile versiunii 1.3, deoarece aceasta este mai răspândită.

Dimensiunea tabelului de fluxuri (*Tsize*) la penultimul salt (i.e. în *switch*-ul din vârful *rack*-ului) este proporțională cu numărul de grupuri utilizate (*Gu*) din totalul de grupuri existente (*G*), cu porturile per grup (*Ppg*) și cu numărul total de porturi ale respectivului *switch (P):*

$$Gu = \left\lceil \frac{P}{Ppg} \right\rceil \quad (6.2)$$

$$Tsize = 3 + (Gu + 1) * Ppg \quad (6.3)$$

$$Ntables = Gu + 2 \quad (6.4)$$

Prin urmare, pentru un *switch* din vârful rack-ului (*Top of rack* – ToR) cu 64 de porturi, avem $Ppg = 8$, și utilizarea tablei de fluxuri este de $Tsize = 75$, o utilizare foarte mică comparat cu cea normală, de câteva mii de fluxuri.

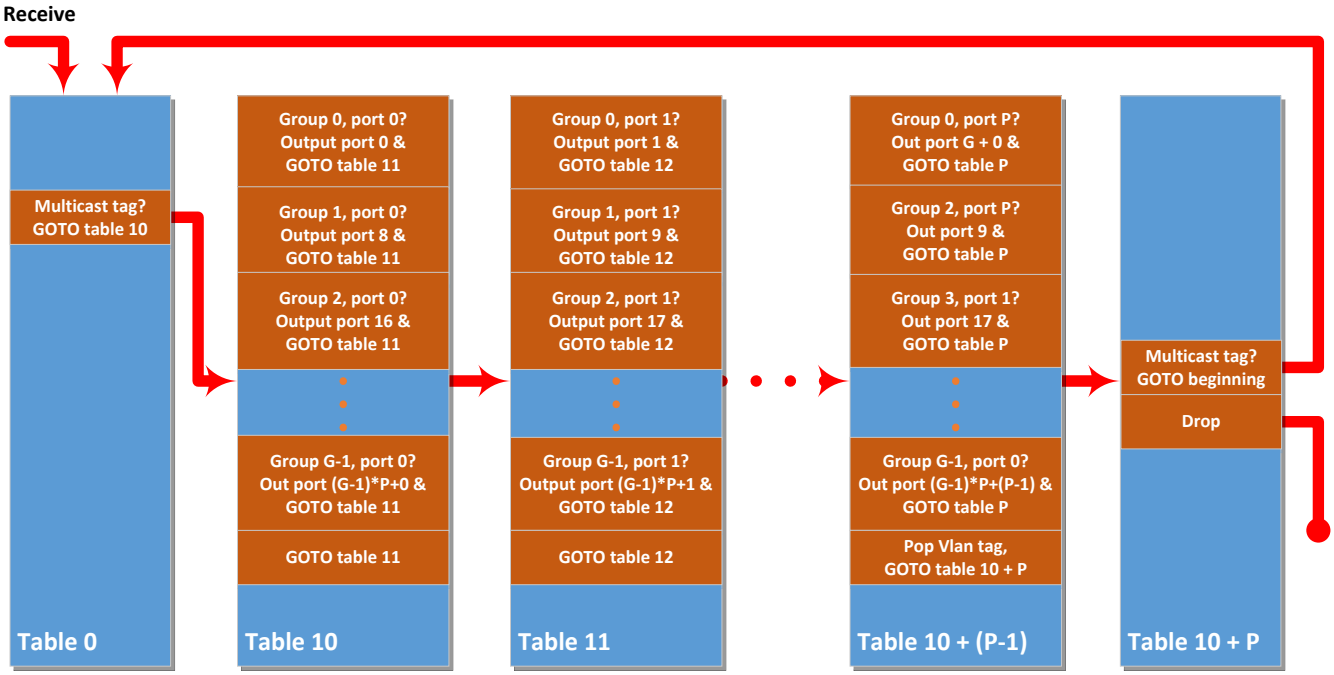


Figura 17: Intrări de flux *multicast* in switch-uri centrale

6.4 Evaluarea performanței VLAN-PSSR folosind Open vSwitch pe servere Xeon

Pentru a înțelege mai bine impactul soluției noastre asupra implementărilor din lumea reală am măsurat utilizarea procesorului când rulează Open vSwitch, în condițiile în care adăugăm etichete dar și în condiții de trafic *multicasting.* Evaluarea performanțelor a fost realizată pe un server al unui centru de date real. Acest server are un procesor din clasa Xeon, Intel Xeon E5-2670 la 2.60GHz cu 16 nuclee.

Am conectat server la un al doilea sistem pentru generarea de trafic (acesta are un procesor Intel Core i7-4820K la 3.70GHz cu 4 nuclee). Conectarea a fost realizată pe fibră optică folosind două interfețe de rețea de 10 Gbps. Ambele sisteme au rulat Ubuntu Linux 14.04 LTS și au fost administrate de la distanță prin conexiuni *ssh* (*Secure Shell*), de la un terminal de management (i.e. laptop-urile noastre).

Pentru a aduna statistici cât mai corecte: (1) am dezactivat toate serviciile neesențiale, (2) am forțat planificatorul de procese al Linux-ului să planifice procesele doar pe un singur nucleu și (3) am mutat toate procesele rămase, care nu fac parte din Open vSwitch, cum ar fi serverul *ssh* si agentul de monitorizare, pe un alt nucleu (nucleul 9). De asemenea, pentru a evita impactul cache *threashing*

generat de aceste procese neesențiale am ales ca acest nucleu să fie pe un alt nod al arhitecturii NUMA [12][10].

Două seturi de măsurători au fost executate:

1. **Am măsurat impactul adăugării de etichete VLAN comparat cu un flux fără etichete.** Am adăugat de la 1 la 64 de etichete și am măsurat utilizarea procesorului. Am avut grija ca deviația standard sa nu influențeze rezultatele.
2. **Am măsurat direcționarea (multiplicarea) *multicast* de la două porturi până la 64.**

În Figura 18 observăm că utilizarea crește liniar cu numărul de etichete, dar crește exponențial cu rata pachetelor de date. Acest lucru este cauzat de supraîncărcarea întreruperilor hardware (i.e. sunt mult prea multe și nu pot fi prelucrate hardware) și de preluarea lor în software. Prelucrarea în software a întreruperilor duce la scăderea performanței și se întâmplă pentru că măsurăm doar un singur nucleu ce poate prelucra un număr limitat de întreruperi. Dacă înlăturăm limitarea la un singur nucleu, utilizarea procesorului scade[11]. Dar, din moment ce Open vSwitch este *multithreaded*, și se poate scala cu ușurință putem extrapola rezultatele obținute și să concluzionam că soluția noastră poate direcționa ușor pachete folosind întreaga lățime de bandă a unei conexiuni de 10Gbps. Acest lucru dovedește că soluția noastră de VLAN-PSSR este fezabilă.

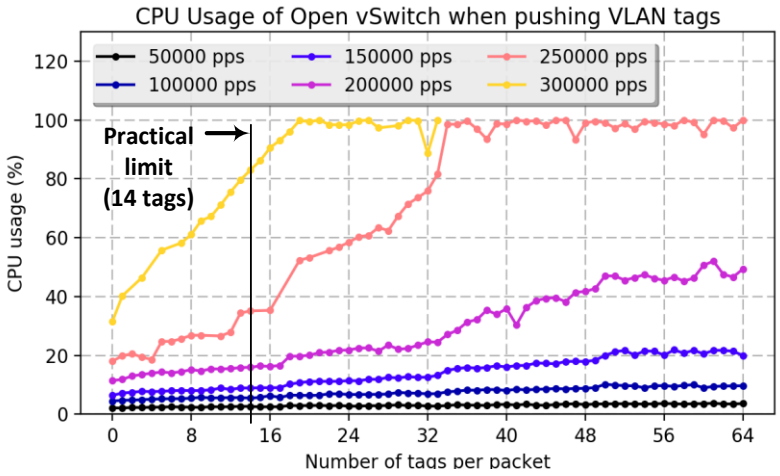


Figura 18: Performanța când sunt adăugate etichete într-un singur nucleu al CPU

De asemenea, adăugarea a 64 de etichete este un scenariu nerealist de utilizare, dat fiind faptul că, într-un centru de date de obicei un pachet este rutat prin nu mai mult de 5 - 6 salturi. Acest lucru înseamnă că pentru *unicast* numărul maxim de etichete ar fi 6. Pentru 64 de porturi *multicast*, în aceleași condiții, s-ar adăuga încă 8 etichete ajungându-se la un maxim de 14. În acest caz, rata de pachete poate ajunge la maxim 300 mii de pps/nucleu, o valoare bună. Iar fără supraîncărcarea întreruperilor rezistă la în jur de 200 de mii de pps/nucleu. Prin urmare, pe un sistem de 16 nuclee, cu o utilizare a procesorului sub 20% pe toate nucleele, se poate ajunge cu ușurință la un maxim de peste

[10] Primele 8 nuclee sunt pe primul nod NUMA în timp ce celelalte 8 sunt pe al doilea nod, astfel am ales nucleul 0 pentru Open vSwitch și 9 pentru toate celelalte procese.

[11] Dar, fără izolarea la un singur nucleu, nu mai reușim să măsurăm performanța în mod corect datorită interferențelor de la celelalte procese.

3,2 milioane de pps. Dacă înmulțim această valoare cu 512B (dimensiunea medie a pachetului), ajungem la 11Gbps, peste pragul de 10Gbps al unei singure conexiuni și asta la 20% utilizare.

În cazul *multicast*, în Figura 19, utilizarea procesorului este mult mai mare. Pentru doar 30 de mii de pachete se utilizează în jur de 30% dintr-un singur nucleu. Cauza principală este faptul că pachetele cu mai multe etichete se deplasează de mai multe ori prin același *pipeline* (o singură trecere per etichetă). Dimensiunea pachetului nu are un impact notabil asupra performanței pentru că procesarea se face folosind *zero copy buffers*. În timpul transmisiei se face o singură copie, care utilizează acces direct la memorie (DMA).

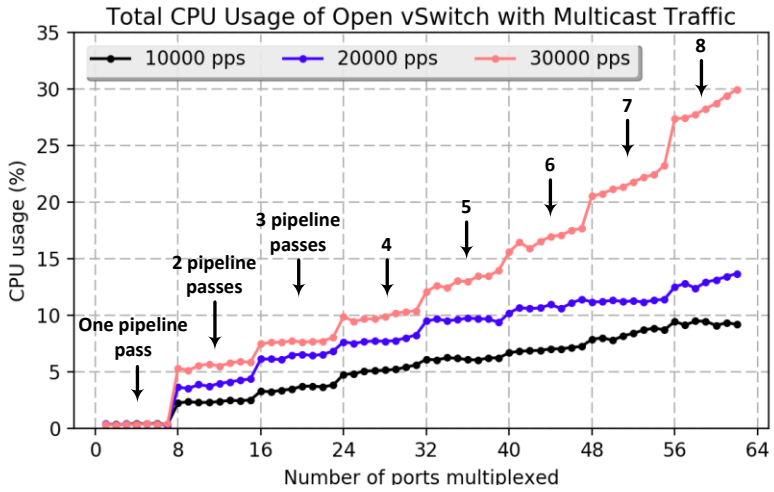


Figura 19: Performanța Multicast

După cum ne așteptam, aceste rezultate dovedesc că *multicasting*-ul cu rutare sursă nu este o soluție bună pentru fluxurile cu trafic mare atunci când se realizează în software[12]. Prin urmare, este necesară o accelerare hardware folosind circuite specializate (ASIC-uri).

6.5 Concluzii

Rutarea sursă bazată pe etichete VLAN poate fi implementată cu ușurință folosind funcționalitatea OpenFlow 1.3 existentă. Această soluție poate fi folosită într-un întreg Centru de Date, simplificând astfel tabelele de fluxuri din switch-urile centrale sau, în cazul în care dimensiunea pachetului reprezintă o problemă, activarea acestei funcționalități se poate face numai în cazul în care numărul de fluxuri din *switch*-uri se apropie de capacitatea maximă a acestora (numărul de flux-uri suportat în hardware este limitat).

Acest lucru poate fi de asemenea utilizat pentru a îmbunătăți rutarea *multipath*, căci pachetele pot fi dirijate de la sursă pe căi diferite folosind algoritmi de distribuție fină, iar switch-urile din nucleu nu vor fi conștiente de acest mecanism.

De asemenea, având în vedere faptul că tabelele de fluxuri au dimensiune fixă în VLAN-PSSR, se poate dezvolta un hardware mult mai simplu și rapid. Acest lucru ar reprezenta o reducere impresionantă a costului fiecărui switch.

[12] Soluția poate fi utilizată totuși pentru a izola domenii de broadcast mici (i.e. pentru un singur *tenant*), atunci când traficul este neglijabil.

7 CONCLUZII ŞI DEZVOLTĂRI VIITOARE

7.1 Contribuţiile originale ale tezei

Această teză aduce următoarele contribuții originale:

Contribuţii la analiza SDN (capitolul 2)

Pornind de la documentația existentă în domeniu, am propus un set relevant de cerințe și am evidențiat avantajele rețelelor definite software în comparație cu rețelele tradiționale (secțiunea 2.3). De asemenea, am analizat activitatea celor mai importante zece proiecte de controler-e SDN *open source* (secțiunea 2.4), pe baza numărului de contribuții de cod pe lună.

Contribuţii la analiza Centrelor de Date (capitolul 3)

În acest capitol am propus un set cuprinzător de criterii de clasificare a centrelor de date, care ia în considerare opt caracteristici importante: disponibilitate, scop, modelul de orchestrare, model de servire, implementare, topologia rețelei, modelul de stocare și modelul de calcul (secțiunea 3.2). Am analizat modul în care suntem restricționați în alegerea topologiei, în cele mai multe cazuri, la un singur tip: topologie *Clos* cu două dintre variantele sale: *Fat-tree* și *Leaf-spine*. Mai mult decât atât, am comparat din punct de vedere al înălțimii și al diametrului cele două topologii cu încă două, care sunt mai puțin utilizate, *torus* și *hypercube*, și, pe baza acestei analize, am prezentat recomandări privind cele mai bune scenarii de utilizare pentru fiecare dintre ele (secțiunea 3.4) [ANDR, 2015]. O exemplificare a unei topologii *Fat-tree* optimizată pentru aplicații de e-learning a fost prezentată în [PIST, 2014].

Cea mai mare parte a traficului Centrului de Date este creat de serverele de stocare și de cele de procesare sau provine din lumea exterioară. Pentru o mai bună înțelegere a traficului de stocare am măsurat traficul generat de o soluție de stocare distribuită bazată pe Ceph, am analizat impactul acesteia asupra rețelelor și am oferit recomandări pentru scalarea capacității conexiunilor către serverele de stocare. Am propus un set comprehensiv de criterii de clasificare a traficului din servere și a traficului exterior într-un Centru de Date (secțiunea 3.3). De asemenea, am analizat modelul de comunicare al OpenStack-ului, o platforma de management a *Cloud*-urilor bazată pe microservicii, ajungând la concluzia că cea mai mare parte a traficului său este compus din pachete mici, care ar putea beneficia de căi rapide cu întârzieri scurte și puține pierderi de pachete.

Proiectarea şi implementarea controler-ului Contron pentru simulatorul NS-3 (capitolul 4)

Principala contribuție a acestui capitol este un nou design de controler adaptat pentru simulatoare de rețea. Deși există multe controlere care pot fi utilizate fie pentru cercetarea de noi caracteristici SDN (e.g. Ryu sau Pox), fie pentru implementarea în producție (e.g. Onos sau OpenDaylight), nu există nici o implementare pentru un simulator de rețea Open Source. Acest lucru s-a dovedit a fi o problemă pentru că atât configurațiile fizice cât și cele emulate nu sunt în măsură să ofere un mediu controlat, în care să putem reproduce în mod fiabil aceleași condiții care ar genera un anumit set de rezultate. Mai mult decât atât, complexitatea software-ului în implementări reale, deficitul de hardware și lipsa de fidelitate a performanței în medii emulate (virtuale) agravează problema.

Prin urmare, pentru a rezolva această problemă, am propus un nou controler OpenFlow integrat cu NS-3, care este simplu, modular și care, prin intermediul abstractizărilor și serviciilor

sale de administrare a switch-urilor OpenFlow, oferă un mediu controlat pentru experimentare și cercetare.

Acest nou controler a fost utilizat în cercetarea mea asupra congestiilor și, în afară de acest lucru, reprezintă o platformă bună pentru validarea soluțiilor în domenii SDN, cum ar fi: scalabilitate, încărcare eficientă, *service-chaining*, calitatea serviciilor, rutare *multipath* și integrare cu rețele clasice.

Contron oferă o viziune globală și centralizată a topologiei rețelei ca un grafic de noduri. El are un serviciu ce permite *Descoperirea Topologiei* care populează această topologie cu gazde și *switch*-uri, o componentă de *Abstractizare a Fluxurilor*, care permite adăugarea sau eliminarea cu ușurință a fluxurilor, o componentă pentru *Supravegherea Conexiunilor* care monitorizează conexiunile între gazde, un serviciu de *Statistici* pentru monitorizarea *switch*-urilor, un *Proxy* pentru a gestiona cererile și răspunsurile ARP, precum și o componentă ce realizează *Rutarea Shortest Path* care utilizează algoritmul pentru obținerea drumului minim al lui Dijkstra pentru conectarea nodurilor cu rute definite prin fluxuri OpenFlow (secțiunea 4.1).

În continuare, folosind Contron am realizat un studiu de caz asupra migrării traficului în afara conexiunilor congestionate într-o topologie de tip *Leaf-spine* (secțiunea 4.2 și 4.3). Am remarcat cele mai importante efecte: (1) rezolvarea congestiei pe o conexiune tinde să mute congestia pe următoarea conexiune din cale, (2) multe rearanjari de pachete și (3), timpi de reacție mari ai planului de control. Am propus apoi trei soluții la aceste probleme: creșterea capacității de reacție a planului de control, creșterea dimensiunii cozilor in switch-uri sau folosirea unui mecanism de notificare a congestiei (secțiunea 4.4). Ultima soluție este detaliată în capitolul 5.

Rezultatele cercetării descrise în acest capitol au fost publicate în [PONC, 2016/1].

Reducerea congestiei rețelelor din Centre de Date folosind indicativii SDN, QCN și WFQR (capitolul 5)

În acest capitol propun câteva metode de reducere a congestiei rețelei în centrele de date prin adaptarea *Quantized Congestion Notification* (QCN) și a indicativilor *Weighted Flow Queue Ranking* (WFQR) la SDN [PIST, 2015].

QCN reduce congestiile prin monitorizarea utilizării cozilor la transmisie pe porturile unui switch și, în cazul în care utilizarea depășește o anumită limită, transmisia surselor este încetinită pentru a evita depășirea capacității maxime a cozii și pierderea de pachete. Pe lângă adaptarea QCN-ului la SDN, propunerea mea îmbunătățește mecanismul original QCN prin adăugarea în controlere a posibilității de a migra fluxurile spre căi mai puțin aglomerate, îmbunătățind astfel viteza de transmisie (secțiunea 5.3). Mai mult decât atât, propun o metodă de echilibrare a încărcării (*load-balancing*) pe baza informațiilor legate de congestie, care crește performanța rețelei prin oferirea acestor indicativi aplicațiilor SDN (secțiunea 5.2).

În secțiunea 5.1, propun trei modele de adaptare a comunicării QCN la SDN care au avantaje și dezavantaje, în funcție de capacitățile existente ale controler-ului, performanța dorită și efortul de dezvoltare necesare.

Primul model oferă integrare minimă (secțiunea 5.1.1). Protocolul QCN este păstrat și îmbunătățit prin calcularea indicativilor de congestie în switch-uri și prin transmiterea lor către controler-e. Avantajul său principal este folosirea mecanismelor validate și standardizate ale QCN, prin urmare este necesar un efort minim de integrare. Soluția are trei dezavantaje: în primul rând, software-ul în switch-uri trebuie modificat pentru a păstra indicativii de congestie; în al

doilea rând, interfața *southbound* trebuie să fie actualizată pentru a transmite acești indicativi la controler (e.g. în OpenFlow); în al treilea rând, deciziile în timp real sunt mai greu de luat, deoarece indicativii de congestie sunt calculaţi de switch-uri și sunt recepționaţi de către controler-e cu o anumită întârziere.

Al doilea model adaugă capabilități de decizie în timp real și elimină necesitatea de a face modificări la interfața *southbound* prin transmiterea notificărilor de congestie direct către controlere ca mesaje de tip *packet-in*. Cu toate acestea, el are propriile sale limitări: controler-ul poate întârzia mesajele de congestie, poate fi supraîncărcat de ele și poate congestiona rețeaua de management. Din fericire, aceste probleme pot fi evitate cu un design atent (secțiunea 5.1.3).

Al treilea model oferă integrare completă și se distanțează de QCN până la punctul în care implementarea sa în conformitate cu standardul nu mai este necesară. Modelul are două dezavantaje: funcționează numai când la periferia rețelei sunt folosite switch-uri virtuale și necesită eforturi mult mai mari de proiectare și implementare, deoarece pachetele sunt procesate în întregime de către controler (secțiunea 5.1.4).

Aceste rezultate au fost publicate în [PIST, 2015].

O soluție Rutare Sursă în SDN folosind etichete VLAN stivuite: VLAN-PPSR (capitolul 6)

În acest capitol propun o Soluție de Rutare și de redirectare a pachetelor atât pentru *unicast* cât și pentru *multicast*. Această soluție reduce utilizarea tabelelor de flux în switch-urile centrale, îmbunătățește viteza de actualizare a rețelei și simplifică rutarea multi-cale (*multipath*). În plus, propunerea poate fi pusă în aplicare cu dispozitive OpenFlow v1.3 existente, fără a fi nevoie de modificări costisitoare în *switch*-uri.

Soluția VLAN-PSSR se bazează pe etichete VLAN stivuite care sunt adăugate la margine și eliminate la fiecare salt după ce directarea pachetelor a fost decisă. Acest lucru oferă unele avantaje față de soluțiile similare implementate folosind etichete MPLS, cum ar fi antet (*header*) mai mic și evitarea suportului limitat al MPLS în switch-urile centrale (secțiunea 6.3).

Chiar dacă *multicast*-ul este limitat la un singur rack, acesta acoperă în continuare cele mai multe cazuri de utilizare a Centrelor de date definite software ce suportă mai mulți utilizatori (*multitenant*) și, pentru cazurile neacoperite, prezint două soluții indirecte, care pot fi aplicate întregului centru de date (secțiunea 6.3.2).

Atât soluția *unicast* (secțiunea 6.3.1) cât și cea de *multicast* (secțiunea 6.3.2) s-au dovedit a fi funcționale. Le-am validat pe ambele prin conectarea topologiilor *Open vSwitch* create în Mininet cu controler-ul Ryu și gestionate prin implementarea mea VLAN-PSSR.

Performanța s-a dovedit a fi foarte bună pentru adăugarea de etichete pe hardware real (servere clasa Xeon), cu o utilizare în mică măsură a procesorului chiar și în cele mai rele scenarii (adăugarea a 64 etichete VLAN). În cazul direcționării *multicast*, performanța este mai mică decât ceea ce se dorește, și demonstrează că accelerarea hardware furnizată de ASIC-uri este o necesitate pentru SDDCs (secțiunea 6.4).

Rezultatele au fost publicate în [PONC, 2016/2].

7.2 Activităţi viitoare

Cercetarea mea în simularea rețelelor SDN s-a concentrat pe implementarea unui model de controler de bază, dar funcțional, folosind implementarea NS-3 OpenFlow existentă. Acest model poate fi îmbunătățit. În primul rând, versiunea OpenFlow în NS-3 ar trebui actualizată de la 0.98 până la cel

puțin 1.3, apoi Contron în sine ar trebui extins cu funcționalitățile noii versiuni de OpenFlow, iar caracteristicile sale existente îmbunătățite.

Cu controlerul îmbunătățit și folosind rezultatele obținute în această teză putem cerceta o soluție de rutare multi-cale *end-to-end*. Aceasta ar combina mai multe metrici, cum ar fi: capacitatea conexiunii, debit mediu și informații despre congestii cu caracteristici de clasificare a fluxului de trafic pentru a oferi o soluție de rutare complet automatizată și mai bine optimizată pentru un întreg centru de date. Algoritmul ar fi în măsură să aleagă cea mai bună cale pentru noi fluxuri, să redirecționeze fluxurile existente pentru a evita congestiile și să optimizeze tranzitul și latențele.

Apoi, această soluție de rutare poate fi implementată în producție, pe ONOS sau OpenDaylight, pentru a furniza rutare optimizată. Mai mult decât atât, pentru a reduce timpul rerutării multor fluxuri și pentru a reduce utilizarea tabelului de fluxuri, algoritmul de rutare multi-cale (*multipath*) poate fi extins pentru a utiliza rutarea sursă VLAN-PSSR. Acest lucru ar face posibilă o rutare unde traficul ar fi distribuit optim pe mai multe căi încă de la periferie.

LISTA DE PUBLICAȚII

Publicații în conexiune cu această teză:

[PONC, 2016/1] Poncea, Ovidiu Mihai, Andrei Pistirica, Florica Moldoveanu, Victor Asavei. "Design and Implementation of an Openflow SDN Controler in NS-3 Discrete-Event Network Simulator*", International Journal of High Performance Computing and Networking (IJHPCN)*, accepted for publication. (Scopus/Elsevier, ACM digital Library)

[PONC, 2016/2] Poncea, Ovidiu Mihai, Florica Moldoveanu, Victor Asavei. "VLAN-PSSR: Port-Switched based Source Routing using VLAN tags in SDN Data Centers". Scientific Bulletin of UPB, C series - Electrical Engineering and Computer Science, accepted for publication. (B+)

[PIST, 2015] Pistirica, Sorin Andrei, Ovidiu Poncea, and Mihai Claudiu Caraman. "QCN Based Dynamically Load Balancing: QCN Weighted Flow Queue Ranking." In *2015 20th International Conference on Control Systems and Computer Science*, pp. 197-204. IEEE, 2015. (ISI)

[ANDR, 2015] Andrus, Bogdan, Ovidiu Mihai Poncea, JJ Vegas Olmos, and I. Tafur Monroy. "Performance evaluation of two highly interconnected Data Center networks." In *2015 17th International Conference on Transparent Optical Networks (ICTON)*, pp. 1-4. IEEE, 2015. (ISI)

[PIST, 2014] Pistirica, Sorin Andrei, Ovidiu Mihai Poncea, Victor Asavei, and Alexandru Egner. "IMPACT OF DISTRIBUTED FILE SYSTEMS AND COMPUTER NETWORK TECHNOLOGIES IN ELEARNING ENVIRONMENTS." In *The International Scientific Conference eLearning and Software for Education*, vol. 4, pp. 85. "Carol I" National Defence University, 2014. (ISI)

Alte publicații:

[PONC, 2011] Poncea, Ovidiu Mihai. "Integrarea Multinivel a Informațiilor de Rutare si Tactice in Rețele Ad-hoc de Capacitate Redusă". Sesiunea de comunicări științifice cu tema Cercetarea științifică militară în sprijinul interoperabilității, ACTTM 2011.

[SIM, 2015] Simion, Andrei, Victor Asavei, Sorin Andrei Pistirica, and Ovidiu Poncea. "Practical GPU and Voxel-Based Indirect Illumination for Real Time Computer Games." *In 2015 20th International Conference on Control Systems and Computer Science*, pp. 379-384. IEEE, 2015 (ISI)

[GREU, 2012] Greu, Victor, Petrica Ciotirnae, Constantin Vizitiu, Sorin Cernea, and Ovidiu Poncea. "A secure routing algorithm with additional cognitive information scalable features for the design approach of the tactical frequency hopping radios Ad-hoc networks (TAFHNET)." In *2012 9th International Conference on Communications (COMM)*. pp. 181 – 184, IEEE 2012 (ISI)

BIBLIOGRAFIE

[1] "Internet Users," [Online]. Available: http://www.internetlivestats.com/internet-users/. [Accessed 28 February 2016].

[2] Cisco, "Visual Networking Index," 2015. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html. [Accessed 15 February 2016].

[3] Open Networking Foundation, "Software-Defined Networking: The new Norm for Networks," Open Networking Foundation, 13 04 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf. [Accessed 23 02 2016].

[4] B. Andrus, O. M. Poncea, J. V. Olmos and I. T. Monroy, "Performance evaluation of two highly interconnected Data Center networks," in *17th International Conference on Transparent Optical Networks (ICTON)*, Budapest, Hungary, 2015.

[5] NS-3 Consortium, "NS-3 Tracing," [Online]. Available: https://www.nsnam.org/docs/manual/html/tracing.html. [Accessed 23 02 2016].

[6] A. Vishwanath, V. Sivaraman and M. Thottan, "Perspectives on router buffer sizing: recent results and open problems," *ACM SIGCOMM Computer Communication Review,* vol. 39, no. 2, pp. 24-29, 2009.

[7] J. Warner, "Size of switch buffers," [Online]. Available: https://people.ucsc.edu/~warner/buffer.html. [Accessed 28 02 2016].

[8] IEEE, "802.1Qau - Congestion Notification," [Online]. Available: http://www.ieee802.org/1/pages/802.1au.html. [Accessed 23 02 2016].

[9] M. Soliman, B. Nandy, I. Lambadaris and P. Ashwood-Smith, "Source routed forwarding with software defined control, considerations and implications," in *Proceedings of the 2012 ACM conference on CoNEXT student workshop*, Nice, France, 2012.

[10] "Mininet: An Instant Virtual Network on your Laptop (or other PC)," Mininet Team, [Online]. Available: http://mininet.org. [Accessed 23 02 2016].

[11] Wireshark Foundation, "Wireshark Home Page," [Online]. Available: https://www.wireshark.org. [Accessed 23 02 2016].

[12] Z. Majo and T. R. Gross, "Memory system performance in a NUMA multicore multiprocessor," in *Proceedings of the 4th Annual International Conference on Systems and Storage*, Haifa, Israel, 2011.

[13] C. Black, "What is 'real SDN'?," October 2014. [Online]. Available: http://searchsdn.techtarget.com/answer/What-is-real-SDN. [Accessed 23 02 2016].

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, "OpenFlow: Enabling Innovation," *ACM SIGCOMM Computer Communication Review,* vol. 38, no. 2, pp. 69-74, 2008.

[15] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown and S. Shenker, "NOX: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review,* vol. 38, no. 3, pp. 105-110, 2008.

[16] The Linux Foundation, "Open vSwitch Web Page," [Online]. Available: http://openvswitch.org. [Accessed 23 02 2016].

[17] Open Networking Foundation, "ONF Overview," [Online]. Available: https://www.opennetworking.org/about/onf-overview. [Accessed 23 02 2016].

[18] Hoelzle, Urs @ Open Networking Summit 2012, "OpenFlow @ Google," ONF, [Online]. Available: https://www.youtube.com/watch?v=VLHJUfgxEO4. [Accessed 23 02 2016].

[19] R. Merritt, "Google describes its OpenFlow network," 17 04 2012. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1261562. [Accessed 23 02 2016].

[20] The Linux Foundation, "The OpenDaylight Platform," [Online]. Available: https://www.opendaylight.org. [Accessed 23 02 2016].

[21] Open Networking Lab (ON.Lab), "ONOS: Open Networking Operating System," [Online]. Available: http://onosproject.org. [Accessed 23 02 2016].

[22] "The Linux Foundation," [Online]. Available: https://www.linuxfoundation.org. [Accessed 23 02 2016].

[23] OpenHUB, "The OpenDaylight Open Source Project on Open Hub," [Online]. Available: https://www.openhub.net/p/opendaylight. [Accessed 23 02 2016].

[24] "Open Networking Lab (ON.Lab)," [Online]. Available: http://onlab.us. [Accessed 23 02 2016].

[25] D. Pitt, "What's Ahead For SDN In 2016," 14 12 2015. [Online]. Available: http://www.networkcomputing.com/networking/whats-ahead-sdn-2016/778103327. [Accessed 23 02 2016].

[26] The Apache Software Foundation, "Apache Karaf," [Online]. Available: http://karaf.apache.org. [Accessed 23 02 2016].

[27] ONF, "OpenFlow Switch Specification - Version 1.5.1," 26 03 2015. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf. [Accessed 23 02 2016].

[28] Cisco, "Cisco Networking Academy Connecting Networks Companion Guide: Hierarchical Network Design," Cisco Networking Academy, [Online]. Available: http://www.ciscopress.com/articles/article.asp?p=2202410&seqNum=4. [Accessed 23 02 2016].

[29] N. Brownlee and K. C. Claffy, "Understanding Internet traffic streams: dragonflies and tortoises," *IEEE Communications magazine,* vol. 40, no. 10, pp. 110-117, 2002.

[30] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Transactions on Computer Systems (TOCS),* vol. 21, no. 3, pp. 270-313, 2003.

[31] K. Papagiannaki, N. Taft, S. Bhattacharyya, P. Thiran, K. Salamatian and C. Diot, "A pragmatic definition of elephants in internet backbone traffic," *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment,* pp. 175-176, 6-8 11 2002.

[32] K. Lan and J. Heidemann, "On the correlation of internet flow characteristics," Technical Report ISI-TR-574, USC/ISI, 2003.

[33] Cisco, "Monitor Microbursts on Cisco Nexus 5600 Platform and Cisco Nexus 6000 Series Switches," 15 09 2014. [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/switches/nexus-5000-series-switches/white-paper-c11-733020.html. [Accessed 23 02 2016].

[34] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson and S. Seshan, "Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems," in *6th USENIX Conference on File and Storage Technologies (FAST '08)*, San Jose, CA, US, 2008.

[35] Y. Chen, R. Griffith, J. Liu, R. H. Katz and A. D. Joseph, "Understanding TCP incast throughput collapse in datacenter networks," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*, Barcelona, Spain, 2009.

[36] E. Krevat, V. Vasudevan, A. Phanishayee, D. G. Andersen, G. R. Ganger, G. A. Gibson and S. Seshan, "On application-level approaches to avoiding TCP throughput collapse in cluster-based storage systems," in *Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing'07*, New York, NY, US, 2007.

[37] W. John and S. Tafvelin, "Analysis of internet backbone traffic and header anomalies observed," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, New York, NY, US, 2007.

[38] D. a. K. T. Murray, "The state of enterprise network traffic in 2012," in *18th Asia-Pacific Conference on Communications (APCC)*, Jeju Island, Korea, 2012.

[39] B. Lantz, B. Heller and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Monterey, CA, 2010.

[40] Riverbed, "SteelCentral Riverbed Modeler," [Online]. Available: http://www.riverbed.com/gb/products/steelcentral/steelcentral-riverbed-modeler.html. [Accessed 23 02 2016].

[41] SDx Central, "EstiNet 9.0 OpenFlow Network Simulator and Emulator, EstiNet Technologies Inc.," [Online]. Available: https://www.sdxcentral.com/products/estinet-8-1-openflow-network-simulator-and-emulator/. [Accessed 23 02 2016].

[42] NS-3 Consortium, "NS-3 Homepage," [Online]. Available: https://www.nsnam.org. [Accessed 23 02 2016].

[43] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on networking,* vol. 1, no. 4, pp. 397-413, 1993.

[44] R. Edwards, Y. Ye, A. Kaul and J. Yu, "Characterization of SDN Switch Response Time in Proactive Mode," in *Proceedings of the Open Networking Summit (ONS)*, Santa Clara, CA, US, 2014.

[45] Y. Zhao, L. Iannone and M. Riguidel, "On the performance of SDN controllers: A reality check," in *Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on*, 2015.

[46] Z. K. Khattak, M. Awais and A. Iqbal, "Performance evaluation of OpenDaylight SDN controller," in *20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, Hsinchu, Taiwan, 2014.

[47] "OpenDaylight Performance Stress Tests Report v1.0: Lithium vs Helium Comparison," Intracom Telecom, 2015.

[48] S. A. Pistirica, O. Poncea and M. C. Caraman, "QCN Based Dynamically Load Balancing: QCN Weighted Flow Queue Ranking," in *20th International Conference on Control Systems and Computer Science*, Bucharest, Romania, 2015.

[49] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE,* vol. 103, no. 1, pp. 14-76, 2015.

[50] S. A. Jyothi, M. Dong and P. Godfrey, "Towards a flexible data center fabric with source routing," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, Santa Clara, CA, US, 2015.

[51] Ryu SDN Framework Community, "Ryu SDN Framework," [Online]. Available: https://osrg.github.io/ryu/.

[52] "LinuxContainers," [Online]. Available: https://linuxcontainers.org.

[53] L. Deri, "nCap: Wire-speed packet capture and transmission," in *Workshop on End-to-End Monitoring Techniques and Services*, Nice, France.

[54] ntop, "PF_RING: High-speed packet capture, filtering and analysis," [Online]. Available: http://www.ntop.org/products/packet-capture/pf_ring/.

[55] Arista Networks, "Deploying IP Storage," 2014. [Online]. Available: http://goo.gl/Z9w0kp. [Accessed 10 02 2016].