



UNIVERSITATEA “POLITEHNICA” DIN BUCUREȘTI

ȘCOALA DOCTORALĂ ETTI-B

Nr. Decizie ..... din .....

## TEZĂ DE DOCTORAT

Contribuții la dezvoltarea și implementarea rețelelor  
definite prin programe soft

Contributions to the development and  
implementation of Software-Defined Networks

Doctorand: **Ing. Liviu-Alexandru STANCU**

Conducător de doctorat: **Prof. Dr. Ing. Simona HALUNGA**

### COMISIA DE DOCTORAT

Președinte	<b>Prof. Dr. Ing. Gheorghe BREZEANU</b>	de la	<b>Univ. “Politehnica” din București</b>
Conducător de doctorat	<b>Prof. Dr. Ing. Simona HALUNGA</b>	de la	<b>Univ. “Politehnica” din București</b>
Referent		de la	<b>Univ. “Politehnica” din București</b>
Referent		de la	<b>Univ. “Politehnica” din București</b>
Referent		de la	<b>Univ. “Politehnica” din București</b>

București 2017



# Muṭumiri



# Cuprins

Mulțumiri . . . . .	iii
Lista tabelelor . . . . .	viii
Lista figurilor . . . . .	x
Lista abrevierilor . . . . .	xiii
<b>1 Introducere</b>	<b>1</b>
1.1 Prezentarea domeniului tezei de doctorat . . . . .	1
1.2 Scopul tezei de doctorat . . . . .	1
1.3 Conținutul tezei de doctorat . . . . .	2
<b>2 Introducere în rețelele definite prin software</b>	<b>5</b>
2.1 Istoria și evoluția SDN . . . . .	5
2.1.1 Istoria SDN . . . . .	5
2.1.2 Evoluția SDN . . . . .	9
2.2 Standardizarea SDN . . . . .	18
2.2.1 Open Networking Foundation . . . . .	19
2.2.2 IETF . . . . .	22
2.3 SDN în contextul rețelor actuale . . . . .	22
2.3.1 SDN în centrele de date . . . . .	23
2.3.2 SDN în rețelele hibride . . . . .	25
2.3.3 Studii de caz. Implementări. Experimente în rețele de producție	25
<b>3 Unelte SDN în contextul rețelor de transport de date fără fir</b>	<b>27</b>
3.1 ONF TR-512.1 - Modelul informațional de bază ( <i>Core Model</i> ) . . . . .	28
3.1.1 Obiectul Element de Rețea ( <i>NetworkElement - NE</i> ) . . . . .	28
3.1.2 Obiectele Punct Logic de Terminație ( <i>LogicalTerminationPoint - LTP</i> ) și Nivelul Protocolului ( <i>LayerProtocol - LP</i> ) . . . . .	29
3.1.3 Obiectul Construct de Dirijare ( <i>ForwardingConstruct - FC</i> ) . . . . .	30
3.1.4 Obiectul Port al Constructului de Dirijare ( <i>FC Port</i> ) . . . . .	31
3.1.5 Obiectul Domeniu de Dirijare ( <i>ForwardingDomain - FD</i> ) . . . . .	31
3.2 ONF TR-532 - Modelul informațional pentru microunde ( <i>Microwave Information Model</i> ) . . . . .	31
3.2.1 Obiectul <i>MW_AirInterface_Pac</i> . . . . .	33
3.2.2 Obiectul <i>MW_PureEthernetStructure_Pac</i> . . . . .	34
3.2.3 Obiectul <i>MW_EthernetContainer_Pac</i> . . . . .	35
3.3 Protocolul NETCONF și limbajul YANG . . . . .	36

3.3.1	NETCONF . . . . .	36
3.3.2	YANG . . . . .	38
3.4	Alegerea unei soluții software pentru serverul NETCONF . . . . .	39
3.4.1	Netopeer . . . . .	39
3.4.2	OpenYuma . . . . .	40
3.4.3	Unealta de Test Netconf - <i>Netconf Testtool</i> . . . . .	40
3.4.4	Comparație între soluțiile care oferă server NETCONF . . . . .	41
3.5	Arhitectura demonstrațiilor de concept WT SDN . . . . .	43
<b>4</b>	<b>Mediatorul cu valori implicite (DVM)</b>	<b>47</b>
4.1	Arhitectura DVM versiunea 1 . . . . .	48
4.2	Implementarea DVM versiunea 1 . . . . .	49
4.3	Folosirea în contextul demonstrațiilor de concept a DVM versiunea 1	51
4.4	Arhitectura DVM versiunea 2 . . . . .	52
4.5	Implementarea DVM versiunea 2 . . . . .	55
4.6	Folosirea în contextul demonstrațiilor de concept a DVM versiunea 2	58
4.7	Integrarea DVM cu comutatorul software LINC: <i>LINC-WE</i> . . . . .	60
4.7.1	LINC . . . . .	60
4.7.2	LINC-WE . . . . .	62
4.7.3	Concluziile integrării . . . . .	65
<b>5</b>	<b>Simulatorul rețelelor de transport de date fără fir (WTE)</b>	<b>67</b>
5.1	Arhitectura . . . . .	67
5.2	Implementarea . . . . .	70
5.2.1	Fișierele de configurare . . . . .	70
5.2.2	Adaptarea DVM pentru WTE . . . . .	73
5.2.3	Nucleul simulatorului WTE . . . . .	75
5.2.4	Reprezentarea interfețelor unui dispozitiv de rețea . . . . .	77
5.2.5	Legăturile dintre interfețele elementelor de rețea . . . . .	79
5.2.6	Modelarea conexiunilor dintre dispozitivele simulate ca legături fără fir . . . . .	80
5.2.7	Linia de comandă . . . . .	82
5.2.8	Generarea traficului de date . . . . .	83
5.3	Folosirea în contextul demonstrațiilor de concept . . . . .	84
5.3.1	Altă implementare de server NETCONF . . . . .	84
<b>6</b>	<b>Rezultate și discuții</b>	<b>85</b>
6.1	Evaluarea soluțiilor propuse . . . . .	85
6.2	Comparație între WTE și alte abordări . . . . .	85
6.3	Demonstrarea cazurilor de utilizare cu ajutorul WTE . . . . .	85
<b>7</b>	<b>Concluzii</b>	<b>87</b>
7.1	Rezultate obținute . . . . .	87
7.2	Contribuții originale . . . . .	87
7.3	Lista contribuțiilor originale . . . . .	87

7.4	Perspective de dezvoltare ulterioară . . . . .	87
	<b>Bibliografie</b>	<b>89</b>





# Lista tabelelor

3.1	Comparație a caracteristicilor oferite de cadrele software considerate.	41
3.2	Compararea practică a cadrelor software considerate . . . . .	42
4.1	Comparație între comportamentele unui mediator real și al simulatorului în diferite situații [90]. . . . .	59



# Lista figurilor

2.1	Arhitectura SDN și abstractizările fundamentale [2] . . . . .	12
2.2	Nivelurile rețelelor definite prin software . . . . .	13
2.3	Implementarea SDN în contextul rețelelor actuale [51]. . . . .	23
3.1	Reprezentare UML simplificată a <i>CoreModel</i> [71]. . . . .	29
3.2	Reprezentare UML simplificată a <i>MicrowaveModel</i> și legătura acestuia cu <i>CoreModel</i> [72]. . . . .	32
3.3	Reprezentare UML simplificată a obiectului <i>MW_AirInterface_Pac</i> [72].	33
3.4	Reprezentare UML simplificată a obiectului <i>MW_PureEthernetStructure_Pac</i> [72]. . . . .	34
3.5	Reprezentare UML simplificată a obiectului <i>MW_EthernetContainer_Pac</i> [72]. . . . .	35
3.6	Arhitectura demonstrațiilor de concept WT [46–48]. . . . .	44
3.7	Poziționarea simulatoarelor în arhitectura ONF [46–48]. . . . .	45
3.8	Topologie simulată a unei rețele de transport de date bazată pe WTE.	45
4.1	Arhitectura primei versiuni a DVM [88]. . . . .	48
4.2	Organigramă a dezvoltării și implementării DVM [88]. . . . .	49
4.3	Structura fișierului de configurare al DVM. Exemplu pentru un echipament cu două interfețe radio. . . . .	51
4.4	Configurarea rețelei de test SDN utilizând mașini virtuale [47]. . . . .	52
4.5	Arhitectura simulatorului DVM versiunea 2. . . . .	54
4.6	Fluxul de lucru pentru aducerea atributelor: a) de execuție; b) de inițializare [90]. . . . .	55
4.7	Arhitectura comutatorului software LINC [97]. . . . .	61
4.8	Arhitectura comutatorului software LINC-WE [97]. . . . .	63
4.9	Structura aplicației Erlang <i>erl_yuma</i> [97]. . . . .	64
5.1	Arhitectura WTE. . . . .	68
5.2	Componentele majore ale WTE. . . . .	69
5.3	Formatul fișierului JSON care descrie topologia rețelei simulată cu WTE.	71
5.4	Formatul fișierului JSON de configurare a simulatorului WTE. . . . .	72
5.5	Organigrama mecanismului de generare a notificărilor NETCONF fictive al WTE. . . . .	74
5.6	Diagrama de clase a nucleului WTE. . . . .	75
5.7	Diagrama de secvență simplificată a inițializării WTE. . . . .	77

5.8	Obiectele LTP în contextul legăturii dintre două dispozitive de transport de date fără fir. . . . .	78
5.9	Obiectele LTP în interiorul unui dispozitiv de transport de date fără fir.	78
5.10	Legăturile dintre interfețele elementelor de rețea considerate pentru WTE: a) prin OVS; b) prin perechi Ethernet virtuale . . . . .	79
5.11	Captură de ecran a interfeței prin linie de comandă a WTE. . . . .	82

# Lista abrevierilor

API	Application Programming Interface
BBF	BroadBand Forum
BEEP	Blocks Extensible Exchange Protocol
CLI	Command Line Interface
CORBA	Common Object Request Broker Architecture
DVM	Default Values Mediator
ETC	Ethernet Container
ETH	Ethernet MAC Layer
ETSI	European Telecommunications Standards Institute
ETY	Ethernet Physical Layer
FC	Forwarding Construct
FD	Forwarding Domain
ForCES	Forwarding and Control Element Separation
HAL	Hardware Abstraction Layer
I2RS	Interface to the Routing System
IEEE	Institute of Electrical and Electronic Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IRTF	Internet Research Task Force
ITU-T	International Telecommunications Union - Telecommunications Standardization Sector
JSON	JavaScript Object Notation
LINC	Link Is Not Closed
LINC-OE	Link Is Not Closed - Optical Extensions
LINC-WE	Link Is Not Closed - Wireless Emulator
LLDP	Link Layer Discovery Protocol
LP	Layer Protocol
LTP	Logical Termination Point
MAC	Medium Access Control
MEF	Metro Ethernet Forum
MIMO	Multiple-Input Multiple-Output

MPLS	Multi-Protocol Label Switching
MPLS-TP	Multi-Protocol Label Switching Transport Profile
MWPS	Microwave Physical Section
MWPS-TTP	Microwave Physical Section - Trail Termination Point
MWS	Microwave Section
MWS-TTP	Microwave Section - Trail Termination Point
NAT	Network Address Translation
NCP	Network Control Point
NE	Network Element
NETCONF	Network Configuration Protocol
NVP	Network Virtualization Platform
OCh	Optical Channel
ODL	OpenDaylight
ODU	Optical Data Unit
OIF	Optical Interface Forum
OMS	Optical Multiplex Section
ONF	Open Networking Foundation
ONOS	Open Network Operating System
OPS	Optical Protection Switch
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
OT	Optical Transport
OTP	Open Telecom Platform
OTS	Optical Transmission Section
OTU	Optical channel Transport Unit
OTWG	Open Transport Working Group
OVS	Open Virtual Switch
OVSDB	Open vSwitch Database Management
PoC	Proof of Concept
POF	Protocol-Oblivious Forwarding
POSIX	Portable Operating System Interface
PTP	Precise Time Protocol
QAM	Quadrature Amplitude Modulation
QoS	Quality of Service
REST	Representational State Transfer
ROFL	Revised Open Flow Library
RPC	Remote Procedure Call
SDN	Software-Defined Networking

SDNRG	Software-Defined Networking Research Group
SDO	Standards Developing Organization
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SPRING	Source Packet Routing in Networking
SSH	Secure Shell
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TR	Technical Recommendation
TS	Technical Specification
UDP	User Datagram Protocol
UML	Unified Modeling Language
VLAN	Virtual Local Area Network
WAN	Wide Area Network
WLAN	Wireless Local Area Network
WT	Wireless Transport
WTE	Wireless Transport Emulator
xDPd	eXtensible Datapath Daemon
XML	Extensible Markup Language
XPIC	Cross Polarization Interference Cancellation
YANG	Yet Another Next Generation

# Capitolul 1

## Introducere

### 1.1 Prezentarea domeniului tezei de doctorat

În urma dezvoltării tehnologice recente în toate domeniile, în general și în domeniul calculatoarelor și al telecomunicațiilor, în particular, a apărut nevoia de a redefini arhitectura rețelelor de comunicații, din cauza faptului că rețelele tradiționale au început să își arate limitele. În zilele noastre, există o tendință de a interconecta toate echipamentele, cu ajutorul unor tehnologii care permit acest lucru, cum ar fi arhitectura *Cloud Computing*, de a crește mobilitatea tuturor aplicațiilor, de a defini concepte și aplicații noi, cum ar fi *Internetul Tuturor Lucrurilor* - IoT (Internet of Things) sau sistemele de comunicații de generația a cincea - 5G. Aceste noi abordări au nevoie, pe lângă o lărgime de bandă crescută, de o rețea mai simplă și agilă, care să faciliteze inovarea. Rețelele definite prin software - SDN (Software-Defined Networking), reprezintă o nouă paradigmă care a apărut în industria rețelisticii, pentru a contracara dezavantajele pe care rețelele tradiționale le-au dovedit.

Tehnologia SDN nu a ajuns încă la maturitate și nu a pătruns în toate tipurile de rețele de comunicații existente. Este prezentă în campusuri universitare, sau în centre de date, însă se încearcă introducerea acesteia în alte rețele de comunicații, cum ar fi transportul de date optic - OT (Optical Transport), transportul de date fără fir - WT (Wireless Transport) sau noduri de interconectare ale Internetului. Aceste încercări presupun muncă de cercetare, standardizare și demonstrații de concept - PoC (Proof of Concept), pentru prezentarea avantajelor pe care această nouă soluție le oferă, până când tehnologia se va maturiza și va fi adoptată de toată industria rețelisticii.

### 1.2 Scopul tezei de doctorat

Această lucrare își propune să prezinte noua paradigmă introdusă anterior, SDN, împreună cu avantajele pe care această abordare le poate aduce dacă ar fi aplicată în toate aspectele unei rețele de comunicații, punând accent pe rețelele de transport de date fără fir. Autorul își va prezenta activitatea de cercetare, constând în unelte software care pot fi folosite ca simulatoare de echipamente de transport de date fără fir, ce expun interfețe folosite în tehnologia rețelelor definite prin software.



Aceste unelte software au fost folosite cu succes în procesul de standardizare al SDN, care încă se desfășoară în cadrul ONF (Open Networking Foundation), facilitând testarea modelelor informaționale ce se dezvoltă în contextul rețelelor definite prin software și ușurând dezvoltarea și testarea aplicațiilor SDN care fac parte din acest ecosistem. Simulatorul rezultat în urma acestei cercetări, în forma sa finală, poate emula o întreagă rețea de echipamente de transport de date fără fir, care expun interfețe specifice SDN. El poate fi folosit de către dezvoltatorii de produse software SDN pentru rețele de transport de date fără fir, eliminând nevoia acestora de a deține astfel de echipamente scumpe pentru a-și putea testa aplicațiile. Poate fi folosit și de către operatorii care vor să lanseze această tehnologie în rețelele de producție, pentru a simula consecințele instalării unor noi aplicații anterior lansării, fără a afecta rețeaua.

### 1.3 Conținutul tezei de doctorat

Lucrarea este împărțită în opt capitole, primul prezentând domeniul abordat în teză, iar ultimul fiind dedicat concluziilor. În continuare va fi prezentat, pe scurt, conținutul fiecărui dintre celelalte capitole.

Capitolul 2 introduce domeniul rețelelor definite prin software, plecând de la istoria sa și nevoia pentru care această nouă paradigmă a apărut. Apoi va fi ilustrată activitatea de standardizare în acest domeniu, inclusiv demonstrațiile de concept conduse de către ONF, în particular de către grupul WT, care va duce la maturizarea soluției și adoptarea acesteia pe scară largă, în toate aspectele unei rețele. Tot în acest capitol se va evidenția și prezența SDN în contextul rețelelor actuale.

Cel de-al 3-lea capitol pune accent pe prezența SDN în rețelele de transport de date fără fir. Sunt prezentate modelele informaționale dezvoltate în cadrul ONF în acest context, având rolul de recomandări tehnice - TR (Technical Recommendation): modelul informațional pentru microunde - *TR-532, Microwave Information Model* și modelul informațional de bază - *TR-512, Core Information Model*. Ulterior se vor da detalii despre NETCONF (Network Configuration Protocol), care este protocolul de bază pentru rețelele de transport de date fără fir, în contextul SDN. În următoarea secțiune se vor compara câteva soluții software cu sursă deschisă, ce oferă facilitatea unui server NETCONF. Pe baza acestei comparații s-a ales una dintre software care va face parte din simulatorul propus în lucrare. Capitolul va fi încheiat de o prezentare a arhitecturii demonstrațiilor de concept organizate de grupul WT din ONF, ce va ajuta la înțelegerea necesității unui astfel de simulator.

Capitolul 4 prezintă prima versiune a simulatorului, numită *Mediatorul cu valori implicite* - DVM (Default Values Mediator), folosită în cea de-a doua demonstrație de concept WT din ONF. Se vor prezenta, pe rând, arhitectura și implementarea, iar apoi se va evidenția folosirea acestui simulator în contextul demonstrațiilor de concept.

Următorul capitol, ??, descrie cea de-a doua versiune a DVM, abordând aspecte despre arhitectura, implementare și folosire în cadrul celei de-a treia demonstrații de concept WT din ONF. În plus, se va evidenția încercarea de a integra acest simulator cu o soluție de comutator software, LINC, folosit în SDN, în special în cadrul rețelelor

de transport optic de date, prezentând avantajele și dezavantajele date de această abordare.

Capitolul 5 descrie ultima și cea mai avansată versiune a *simulatorului rețelelor de transport de date fără fir* - WTE (Wireless Transport Emulator), prezentând arhitectura, detaliile implementării și folosirea acestuia în cea de-a patra demonstrație de concept a grupului WT din cadrul ONF.

Capitolul 6 ilustrează rezultatele obținute în urma acestei cercetări și propune discuții pe baza simulatorului implementat. În primul rând, această soluție este evaluată din punctul de vedere al resurselor consumate și al extensibilității pe care o oferă. Apoi, se compară simulatorul cu alte soluții care există în momentul de față în contextul SDN. Ulterior, se prezintă câteva cazuri de utilizare, propuse în cadrul grupului WT din ONF, care pot fi demonstrate cu ajutorul simulatorului, eliminând nevoia unor echipamente de transport de date fără fir.



# Capitolul 2

## Introducere în rețelele definite prin software

În zilele noastre, rețelele de comunicații au devenit complexe, greu de administrat și configurat. De asemenea, numărul dispozitivelor mobile a crescut considerabil, precum și conținutul pe care acestea îl accesează. Aceste lucruri au dus la evidențierea limitărilor pe care rețelele tradiționale le presupun. Chiar dacă nu toate ideile ce stau la baza ei sunt noi, datorită unui context favorabil, acestea, împreună cu alte noi idei, au dus la apariția paradigmei SDN în industria rețelisticii.

Această nouă tehnologie nu a ajuns încă la maturitate și la adoptarea pe scară largă, în toate aspectele rețelilor, însă eforturile considerabile care se fac în activitățile de standardizare și crearea de ecosisteme SDN vor duce la această adoptare. După cum este evidențiat și în [1], se tinde către crearea unor rețele care pot fi programate prin software, crescând astfel flexibilitatea și agilitatea lor.

### 2.1 Istoria și evoluția SDN

#### 2.1.1 Istoria SDN

Rețelele definite prin software își au originile în activitatea și ideile din cadrul proiectului OpenFlow, început la universitatea Stanford, în jurul anului 2009. Multe dintre conceptele și ideile folosite în SDN au evoluat însă în ultimii 25 de ani și acum își găsesc locul în această nouă paradigmă, care își propune să schimbe modul în care rețelele sunt proiectate și administrate.

SDN reprezintă o arhitectură nouă de rețea, în care starea de dirijare a planului de date este administrată de un plan de control distant, decuplat de cel de date. Rețelele definite prin software reprezintă o arhitectură de rețea ce se bazează pe următoarele 4 concepte, conform [2]:

1. Decuplarea planurilor de date și de control;
2. Deciziile de dirijare se bazează pe fluxuri de date, nu pe adresa destinație;

3. Logica de control se mută într-o entitate externă, un echipament de control SDN (care are un sistem de operare de rețea);
4. Rețeaua este programabilă prin aplicații software care rulează peste sistemul de operare de rețea și care interacționează cu echipamentele din planul de date.

Rețelele definite prin programe software au apărut în scopul de a oferi posibilitatea inovației în cadrul administrării rețelelor și pentru a ușura introducerea de noi servicii. Aceste nevoi nu sunt însă noi, ele mai fiind studiate și în trecut, însă abia acum, prin SDN, pot fi satisfăcute într-un mod viabil, fără a implica schimbări majore în infrastructura rețelelor deja existente.

Istoria SDN poate fi împărțită în trei etape, fiecare influențând această nouă paradigmă prin conceptele propuse, așa cum este evidențiat în [3]:

1. Rețelele active, care au introdus funcțiile programabile în rețea, sporind gradul de inovație (mijlocul anilor 1990 – începutul anilor 2000);
2. Separarea planurilor de date și de control, care a condus la dezvoltarea de interfețe deschise între planurile de date și de control (aproximativ 2001 – 2007);
3. Dezvoltarea protocolului OpenFlow și a sistemelor de operare de rețea, care reprezintă prima adoptare pe scară largă a unei interfețe deschise, făcând separarea planurilor de control și de date extensibilă și practică.

## Rețelele active

Rețelele active reprezintă rețele în care comutatoarele pot efectua anumite calcule sau operații asupra pachetelor de date. Rețelele tradiționale nu pot fi considerate programabile. Rețelele active au reprezentat o schimbare de concept radicală asupra controlului unei rețele, propunând o interfață de programare care să expună resurse în noduri individuale de rețea și să susțină construirea de funcționalități specifice, care să fie aplicate unui subset de pachete care tranzitează acel nod.

Motivul principal pentru care rețelele active au apărut a fost cel de accelerare a inovației. La momentul respectiv, introducerea unui nou concept, serviciu sau tehnologie, într-o rețea de mari dimensiuni, cum ar fi Internet-ul, putea dura până la zece ani, de la faza de prototip până la implementare. Se dorea ca nodurile active din rețea să permită rutelor/comutatoarelor să descarce și să implementeze servicii noi în infrastructura deja existentă. În același timp, aceste noduri active ar fi putut coexista fără probleme în aceeași rețea cu vechile dispozitive.

Au existat două tipuri de abordări în cadrul rețelelor active, în funcție de modelul ales pentru programarea rețelei:

- *Modelul încapsulat* – unde codul care trebuia executat în cadrul nodurilor active era transportat în bandă, în pachetele de date; fiecare pachet de date conținea cod care trebuia rulat;

- *Modelul comutatoarelor programabile* – codul care trebuia executat în cadrul nodurilor active era stabilit prin mecanisme din afara benzii. Execuția programelor era determinată de antetul pachetului.

Rețelele active nu au ajuns niciodată să fie implementate pe scară largă, din mai multe cauze: momentul de timp la care au apărut acestea nu a fost potrivit; la acel moment nu aveau o aplicabilitate clară, deoarece nu apăruseră încă centrele de date sau infrastructurile de tip cloud; implementarea rețelelor active avea nevoie și de suport hardware, care nu era tocmai ieftin, ceea ce a constituit încă un dezavantaj.

Chiar dacă rețelele active nu au ajuns să fie implementate pe scară largă, câteva idei au fost preluate în cadrul rețelelor definite prin programe software:

- **Funcții programabile în rețea, care să faciliteze inovația.** În motivația introducerii rețelelor definite prin software se acuză dificultatea inovației în rețelele de producție. Rețelele active foloseau programarea planului de date, în timp ce în SDN se programează atât planul de control cât și cel de date.
- **Virtualizarea rețelei și capacitatea de a demultiplexa programe soft pe baza antetului pachetelor.** Rețelele active au dezvoltat un cadru arhitectural care să permită funcționarea unei astfel de platforme, având drept componente de bază un sistem de operare comun (al nodurilor), un set de medii de execuție și un set de aplicații active, care oferă de fapt un serviciu capăt-la-capăt.
- **Atenția la aparatele de rețea și la felul în care funcțiile acestora sunt compuse.** În cercetarea din cadrul rețelelor active se vorbea despre nevoia unificării gamei largi de funcții oferite de aparatele de rețea într-un cadru programabil sigur.

## Separarea planurilor de date și de control

Rețelele, încă de la început, au avut planurile de date și de control integrate. Acest lucru a dus la câteva dezavantaje: îngreunarea sarcinilor de administrare a rețelei, de depanare a configurării rețelei sau de controlul/prezicerea comportamentului de dirijare.

Primele inițiative de separare a planurilor de control și de date datează din anii 1980. La acel moment, cei de la AT&T propuneau renunțarea la semnalizarea în bandă și introducerea unui Punct de Control al Rețelei - NCP (Network Control Point), realizând astfel separarea planurilor de control și de date. Această modificare a înlesnit accelerarea inovației în rețea, prin posibilitatea de introducere rapidă de noi servicii și a furnizat noi metode de a îmbunătăți eficiența, prin introducerea unei vederi de ansamblu asupra rețelei oferită de punctul de control al rețelei. Există și inițiative mai recente care și-au propus separarea planurilor de control și de date, cum ar fi ETHANE [4], NOX [5], OpenFlow [6]. Acestea au ca avantaj faptul că nu au nevoie de modificări substanțiale în echipamentele de dirijare, ceea ce înseamnă că pot fi adoptate mai ușor de către industria rețelelor.

Ideile preluate în rețelele definite prin programe software din cercetarea care propunea separarea planurilor de control și de date sunt următoarele:

- **Control logic centralizat care folosește o interfață deschisă către planul de date.** O interfață deschisă către planul de date, care să permită inovația în aplicațiile software din planul de control a fost propusă de activitățile de cercetare din cadrul ForCES [7]. Însă, această interfață nu a fost adoptată de marile companii furnizoare de echipamente de rețea, astfel că aceasta nu a fost implementată pe scară largă.
- **Administrarea stărilor distribuite.** Controlul logic centralizat al rețelei a atras alte provocări, cum ar fi administrarea stărilor distribuite. Un echipament de control logic centralizat trebuie reprodus pentru a face față defectării acestuia. Însă această reproducere poate duce la stări de inconsistență între copiile echipamentului de control. Aceste probleme apar și în cadrul SDN, în contextul echipamentelor de control distribuite.

### Protocolul OpenFlow și sistemele de operare de rețea

Înainte de apariția protocolului OpenFlow, ideile care stăteau la baza rețelelor definite prin programe software aveau parte de o contradicție între viziunea unor rețele complet programabile și pragmatismul care ar fi permis lansarea în rețele reale. Protocolul OpenFlow a găsit un echilibru între aceste două obiective, prin posibilitatea de implementare la nivelul comutatoarelor deja existente în rețele (suport hardware deja existent) și prin implementarea mai multor funcții decât predecesorii săi. Chiar dacă, bazându-se pe suportul hardware deja existent, și-a asumat anumite limitări, protocolul OpenFlow a fost astfel imediat pregătit pentru lansare în rețelele de producție.

Inițial, s-a dorit ca protocolul OpenFlow să fie implementat în rețelele din campusuri studențești, pentru a putea fi conduse experimente în arhitectura de rețea, într-un mediu care să permită cercetarea.

După ce protocolul OpenFlow a avut succes în aceste rețele din campusuri, a început să ia amploare în alte domenii, cum ar fi centrele de date. S-a dovedit a fi mai eficient din punct de vedere al costurilor angajarea de ingineri care să dezvolte aplicații software sofisticate, de control al rețelei, decât cumpărarea de echipamente care să suporte aceleași facilități în mod proprietar.

Ideile care apar în SDN, derivate din cercetarea pentru dezvoltarea protocolului OpenFlow sunt următoarele:

- **Generalizarea funcțiilor și echipamentelor de rețea.** Ruterele clasice folosesc, în principiu, adresa IP (Internet Protocol) destinație pentru a dirija traficul. În schimb, protocolul OpenFlow poate defini comportamentul de dirijare în funcție de orice set din treisprezece antete diferite de pachet. Astfel, acest protocol unifică mai multe tipuri de echipamente de rețea, care diferă doar prin câmpurile din antetul pachetelor pe care le folosesc la dirijare și tipul de acțiuni pe care le efectuează.
- **Viziunea unui sistem de operare de rețea.** Spre deosebire de rețelele active, care propuneau un sistem de operare la nivelul nodurilor de rețea, cercetarea din cadrul OpenFlow a condus la noțiunea de sistem de operare de

rețea. Acesta oferă o împărțire a rețelei în trei niveluri: un plan de date cu o interfață deschisă, un nivel de administrare a stărilor, care are ca responsabilitate menținerea unei imagini consistente asupra stării rețelei și o logică de control, care să efectueze diferite operații, în funcție de imaginea sa asupra rețelei.

- **Tehnici de administrare a stărilor distribuite.** Separarea planurilor de date și de control a dus la provocări privind administrarea stărilor. E nevoie de existența mai multor echipamente de control, pentru performanța, extensibilitatea și siguranța rețelei, însă acestea trebuie să conlucreze ca un singur sistem logic de control.

### 2.1.2 Evoluția SDN

#### Motivația apariției SDN

Explozia numărului de dispozitive mobile și a conținutului pe care acestea îl accesează, apariția serviciilor de tip cloud, precum și virtualizarea serverelor au condus la reexaminarea arhitecturilor de rețea de către industria rețelisticii [8]. Astfel, s-au identificat limitări ale rețelelor tradiționale și, împreună cu nevoile determinate de evoluția tehnologiei s-a ajuns la concluzia că o nouă abordare în rețelistică este necesară: rețelele definite prin software.

Îndeplinirea cerințelor pieței în momentul de față, cu ajutorul arhitecturilor de rețea tradiționale, este aproape imposibilă. Costurile operaționale pentru o astfel de rețea sunt foarte mari, din cauza faptului că echipamentele de rețea trebuie administrate individual în momentul implementării de noi politici sau din cauza faptului că acele care provin de la producători diferiți trebuie controlate diferit. Pe lângă cele operaționale și costurile de capital au crescut pentru o rețea, din cauza nevoii așa numitor aparate de rețea care trebuie introduse pentru asigurarea securității rețelei, sau pentru a putea efectua operații de inginerie de trafic asupra rețelei respective. Așa cum este ilustrat în [8], printre limitările din rețelele tradiționale care au condus la nevoia apariției acestei noi paradigme amintim:

- **Complexitatea** – aceasta duce la stagnarea rețelelor. În momentul de față, rețelele sunt privite ca seturi discrete de protocoale care conectează utilizatorii, într-un mod sigur, indiferent de distanță, viteza conexiunilor sau topologiile folosite. Aceste protocoale sunt definite punctual, pentru a rezolva o anumită problemă și fără a beneficia de abstractizări fundamentale, ceea ce duce la creșterea complexității lor. Pentru adăugarea sau mutarea unui echipament de rețea, administratorii acestora trebuie să reconfigureze mai multe entități, atât hardware cât și software, folosind aplicații de administrare și trebuie să ia în considerare mai mulți factori, printre care topologia rețelei, modelul echipamentului, versiunea de software etc. Astfel, această complexitate a rețelelor tradiționale duce la o evoluție lentă a acestora, pentru a scădea riscul întreruperii serviciilor oferite de rețea. Acest lucru duce, de asemenea, la incapacitatea unei adaptări dinamice la modificările de trafic, aplicații și cereri ale utilizatorilor.



- **Inconsistența politicilor de rețea** – într-o rețea de producție, pentru implementarea unor politici care să fie valabile în întreaga rețea, este nevoie de configurarea a mii de dispozitive. Tot din cauza complexității rețelelor tradiționale, aplicarea unor politici consistente pentru acces, securitate sau calitatea serviciilor este foarte dificilă.
- **Probleme de extensibilitate** – ideea de a garanta mai multă bandă decât poate oferi o conexiune (over-subscription), având la bază modele de trafic predictibile, nu mai este o soluție pentru rețelele de la momentul actual. În centrele mari de date, care se bazează pe virtualizare, modelele de trafic sunt foarte dinamice și, implicit, greu de anticipat. Furnizorii foarte mari de servicii, cum ar fi Google sau Facebook, au nevoie de *rețele hiperscalabile*, care să poată asigura o conectivitate cu performanțe ridicate și un cost scăzut între sutele de mii de servere fizice de care aceștia dispun. Acest lucru implică mii de echipamente de rețea și o astfel de extensibilitate este imposibil de oferit printr-o configurare manuală a dispozitivelor rețelei.
- **Dependența de producători** – marile companii doresc un răspuns rapid la schimbările nevoilor de afaceri sau la cererile clienților. Însă, acest lucru este împiedicat de ciclul produselor vânzătorilor de echipamente, care poate fi chiar de câțiva ani. De asemenea, operatorii de rețea sunt limitați în a își personaliza rețeaua după bunul plac, din cauza lipsei de standarde și de interfețe deschise ale echipamentelor de rețea.

Majoritatea rețelelor tradiționale sunt rețele ierarhice, bazate pe niveluri de comutatoare Ethernet dispuse într-o structură de tip arbore. Această arhitectură era potrivită pentru modelul de calcul de tip client-server. Însă, o astfel de arhitectură statică nu mai este potrivită pentru corporațiile din zilele noastre, unde este nevoie de putere de calcul și de stocare dinamice în centrele de date. Printre ideile promotoare ale paradigmei rețelelor definite prin software se numără:

- **Modele dinamice de trafic** – o dată cu apariția marilor centre de date, modelele de trafic s-au schimbat radical. Dacă înainte, majoritatea aplicațiilor erau de tip client-server și majoritatea comunicației era realizată între client și server, aplicațiile mai noi accesează mai multe servere și baze de date, ceea ce implică o rafală de trafic de tip *est-vest* între diferite mașini, până ca datele să ajungă la utilizator, într-un tradițional model de trafic *nord-sud*. În același timp și modelele de trafic ale utilizatorilor se schimbă, aceștia dorind acces la resurse și aplicații de pe orice tip de dispozitiv, de oriunde și la orice oră.
- **Nevoia unui acces flexibil la resursele IT** – angajații încearcă tot mai mult să folosească dispozitive mobile personale, cum ar fi telefoanele inteligente, tabletele sau laptopurile, pentru a accesa rețeaua întreprinderilor. Administratorii rețelelor sunt astfel nevoiți să permită accesul acestor dispozitive și, în același timp, să protejeze datele întreprinderilor, proprietatea intelectuală și să îndeplinească mandatele de conformitate.

- **Dezvoltarea serviciilor de tip cloud** – marile companii au apelat la servicii de tip cloud, atât publice cât și private, ducând la o creștere masivă a acestui tip de servicii. Companiile doresc acum să poată accesa aplicații, infrastructura și alte resurse IT la cerere și la alegere. Pentru a se putea implementa acest tip de cereri, este nevoie de o extensibilitate a puterii de calcul, a puterii de stocare și a resurselor de rețea și este preferabil ca aceasta să poată fi făcută dintr-un punct comun și utilizând instrumente comune.
- **Nevoia de lărgime de bandă mai mare** – volumele foarte mari de date din ziua de astăzi necesită procesare paralelă masivă pe mii de servere interconectate, care au nevoie de conexiuni directe. Creșterea acestor volume de date înseamnă și nevoia creșterii capacității rețelelor. Operatorii acestor centre de date au complexa sarcină de a crea o rețea în acel centru de date care să fie extensibilă până la o dimensiune inimaginabilă și având grijă să nu se piardă conectivitatea între oricare două elemente de rețea.

Rețelele definite prin programe soft se dovedesc a fi foarte potrivite și în contextul apariției IoT, satisfăcând exact nevoile pe care acesta le are: creșterea lărgimii de bandă, configurarea dinamică a rețelei, arhitectură de rețea simplificată care să faciliteze inovația [9].

## Introducere în SDN

Rețelele definite prin software reprezintă o nouă paradigmă în arhitecturile de rețea și au la bază patru idei principale: (i) decuplarea planurilor de date și de control, (ii) deciziile de dirijare sunt luate pe baza fluxurilor de date, în loc de adresa destinație, (iii) planul de control se mută într-o entitate logică externă, numită echipament de control SDN, care rulează un sistem de operare de rețea și (iv) rețeaua este programabilă prin aplicații software care rulează în sistemul de operare de rețea și interacționează cu echipamentele de dirijare.

Astfel, rețelele definite prin software pot fi definite cu ajutorul a trei abstractizări, conform [2], cum se poate observa în Figura 2.1:

- Abstractizarea dirijării;
- Abstractizarea distribuției;
- Abstractizarea specificărilor.

În mod ideal, *abstractizarea dirijării* reprezintă permiterea oricărui comportament de dirijare dorit de aplicațiile software din rețea (cu ajutorul planului de control), fără a necesita cunoașterea detaliilor legate de capabilitățile hardware ale infrastructurii existente. Un exemplu pentru o astfel de abstractizare este protocolul OpenFlow.

*Abstractizarea distribuției* se referă la faptul că aplicațiile SDN nu ar trebui să cunoască problemele stărilor distribuite din rețea, transformând problemele unui plan de control distribuit, cum era în rețelele tradiționale, într-un plan de control logic centralizat. Acesta este realizat printr-un nivel comun de distribuție, în SDN fiind

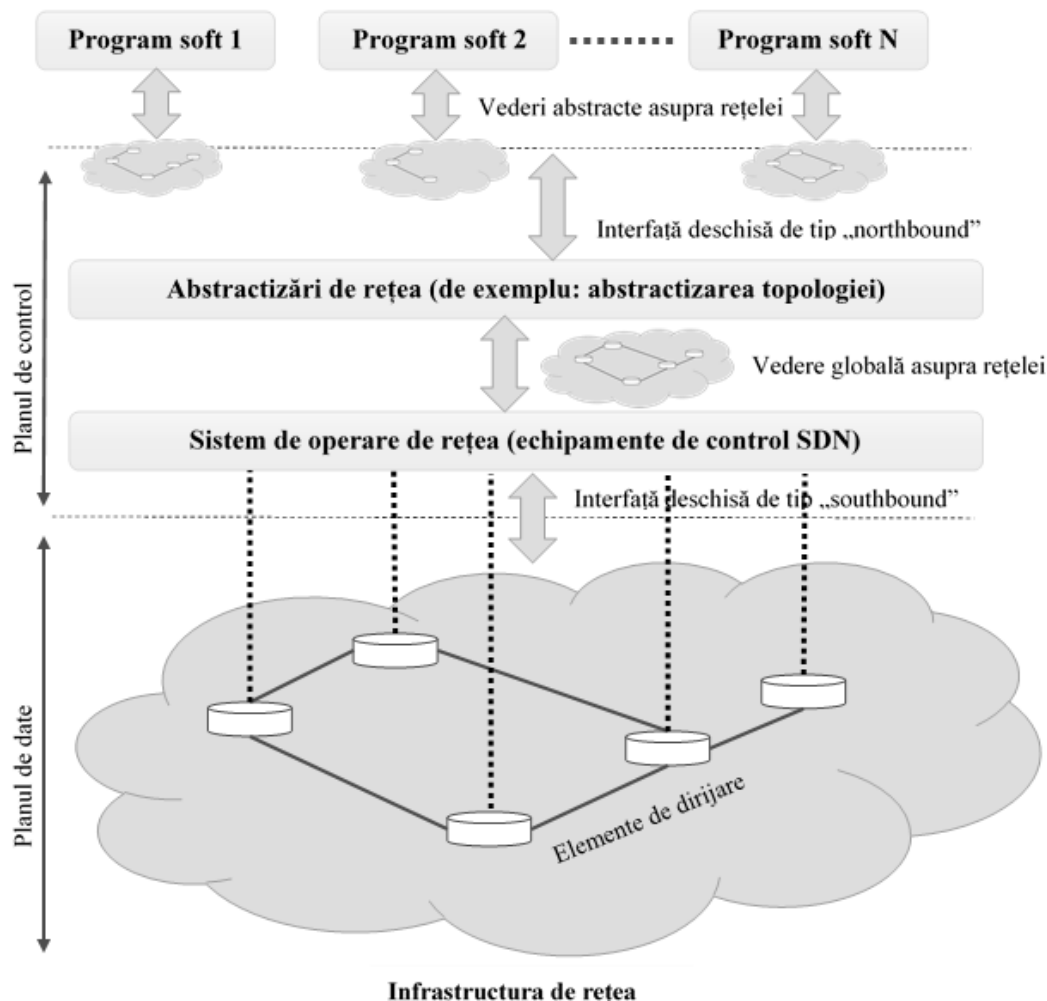


Fig. 2.1: Arhitectura SDN și abstractizările fundamentale [2]

reprezentat de sistemul de operare de rețea. Acesta are două mari funcții: instalarea comenzilor de control pe echipamentele de dirijare și colectarea de informații despre starea planului de date, pentru a putea oferi programelor software o vedere de ansamblu asupra rețelei.

*Abstractizarea specificărilor* reprezintă capacitatea unui program software din rețea de a exprima un anumit comportament al acestuia fără a fi responsabil personal și de implementarea acestui comportament. Acest lucru se poate realiza prin soluții de virtualizare, precum și cu ajutorul limbajelor de programare de rețea.

Arhitectura SDN poate fi privită ca o structură cu mai multe niveluri, având fiecare funcțiile sale specifice. Unele niveluri sunt necesare în orice implementare SDN, în timp ce altele pot fi prezente doar în anumite implementări particulare. Aceste niveluri sunt ilustrate în Figura 2.2 și vor fi prezentate în continuare [2].

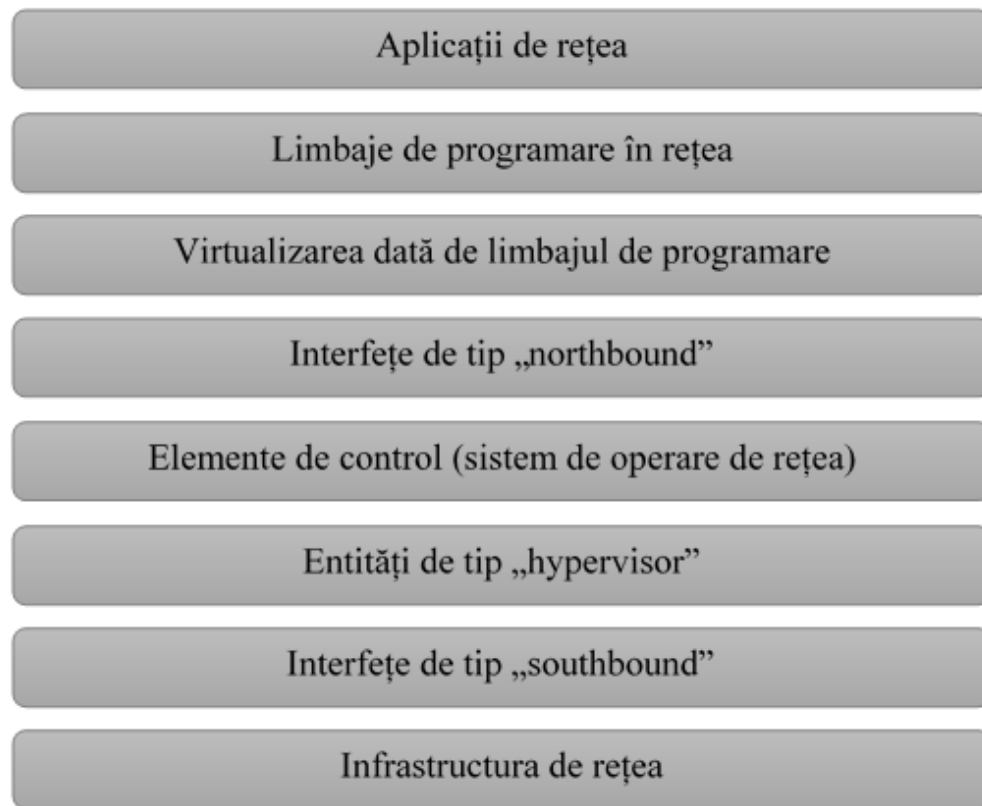


Fig. 2.2: Nivelurile rețelelor definite prin software

**Nivelul 1: Infrastructura** Infrastructura pentru o rețea definită prin programe software este compusă, la fel ca în cazul rețelelor tradiționale, din echipamente de rețea (comutatoare, rutere, echipamente de transport de date etc.). Însă, diferența majoră constă în faptul că, în SDN, echipamentele de rețea sunt simple elemente de dirijare de trafic, inteligența lor fiind mutată în echipamentele de control SDN și în aplicațiile software. O altă diferență importantă este aceea că echipamentele care constituie infrastructura de rețea trebuie să implementeze interfețe standard (cum ar fi OpenFlow), care să asigure compatibilitatea comunicației și a configurărilor, dar și interoperabilitatea cu alte echipamente, atât din planul de control cât și din cel de date, indiferent de producător. Acest lucru era destul de dificil în rețelele tradiționale, din cauza multitudinii de interfețe proprietare.

În paradigma SDN, elementele care alcătuiesc infrastructura rețelei nu mai iau deciziile de dirijare pe baza adresei destinație, ca în cazul rețelelor tradiționale, ci pe baza unor fluxuri de date [6]. Presupunând o rețea bazată pe protocolul OpenFlow, elementele de dirijare se bazează pe o secvență de tabele de fluxuri, unde fiecare intrare din tabel conține: (i) o regulă de asociere, (ii) acțiunile care trebuie executate asupra pachetelor care se potrivesc regulii și (iii) contoare care să mențină o statistică despre pachetele care s-au potrivit. Prioritatea regulilor este dată de ordinea din tabelul de

fluxuri. Acțiunile posibile asupra pachetelor pot fi: dirijarea acestora către un port de ieșire, dirijarea către echipamentul de control, aruncarea pachetelor, trimiterea către secvența normală de prelucrare, trimiterea către următorul tabel de fluxuri etc. Regulile de potrivire care se pot aplica pachetelor se pot baza pe mai multe câmpuri din antetul pachetului, cum ar fi câmpuri de nivel doi (Ethernet), MPLS, de nivel 3 (IPv4/v6), câmpuri de nivel 4 (TCP/IP) etc., în aproape orice fel de combinație.

**Nivelul 2: Interfețele de Sud (*southbound*)** Interfețele de Sud fac legătura între echipamentele de dirijare și echipamentele de control din rețea, îndeplinind astfel una dintre funcțiile de bază ale rețelelor definite prin programe soft: separarea planurilor de date și de control. Terminologia Nord/Sud se raportează la controlerul SDN.

Acest tip de interfețe este foarte bine primit de către industrie. Prin standardizarea acestora se va permite construirea de rețele cu echipamente care să poată proveni de la mai mulți producători, promovând astfel interoperabilitatea.

Cel mai acceptat și utilizat standard pentru acest tip de interfețe este OpenFlow. Acesta propune specificații comune pentru implementarea canalului de comunicație dintre echipamentele de dirijare și cele de control. Protocolul OpenFlow furnizează trei surse de informații pentru sistemul de operare de rețea: (i) mesaje bazate pe evenimente, care sunt trimise de echipamentele de dirijare către cele de control în momentul în care apare o schimbare a legăturii de date sau a unui port; (ii) statistici despre fluxurile de date, care sunt generate în echipamentele de dirijare și trimise către echipamentele de control; (iii) mesaje care conțin și pachete de date și sunt trimise de către echipamentele de dirijare către cele de control în momentul în care nu știu cum să trateze un anumit tip de flux de date sau din cauza faptului că există o acțiune explicită de tipul *trimite la echipamentul de control* în tabela de fluxuri. Aceste tipuri de informații sunt esențiale pentru furnizarea de detalii despre fluxurile de date sistemului de operare de rețea.

Deși este cel mai utilizat protocol, OpenFlow nu este singura interfață de Sud pentru rețelele definite prin programe soft. Există și alte propuneri pentru acest tip de interfețe, dintre care amintim: NETCONF, ForCES (Forwarding and Control Element Separation), OpFlex, POF (Protocol-Oblivious Forwarding), OVSDB (Open vSwitch Database Management), ROFL (Revised Open Flow Library), HAL (Hardware Abstraction Layer), OpenState etc. [7, 10, 11].

**Nivelul 3: Entitățile de tip *hypervisor* de rețea** Entitățile de tip *hypervisor* reprezintă o soluție software, firmware, sau hardware care creează și rulează mașini virtuale. Acestea permit unor mașini virtuale distincte să partajeze aceleași resurse hardware. Astfel au apărut noi modele de afaceri și tehnologii, cum ar fi aplicațiile de tip cloud, unde fiecare utilizator poate avea resursele sale virtuale, de la puterea de calcul până la spațiul de stocare. Din păcate, acest model nu a putut fi aplicat și pentru resursele de rețea, acestea fiind configurate în continuare în mod static și independent una față de cealaltă.

Există două cerințe ale aplicațiilor de la rețea: spațiul de adresare și topologia rețelei. Volume diferite de muncă necesită diferite servicii și topologii de rețea, însă acestea sunt greu de oferit de către o singură topologie fizică. De asemenea, aplicațiile care rulează într-un mediu virtualizat funcționează în același spațiu de adresare ca infrastructura fizică. Astfel, mașinile virtuale nu pot migra în locuri arbitrare, deoarece schema de adresare este fixă și greu de modificat.

Soluția la această problemă ar fi ca rețeaua să ofere la rândul ei virtualizare, din punctul de vedere al topologiei și al spațiului de adresare. Apoi, fiecare aplicație care rulează într-o mașină virtuală ar avea posibilitatea să configureze atât nodurile de calcul cât și rețeaua, în același timp. Acest lucru nu este posibil prin tehnologiile actuale, cum ar fi VLAN (Virtual Local Area Network) - domeniu virtualizat de nivel 2, NAT (Network Address Translation) - spațiu de adresare IP virtualizat și MPLS (Multi-Protocol Label Switching) - rute virtuale, deoarece rețeaua nu se poate reconfigura într-un mod global, fiind nevoie ca fiecare element de rețea să fie reconfigurat individual [2, 12, 13].

Rețelele definite prin software încearcă să ofere posibilitatea de virtualizare a rețelei, prin această entitate de tip *hypervisor*. Există mai multe propuneri în acest sens, dintre care amintim: FlowVisor, NVP (Network Virtualization Platform), FlowN, RadioVisor, IBM SDN VE, OpenVirteX, HyperFlex, xDPd (eXtensible Datapath Daemon) etc. [14–18].

**Nivelul 4: Echipamentele de control / Sistemul de operare de rețea** Sistemele de operare tradiționale oferă abstractizări pentru accesarea resurselor hardware, pentru administrarea accesului concurent la resurse și pentru a oferi mecanisme de securitate și protecție. Prin contrast, rețelele au fost administrate și configurate până acum cu ajutorul unor instrucțiuni de nivel jos, specifice fiecărui echipament și cu sisteme de operare de rețea proprietare (cum ar fi IOS de la Cisco sau JunOS de la Juniper). Acestea nu furnizează abstractizări care să ofere, într-un mod transparent, funcționalități de bază [2].

În cadrul SDN se încearcă găsirea unor soluții în acest sens, printr-un control logic centralizat, oferit de un sistem de operare de rețea. Acesta va trebui să ofere abstractizări, care să fie apoi folosite de dezvoltatorii de aplicații software. Printre serviciile oferite de sistemul de operare de rețea ar trebui să se regăsească funcționalități generale, cum ar fi: descoperirea dispozitivelor de rețea, distribuirea configurației rețelei, starea rețelei sau informații despre topologia rețelei.

Echipamentul de control este elementul critic din arhitectura SDN, deoarece el va trebui să interpreteze aplicațiile dezvoltate pentru administrarea rețelei și să genereze configurația acestora pe baza politicilor definite acolo. Există deja o multitudine de echipamente de control propuse pentru a fi utilizate în arhitectura SDN, diferite în foarte multe aspecte. Unul dintre cele mai importante, care diferențiază echipamentele de control, este tipul arhitectural folosit: centralizat sau distribuit [19–21].

Un echipament de control centralizat reprezintă o entitate care administrează toate dispozitivele din planul de date al rețelei [8]. În mod natural, acesta poate fi privit ca un punct unic de defectare a rețelei și ar putea aduce limitări în extensi-

bilitatea acestora, deoarece este posibil ca un singur echipament de control să nu fie suficient pentru administrarea unui număr mare de echipamente de dirijare. Printre propunerile de astfel de echipamente de control se numără: Maestro, Beacon (care poate administra un număr foarte mare de fluxuri de date, undeva la 12 milioane de fluxuri pe secundă, conform [22]), NOX-MT, sau Floodlight. Acestea se bazează pe mai multe fire de execuție și pe paralelismul oferit de arhitecturile de calculatoare cu mai multe nuclee.

Pe de altă parte, un sistem de operare de rețea distribuit poate rezolva problema extensibilității rețelei, fiind potrivit pentru orice tip de rețea. Această distribuție poate fi realizată printr-un grup de echipamente de control care să se afle în același loc, sau printr-un set de dispozitive distribuite fizic în mai multe locuri. Această ultimă variantă ar putea fi mai potrivită pentru prevenirea diferitelor defecțiuni logice sau fizice. Câteva exemple de astfel de echipamente de control distribuite: Onix [23], HP VAN SDN, PANE, HyperFlow [24], ONOS (Open Network Operating System) [25] sau ODL (OpenDaylight) [26], SMarTLight [27] etc.

Însă, în cazul echipamentelor de control distribuite, apare o problemă destul de importantă: consistența datelor. Toate echipamentele de control ar trebui să citească aceeași valoare imediat după ce aceasta a fost scrisă, pentru a evita cazul când dispozitivele de control au imagini diferite asupra rețelei. Majoritatea propunerilor de până acum oferă doar o *consistență scăzută*, ceea ce înseamnă că, după ce o valoare a fost scrisă într-un nod de control, valoarea se va reflecta, *la un moment dat*, în toate nodurile de control. Acest lucru implică o perioadă de timp în care dispozitivele de control au viziuni diferite asupra rețelei. Există și propuneri de echipamente de control care oferă o *consistență ridicată* (Onix și SMarTLight) [23, 27], care garantează citirea aceleiași valori de către orice nod de control, imediat după scrierea acesteia.

Există și situații unde o abordare hibridă ar fi cea mai potrivită, o arhitectură în care să existe grupuri de echipamente de control într-o parte de rețea și dispozitive de control distribuite în alte locuri din rețea.

**Nivelul 5: Interfețe de Nord (*northbound*)** Interfețele de Nord, împreună cu cele de Sud constituie cele mai importante două abstractizări din arhitectura rețelelor definite prin software. Dacă cele din urmă asigură comunicația dintre echipamentele de control și dispozitivele din planul de date al rețelei, fiind astfel mai mult orientate spre hardware, interfețele de Nord alcătuiesc, în mare parte, un ecosistem software. Încă nu există un astfel de tip de interfețe care să fie acceptat la scară largă, cum este OpenFlow în cazul celor de Sud, însă acest lucru se va întâmpla probabil, pe măsură ce paradigma SDN se va maturiza și cazurile de utilizare ale acestui tip de arhitectură de rețea se vor contura mai exact.

Este nevoie ca acest tip de interfețe să fie deschise și standardizate, astfel încât să se asigure interoperabilitatea și portabilitatea programelor soft pe mai multe tipuri de dispozitive de control. Un exemplu în acest sens îl constituie NOSIX [28]. Aceasta este o propunere, care poate fi comparată cu standardul POSIX (Portable Operating System Interface) din sistemele de operare și care oferă abstractizări ce garantează independența față de limbajul de programare și dispozitivul de control

folosite [29]. Dintre celelalte propuneri de interfețe de Nord amintim: RESTCONF, SFNet, Pyretic, NetCore, Frenetic, Nettle etc. [30–33].

**Nivelul 6: Virtualizarea dată de limbajul de programare** Există două caracteristici principale ale soluțiilor de virtualizare date de limbajele de programare: permiterea mai multor niveluri de abstractizare, concomitent cu oferirea proprietăților dorite, cum ar fi protecția și abilitatea de a exprima modularitatea.

Metodele de virtualizare pot permite, de exemplu, diferite imagini asupra unei singure infrastructuri fizice de rețea. Un *comutator virtual* ar putea reprezenta o combinație de mai multe dispozitive de dirijare. Acest mod de lucru simplifică sarcinile dezvoltatorilor de aplicații, care nu trebuie să țină cont individual de elementele de rețea care alcătuiesc acel *comutator virtual*. Dezvoltarea și implementarea unor aplicații de rețea complexe este simplificată cu ajutorul acestor abstractizări.

O altă formă de virtualizare dată de limbajul de programare o reprezintă împărțirea statică a rețelei în secțiuni. Acest lucru se face de către compilator, bazat pe definițiile date de nivelul aplicație. După compilare va rezulta un program unitar de control, care are deja implementate funcțiile pentru împărțirea rețelei în bucăți și comenzile de configurare a acestora. În acest caz nu mai este nevoie de entitatea *hypervisor* care să administreze dinamic bucățile de rețea.

Există diverse propuneri pentru astfel de soluții de virtualizare, cum ar fi: Pyretic, Splendid, libNetVirt, FlowVisor, IBM SDN VE etc. [14, 32, 34, 35].

**Nivelul 7: limbaje de programare în rețea** Limbajele de programare au evoluat de la limbaje mașină, specifice hardware-ului, cum era limbajul de asamblare pentru arhitecturile x86, până la limbaje de nivel înalt, cum ar fi Java sau Python. În același mod, limbajele de programare folosite în programarea rețelelor evoluează de la OpenFlow (echivalentul limbajului de asamblare) la limbaje de nivel înalt, cum ar fi Pyretic, Protera, NetCore, Frenetic etc. [32, 33, 36].

Aceste limbaje de programare de nivel înalt oferă câteva avantaje în contextul rețelelor definite prin software: facilitează dezvoltarea virtualizării rețelei, creează abstractizări care simplifică programarea elementelor de dirijare, promovează modularizarea software și reutilizarea codului în planul de control și chiar îmbunătățesc dezvoltarea și inovația prin crearea unor medii de lucru mai productive.

Există două tipuri de paradigme de programare în contextul SDN: cea declarativă, care este cea mai răspândită și cea imperativă, care este reprezentată doar prin limbajul Pyretic. Paradigma declarativă reprezintă o abordare în care rețelei i se spune ce tip de comportament să aibă și aceasta se va configura (luând singură decizii) astfel încât să îndeplinească acea cerință. Paradigma imperativă se referă la situația în care programatorul îi spune rețelei cum să acționeze și rezultatul va fi cel așteptat, fără ca aceasta să ia propriile decizii.

Scopul SDN este ca, în final, să ofere facilități de administrare a rețelelor pe baza infrastructurii definite anterior. Cu ajutorul progreselor din domeniul limbajelor de programare de nivel înalt se va facilita crearea unui ecosistem propice pentru dezvoltarea aplicațiilor SDN.



**Nivelul 8: Aplicațiile de rețea** Aplicațiile software vor reprezenta cea mai importantă parte a rețelelor definite prin software, deoarece acestea vor implementa logica de control, care va fi translatată în comenzi ce vor fi instalate la nivelul dispozitivelor din planul de date.

Majoritatea aplicațiilor software din cadrul SDN se încadrează într-una din următoarele cinci categorii, conform [2]: ingineria traficului, securitate și fiabilitate, măsurători și monitorizare, rețelistica centrelor de date, mobilitate și rețele fără fir.

Programele software din categoria mobilitate și microunde își propun să faciliteze implementarea și administrarea rețelelor fără fir, cum ar fi rețelele locale fără fir - WLAN (Wireless Local Area Network) sau rețelele de telefonie mobilă. Un plan de control distribuit, cum există în momentul de față în rețelele fără fir, nu este optim pentru implementarea mecanismelor de transfer dintre celule, pentru micșorarea interferențelor, pentru alocarea resurselor radio, pentru administrarea spectrului limitat de frecvențe etc. Aceste probleme sunt adresate în SDN și se pot rezolva mai ușor cu ajutorul acestei noi paradigme.

## 2.2 Standardizarea SDN

Activitățile de cercetare și standardizare din jurul SDN au loc pe două planuri importante. Pe de o parte, există organizațiile care dezvoltă standarde - SDO (Standards Developing Organization), care sunt formate din reprezentanți ai industriei, ai academiei, sau alte entități și au ca scop dezvoltarea de specificații sau recomandări tehnice, în contextul SDN care să fie folosite de toată industria rețelisticii. Autorii din [37] amintesc astfel de organizații: ONF, IETF (Internet Engineering Task Force), ETSI (European Telecommunications Standards Institute), ITU-T (International Telecommunications Union - Telecommunications Standardization Sector), IEEE (Institute of Electrical and Electronic Engineers).

Pe de altă parte, există asociații sau comunități de oameni, în general care fac parte din industrie, ale căror rezultate ale cercetării candidează pentru a deveni standarde. Aceste rezultate sunt, de obicei, implementări cu sursă deschisă ce vor fi folosite ulterior în industrie. Exemple de astfel de comunități sunt prezente în [38, 39]: OpenDaylight (activități ce se desfășoară sub patronajul fundației Linux), MEF (Metro Ethernet Forum), BBF (BroadBand Forum), OIF (Optical Interface Forum).

Activitățile de standardizare ale SDN sunt foarte importante, asigurându-i acestei tehnologii o evoluție stabilă și aducând-o la o maturitate care îi va permite adoptarea pe scară largă în industria rețelisticii, mitigând astfel dezavantajele rețelelor tradiționale.

Sunt mai multe planuri pe care se lucrează pentru standardizarea SDN. Unul dintre aceste planuri este cel al interfețelor de Sud, care fac legătura între echipamentele de dirijare și echipamentele de control SDN. Protocolul OpenFlow este un exemplu în acest sens, însă nu este singurul protocol capabil să facă legătura între planurile de date și de control. În ultimul timp se pune foarte mare accent pe NETCONF ca o alternativă pentru a configura echipamentele care dirijează traficul, după cum se poate observa în [10, 40, 41]. Astfel apare nevoia de a dezvolta modele informaționale

care să abstractizeze echipamentele din planul de date și care să poată fi folosite de NETCONF pentru a configura dispozitivele. Și planul interfețelor de Nord are un rol important în standardizare, deoarece poate oferi un punct de plecare comun pentru dezvoltatorii de aplicații SDN. De exemplu, în cadrul ONF există un grup care se ocupă cu activități de standardizare în această direcție. Un alt plan este reprezentat de implementările software, cu sursă deschisă (*open-source*), care se crează în acest context. Așa cum evidențiază și autorii din [42, 43], aceste implementări sunt foarte importante prin ecosistemele care apar ca urmare a activității comunităților care dezvoltă software cu sursă deschisă.

O importanță foarte mare în cadrul acestor activități o au și demonstrațiile de concept. Acestea au capacitatea de a demonstra avantajele pe care SDN le aduce nu doar la un nivel teoretic, ci într-un mod practic, propunând diferite cazuri reale de utilizare a acestei tehnologii și aplicând-o, într-un mod restrâns, unor topologii formate din echipamente reale. Scopul acestora este, pe de o parte, de a adăuga un plus de valoare activităților de standardizare și de a testa și proba rezultatele acestor activități. Pe de altă parte, aceste demonstrații de concept au scopul de a atrage atenția și altor entități din industrie și a duce la înlesnirea adoptării acestei tehnologii pe scară largă.

### 2.2.1 Open Networking Foundation

ONF este o organizație non-profit formată din peste două sute de membri care fac parte din industrie, academie, sau institute de cercetare, ce are ca obiectiv accelerarea adoptării SDN pe scară largă în industria rețelisticii prin dezvoltarea de standarde deschise și de ecosisteme software cu sursă deschisă. ONF a apărut ca urmare a activității de cercetare din jurul protocolului OpenFlow din cadrul Universității Stanford. Dintre membrii cei mai importanți amintim operatori de rețele, precum AT&T, Google, Facebook, Verizon, Deutsche Telekom sau Telefonica, producători de echipamente, cum ar fi Cisco, Ericsson, Huawei, Intel sau NEC și reprezentanți ai unor universități cunoscute, ca Stanford sau Princeton.

Activitățile din cadrul ONF sunt împărțite în mai multe zone de interes, conform [44]:

- *Operatori.* Această zonă se ocupă de mai multe aspecte, precum SDN în contextul sistemelor de tip Purtător - *Carrier Grade* - (rețelele de telecomunicații foarte sigure, cu o disponibilitate de peste 99,999% și recuperare în caz de defectare de sub 50 de milisecunde), în centre de date, în întreprinderi sau aspecte legate de migrarea serviciilor dintr-o rețea tradițională într-o rețea definită prin software.
- *Servicii.* Se ocupă de proiecte care permit existența aplicațiilor și serviciilor de rețea care au la bază tehnologia SDN. Astfel, există mai multe grupuri de lucru care analizează arhitectura SDN și modul în care aceste principii se aplică în imaginea de ansamblu a unei rețele de comunicații, un model informațional *de bază*, care să reprezinte piatra de temelie cu ajutorul căreia să se dezvolte alte modele informaționale, specializate pentru anumite aplicații sau tehnologii,

interfețele de Nord, pentru a facilita dezvoltarea aplicațiilor SDN, probleme de securitate pe care această nouă tehnologie le poate întâmpina.

- *Specificații*. Zonă care are ca responsabilitate publicarea tuturor specificațiilor și recomandărilor tehnice create de ONF. Acestea includ protocoalele OpenFlow, OF-Config dar și alte interfețe standard care se dezvoltă pentru tehnologii de transport de diferite tipuri (optic, fără fir). Există și un proiect care se ocupă de testare și interoperabilitate, scopul acestuia fiind accelerarea adoptării protocolului OpenFlow prin certificări și promovarea interoperabilității între diferiți producători de echipamente [45].
- *Piață*. Această zonă se concentrează pe educarea comunității SDN în legătură cu valoarea pe care standardele ONF le oferă rețelelor definite prin software și promovarea adoptării unor astfel de rețele definite prin software cu sursă deschisă. Aceste obiective sunt îndeplinite prin publicații, evenimente care se organizează sau demonstrații care să arate comunității cazuri reale de utilizare.

Există trei tipuri de publicații care reies din activitățile ONF: (i) *specificații*, care includ toate standardele care definesc un protocol, modelul informațional, funcționalități și documente despre cadrele asociate; publicațiile normative astfel rezultate se numesc Specificații Tehnice - TS (Technical Specification) și sunt supuse unor procese de licențiere; (ii) *recomandări*, care includ considerații arhitecturale, cazuri de utilizare, analiza cerințelor, terminologie; aceste documente referă documente normative ONF, dar nu necesită licențiere și pot fi utilizate în mod liber, având rolul de Recomandări Tehnice - TR; (iii) *publicații*, reprezentând documente care conțin informații ce ajută în procesele de lansare a SDN în producție, rezumate ale soluțiilor, studii de caz sau cărți albe (*white papers*); aceste documente nu au caracter normativ și pot fi folosite în mod liber.

În continuare vor fi enumerate câteva astfel de documente produse de către ONF: *OpenFlow Switch Specification Ver. 1.5.1* (TS-025), care descrie cerințele unui comutator logic ce suportă protocolul OpenFlow; *OpenFlow Management and Configuration Protocol 1.2 (OF-Config 1.2)* (TS-016), document ce descrie motivația, cerințele, scopul și specificațiile protocolului OF-Config; *Conformance Test Specification for OpenFlow Switch Specification V1.3.4* (TS-026), definind cerințele și procedurile de test care determină conformitatea unui comutator cu specificațiile protocolului OpenFlow 1.3.4; *Core Information Model (CoreModel) 1.2* (TR-512), care prezintă modelul informațional de bază, pe care celelalte modele informaționale dezvoltate în ONF se bazează; *Microwave Information Model* (TR-532), reprezentând modelul informațional ce descrie echipamentele de transport de date fără fir. Aceste ultime două documente vor fi detaliate în următorul capitol, fiind baza dezvoltării și implementării simulatoarelor propuse în această lucrare.

Un rol important în adoptarea SDN de către operatori și în diseminarea rezultatelor din cadrul ONF îl au demonstrațiile de concept. Acestea adună laolaltă operatori, producători de echipamente și integratori de servicii, cu scopul de a demonstra recomandările produse de activitatea de cercetare. În contextul rețelelor de transport

de date fără fir, proiectul WT, care face parte din grupul OTWG (Open Transport Working Group), a terminat cu succes patru astfel de demonstrații de concept [46–49].

Prima astfel de demonstrație a avut loc în octombrie 2015, în Madrid și a fost organizată de Telefonica, împreună cu Universitatea Carlos III. Scopul acesteia a fost de a extinde protocolul OpenFlow cu atribute specifice echipamentelor de transport de date fără fir. Astfel, interfața de Sud folosită a fost OpenFlow, în timp ce echipamentul de control SDN a fost ONOS. Au fost prezentate două cazuri de utilizare: (i) pornirea/oprirea unei interfețe radio în funcție de nivelul de trafic ce trebuie transmis de către echipament, economisind astfel energie în momentele în care nivelul de trafic în rețea nu este foarte ridicat; (ii) schimbarea căilor de date într-un ruter, în cazul în care se pierde pachete pe legătura radio, ca urmare a unor schimbări meteorologice (simulate prin folosirea unui atenuator variabil pe legătura radio).

A doua demonstrație de concept a avut loc în aprilie 2016, la München și a fost organizată de Telefonica. Spre deosebire de prima demonstrație, în cea de-a doua s-a folosit drept interfață de Sud protocolul NETCONF, iar echipamentul de control SDN a fost ODL. Ca model informațional YANG pentru configurarea echipamentelor s-a ales un model pentru microunde simplificat (conținea un număr limitat de atribute). Acesta a fost dezvoltat de către grup și apoi implementat de către echipamentele de la diferiți producători. Au fost demonstrate mai multe cazuri de utilizare folosind aplicații SDN care se bazează pe acel model: detectarea și configurarea de noi echipamente, detectarea și corecția (printr-o acțiune a operatorului) diferențelor între configurația curentă și cea planificată, detecția și vizualizarea rețelei de transport configurate, primirea, afișarea și stocarea evenimentelor și alarmelor din rețea.

Cea de-a treia demonstrație de concept s-a desfășurat în octombrie 2016 și a avut loc în centrul de cercetare WINLAB de la Universitatea Rutgers, fiind organizat de AT&T. S-au ales aceeași interfață de Sud și același echipament de control SDN, scopul acestei demonstrații fiind de a proba utilitatea întregului model informațional pentru microunde (conținea toate atributele pe care le poate avea un echipament de transport de date fără fir). S-a implementat și modelul de bază (*Core Model*), dezvoltat de alt grup din cadrul ONF. Au fost demonstrate aplicațiile dezvoltate pentru cea de-a doua demonstrație, dar și două noi aplicații: una care administrează spectrul, prin compararea și configurarea frecvențelor planificate și cele configurate pe echipamente, realocându-le în caz că nu se potrivesc; cealaltă, denumită automatizarea în bucla închisă, demonstrează un răspuns simplist la anumiți factori declanșatori (interni, externi sau temporali).

A patra demonstrație de concept a avut loc în iunie 2017, la Bonn și a fost organizată de Deutsche Telekom. S-au folosit mai multe modele informaționale: modelul pentru microunde, modelul de bază, un model Ethernet simplificat și un model pentru sincronizare - pentru PTP (Precise Time Protocol), plecând de la un model dezvoltat de ITU-T. Astfel, s-au putut demonstra mai multe cazuri de utilizare: administrarea echipamentelor care transportă date fără fir, administrarea indicatorilor de performanță ai echipamentelor, administrarea puterii echipamentelor, administrarea echipamentelor capabile Ethernet, recalcularea căilor de trafic Ethernet într-o rețea sau administrarea echipamentelor care suportă PTP.

## 2.2.2 IETF

Un alt SDO care prestează activități de standardizare în domeniul rețelelor definite prin software este IETF. Această organizație, în general, oferă standardele de bază folosite în Internet. Obiectivul lor declarat este îmbunătățirea funcționării Internetului prin dezvoltarea de documente tehnice relevante și de înaltă calitate care să ghideze proiectarea, utilizarea și administrarea Internetului.

Doar puține din grupurile de lucru din cadrul IETF se ocupă cu activități care au legătură cu tehnologia SDN. Unul dintre acestea a dezvoltat ForCES, inițiativă ce separa planurile de date și de control ale echipamentelor [50], cu câțiva ani înainte ca protocolul OpenFlow, mult mai cunoscut, să propună asta într-un mod care să permită o adoptare mai rapidă în industrie.

Alt grup de lucru se ocupă de *interfața către sistemul de rutare* - I2RS (Interface to the Routing System). Acesta își propune să dezvolte interfețe pentru ca aplicațiile SDN să poată accesa sistemul de rutare al rețelei. Se dorește ca beneficiarii I2RS să fie aplicații de administrare, echipamente de control SDN sau aplicații de utilizator care au cereri specifice la adresa rețelei. Acestea ar permite informațiilor, politicilor și parametrilor operaționali să fie introduși sau extrași din sistemul de rutare, fără a afecta consistența datelor și coerența infrastructurii de rutare. Această abordare poate fi privită ca un rival al protocolului OpenFlow, cu mențiunea că se aplică doar de la nivelul 3 în sus, în stiva OSI.

În interiorul grupului de lucru SPRING (Source Packet Routing in Networking) se discută un control al rutării care să permită indicarea unei căi de dirijare încă de la sursa de trafic [37], calcularea acesteia putând fi făcută atât centralizat, cât și distribuit. Tot în cadrul IETF se dezvoltă și protocolul NETCONF, însă acesta va fi detaliat într-o secțiune viitoare.

Parte din IETF este și IRTF (Internet Research Task Force). Aceasta este o organizație paralelă, care se ocupă mai mult de partea de cercetare, sprijinind proiectele care se crede că vor fi benefice comunității Internetului, dar care nu sunt încă pregătite pentru implementare. Aici există un grup de lucru care investighează tehnologia SDN din mai multe perspective, cu scopul de a recunoaște atât abordări ce pot fi definite și lansate pe termen scurt, dar și provocări viitoare: SDNRG (Software-Defined Networking Research Group). Sunt de interes probleme ca extensibilitatea soluției, abstractizări sau limbaje de programare ce pot fi folosite în contextul SDN. Se dorește ca acest grup să ofere un spațiu comun tuturor cercetătorilor cu interes în acest domeniu.

## 2.3 SDN în contextul rețelelor actuale

Chiar dacă nu este o tehnologie ajunsă la maturitate în toate aspectele unei rețele, paradigma SDN este deja folosită în rețele de producție de anumite tipuri. Autorii din [51] evidențiază faptul că evoluția SDN este susținută de către trei mari porțiuni ale industriei rețelisticii: furnizorii serviciilor de tip *cloud* (care utilizează mari centre

de date), furnizorii de servicii de telecomunicații și întreprinderile, așa cum se poate observa și în Figura 2.3.

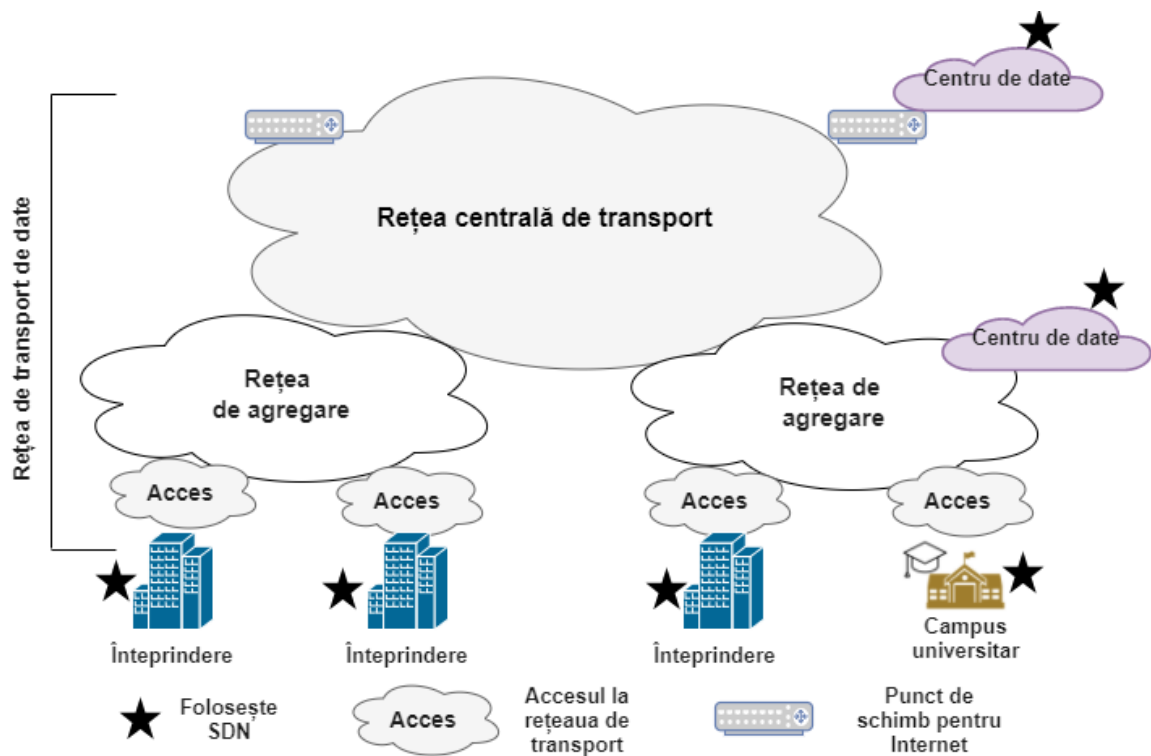


Fig. 2.3: Implementarea SDN în contextul rețelelor actuale [51].

Astfel, SDN este implementat deja cu succes în rețelele unor campusuri universitare (Universitatea Stanford, de exemplu, dat fiind faptul că această paradigmă își are originile acolo), în mari centre de date sau în întreprinderi. În cazul rețelelor de transport, încă se duc activități de standardizare care să permită operatorilor rețelelor de telecomunicații să implementeze această tehnologie în cel mai scurt timp, beneficiind astfel de avantajele pe care aceasta le oferă (cel mai important, din punctul lor de vedere, fiind reducerea costurilor).

Această secțiune va prezenta în continuare utilizarea SDN la momentul actual, în mai multe contexte: în rețelele din marile centre de date, în rețelele de tip hibrid, care combină rețelele tradiționale cu SDN sau prin studii de caz, implementări sau experimente efectuate pe teren (în rețele de producție).

### 2.3.1 SDN în centrele de date

Centrele de date reprezintă grupări de calculatoare cu putere mare de procesare și capacitate mare de stocare. Această idee nu este una nouă, fiind prezentă de câteva decenii, însă, în timp, au evoluat caracteristicile calculatoarelor, precum și numărul lor într-un centru de date. În zilele noastre acestea au ajuns să conțină mii sau chiar zeci de mii de mașini fizice (servere) [52].

Creșterea numărului de servere și a puterii de stocare, combinate cu creșterea vitezei și a lărgimii de bandă din rețea a dus la necesitatea de a stoca din ce în ce mai multă informație în centre de date tot mai mari. În mod natural, acest lucru va însemna combinarea centrelor de date pentru a forma altele mai mari. Pe lângă acest aspect, se pune foarte mare accent în ultimul timp pe tehnologii de virtualizare, care permit unui server rularea mai multor mașini virtuale, pentru diferite aplicații sau servicii ale utilizatorilor.

A fost creat astfel un mediu dinamic, atât în cadrul centrelor de date, cât și între acestea. După cum este prezentat și în [53], operații precum prevenirea sau recuperarea în caz de dezastru, sau echilibrarea încărcării serverelor au nevoie de o creștere a traficului între centrele de date, lucru ce duce la o administrare mai complexă a acelei rețele.

Centrele de date sunt folosite și în cazul oferirii de servicii de virtualizare de servere. Operațiunile într-un astfel de mediu implică lucrul cu mașini virtuale și, de cele mai multe ori, necesitatea migrării acestora între diferite mașini sau centre de date. Asta înseamnă că diferite aplicații sau servicii trebuie să aibă o imagine proprie asupra rețelei dintre servere, care este doar o imagine la nivel logic, în comparație cu rețeaua fizică existentă. Așa cum este prezentat și în [1], crearea de astfel de rețele logice se poate face și fără SDN, prin metode folosite și în rețelele tradiționale, cum ar fi cu ajutorul unor VLAN-uri. Însă, în cazul centrelor de date foarte mari, acestea pot fi insuficiente. Cum ele pot fi exprimate într-un pachet Ethernet doar prin 12 biți, numărul maxim de rețele virtuale este 4096. Apare astfel nevoia unor alte metode, care sunt însă mai complexe de configurat.

O metodă pentru crearea de rețele logice care să fie prezentate unor aplicații sau servicii este chiar folosirea tehnologiei SDN, făcând abstracție de rețeaua fizică. În [53–56] se prezintă diferite metode care pot fi aplicate în astfel de cazuri, sau care chiar au fost demonstrate.

Soluția SDN a fost chiar aplicată în rețele de producție. Așa cum se explică în [57], centrele de date ale Google folosesc deja această soluție, cu ajutorul protocolului OpenFlow. Există două laturi ale rețelei de arie largă - WAN (Wide Area Network) folosită de Google: o parte legată la Internet, care conține traficul utilizatorilor și o parte internă, care transportă traficul dintre centrele de date. Cele două au caracteristici diferite ale traficului și, implicit, nevoi diferite. În acea rețea internă a fost implementat SDN, prin protocolul OpenFlow. Avantajele evidențiate de Google includ o viziune de ansamblu asupra rețelei, un răspuns mai rapid la erori, timp mai scurt de implementare, actualizări mai rapide ale software-ului rețelei, fără pierdere de trafic sau existența unui mediu de test de mare fidelitate. Chiar dacă în 2012, atunci când a fost implementată această tehnologie în rețea, protocolul OpenFlow era încă la început și nu avea toate facilitățile pe care le oferă astăzi, Google a arătat încă de atunci că tehnologia SDN poate fi folosită, aducând foarte multe avantaje, în multe cazuri de utilizare din industria rețelisticii.

### 2.3.2 SDN în rețelele hibride

Rețelele hibride reprezintă o combinație între rețelele tradiționale și SDN, având scopul de a combina avantajele aduse de ambele, cât timp tehnologia SDN nu este încă destul de matură și prezintă unele dezavantaje. Un alt motiv pentru apariția acestor rețele hibride este imposibilitatea de a schimba rețelele de producție într-un timp foarte scurt, în același timp asigurând și funcționarea acestora în parametrii agreeți.

Autorii din [58] prezintă oportunitățile și provocările aduse de o astfel de abordare. Ei propun patru modele ale rețelelor hibride: (i) rețele hibride bazate pe topologie, (ii) rețele hibride bazate pe servicii, (iii) rețele hibride bazate pe clase și (iv) rețele hibride integrate. Avantajele prezentate de aceste modele hibride sunt flexibilitatea, robustețea, extensibilitatea, costuri mai mici ale implementării, date de faptul că nu toată rețeaua este actualizată. Pe de altă parte, această abordare vine și cu dezavantaje, reprezentate de nevoia de a administra paradigme eterogene și garantarea faptului că interacțiunea dintre ele este benefică.

În [59–61] se propun sisteme care să permită o implementare incrementală a SDN în rețele de tip hibrid pentru rețele ale întreprinderilor și ale furnizorilor de servicii. Autorii evidențiază faptul că implementarea directă, într-o rețea de producție a acestei noi tehnologii este imposibilă și ar conduce atât la probleme de implementare, cât și la probleme operaționale în rețea. Ei susțin că implementarea SDN în puncte cheie ale rețelelor va aduce avantajele acestei tehnologii, fără nevoia de a înlocui toată rețeaua de la început. Pentru a determina aceste puncte cheie, autorii propun un *planificator al implementării*, care să analizeze topologia de rețea, împreună cu informațiile istorice despre trafic și constrângeri de resurse.

Lucrarea [62] prezintă o altă abordare pentru aceste rețele hibride. Autorii propun un echipament de control al rețelei care să poată interacționa atât cu echipamentele care folosesc un control centralizat, cât și cu cele care au un control distribuit. Controlarea echipamentelor vechi din rețea, care nu suportă protocolul OpenFlow se face chiar cu ajutorul acestuia. Astfel, dacă se vrea alterarea fluxului de date dintr-un echipament vechi, autorii propun folosirea unei extensii a OpenFlow, numită *LegacyFlowMod*, care să instruiască un echipament OpenFlow să trimită un alt pachet special către echipamentul vechi, astfel încât acesta să își schimbe tabela de dirijare.

Autorii din [63, 64] susțin de asemenea că o soluție hibridă este mai potrivită. Propun astfel folosirea unor rețele care să combine SDN cu protocolul OSPF (Open Shortest Path First), păstrând avantajele date de acest protocol vechi de rutare, care a fost folosit cu succes câteva zeci de ani și, în același timp, aducând și câteva din avantajele propuse de SDN.

### 2.3.3 Studii de caz. Implementări. Experimente în rețele de producție

Există numeroase studii de caz sau experimente în rețele de producție care demonstrează utilitatea SDN și faptul că această tehnologie poate fi implementată cu succes în rețelele din zilele noastre.



În [65] se prezintă soluția implementată în rețeaua spitalului *Kanazawa University*, din Japonia. S-a folosit tehnologia SDN, cu ajutorul protocolului OpenFlow pentru a ușura munca de administrare a rețelei spitalului, care devenea din ce în ce mai complexă, pe măsură ce noi echipamente medicale trebuiau adăugate. Rețeaua, fiind operată de personalul spitalului, era vulnerabilă la erorile provocate de greșelile umane. Trebuia asigurată izolarea între anumite departamente ale spitalului, în unele cazuri, astfel că administrarea rețelei devenea greoaie. Cu ajutorul SDN a fost creată o soluție în care infrastructura de rețea a devenit mai ușor de administrat, prin aplicații software care rulează deasupra echipamentului de control SDN, permițând departamentelor crearea de rețele virtuale proprii, asigurând în același timp conectivitate între departamente. Astfel, rețeaua spitalului a devenit stabilă și pregătită pentru dezvoltările rapide care apar în industria medicală în zilele noastre.

Lucrarea [66] ilustrează folosirea SDN în cadrul unei rețele optice de transport a unui furnizor de servicii prin Ethernet. Autorii au implementat această tehnologie pentru oferirea de servicii în acea infrastructură de rețea. Obiectivele urmărite au fost simplificarea administrării rețelei și furnizarea de trafic, în funcție de serviciul solicitat (de exemplu, lățime de bandă la cerere).

În [67] se analizează posibilitatea folosirii tehnologiei SDN în rețelele Ethernet industriale. Autorul evidențiază punctele în care această tehnologie s-ar putea aplica în rețelele industriale și avantajele pe care o astfel de implementare le-ar aduce.

Autorii din [68] urmăresc evoluția implementărilor SDN pornind de la nivelul unei rețele dintr-un simplu laborator din cadrul Universității Stanford și până la rețele de nivel național. Cu ajutorul acestor studii se evidențiază compromisurile care apar în implementarea acestei noi tehnologii în rețele de producție. De exemplu, pentru o simplă rețea a unui campus universitar, un singur server poate susține tot planul de control al rețelei, incluzând echipamentul de control și diferitele aplicații care rulează acolo pentru administrarea acesteia. Primele cazuri de utilizare demonstrate cu ajutorul SDN au fost rutarea pe cea mai scurtă cale de nivel 2, învățarea adreselor de nivel 2 - MAC (Medium Access Control), descoperirea topologiei cu ajutorul LLDP (Link Layer Discovery Protocol) sau colectarea de statistici de la comutatoarele din rețea. Aceste experimente au subliniat câteva aspecte importante pentru performanțelor rețelelor SDN: timpul necesar configurării fluxurilor de date în rețea, numărul de intrări în tabelele de fluxuri ale comutatoarelor. De asemenea, este important ca un comutator să fie hibrid, adică să suporte atât protocolul OpenFlow, cât și pe cele tradiționale. O altă fază a experimentelor conduse de acești autori a constatat în împărțirea infrastructurii de rețea în mai multe rețele logice destinate diferitelor aplicații.

Toate aceste experimente și studii de caz prezentate au avut la bază protocolul OpenFlow, ducând la evoluția acestuia. Acest lucru nu trebuie să ne ducă însă cu gândul că tehnologia SDN înseamnă doar OpenFlow. După cum se va vedea în capitolele următoare, în special în cazul rețelelor de transport de date prin microunde, există și alte protocole cu ajutorul cărora se pot aduce principiile SDN în rețea.

## Capitolul 3

# Unelte SDN în contextul rețelelor de transport de date fără fir

Așa cum a fost prezentat în capitolul anterior, o mare parte din cercetarea și implementările SDN se bazează pe protocolul OpenFlow. Acesta, însă, nu poate fi utilizat în orice aspect al unei rețele (de exemplu în rețelele de transport de date fără fir). În [46] s-a demonstrat faptul că se poate extinde protocolul OpenFlow pentru a cuprinde atribute specifice echipamentelor de transport de date fără fir, însă s-a ajuns la concluzia că este totuși nevoie de un model informațional care să abstractizeze astfel de echipamente pentru a facilita administrarea acestora prin aplicații software.

Astfel, grupurile de lucru din ONF au formulat recomandări pentru astfel de modele informaționale care să poată fi aplicate în acest context. În martie 2015 a fost publicată de către ONF prima versiune (1.0) a modelului informațional de bază, TR-512, [69], apoi în noiembrie 2015 versiunea 1.1 [70], ca în septembrie 2016 să fie publicată versiunea curentă, 1.2, purtând numele TR-512.1 [71]. Modelul informațional de bază este doar un schelet, care poate fi folosit în toate tipurile de rețele de transport, indiferent de natura acestora. Pentru rețelele de transport de date fără fir, ONF a publicat în decembrie 2016 și modelul informațional pentru microunde [72], pentru abstractizarea echipamentelor din acest tip de rețele, care este integrat cu TR-512.1.

În următoarele secțiuni aceste modele vor fi detaliate, pentru a putea mai apoi înțelege arhitectura simulatoarelor dezvoltate în această lucrare. Apoi va fi prezentat protocolul NETCONF și modul în care utilizează aceste modele informaționale, precum și alegerea unui cadru software cu sursă deschisă care să implementeze un server pentru acest protocol. În finalul capitolului se va prezenta arhitectura demonstrațiilor de concept desfășurate în cadrul ONF, pentru o înțelegere mai bună a motivației din spatele creării simulatoarelor care fac scopul acestei lucrări.

### 3.1 ONF TR-512.1 - Modelul informațional de bază (*Core Model*)

Modelul informațional de bază (*CoreModel*) reprezintă o recomandare tehnică făcută de grupul de Modelare Informațională (*Information Modeling*) din cadrul ONF. Aceasta propune un model care să descrie resursele din planul de date al unei rețele de transport, indiferent de tehnologia folosită, cu scopul de a fi folosit în activitățile de control și administrare.

Un model informațional descrie elementele dintr-un domeniu, în ceea ce privește obiectele, proprietățile lor (reprezentate prin attribute) și relațiile dintre acestea [69]. Scopul dezvoltării unui astfel de model este acela de a fi folosit de către echipamentele de control SDN, pentru administrarea automată a unor astfel de rețele de transport, în concordanță cu arhitectura SDN. Echipamentul de control va prezenta cu ajutorul acestui model viziunea sa asupra rețelei către clienții acestuia (care pot fi aplicații software sau alte echipamente de control).

Modelul informațional de bază propune un set de obiecte de bază care reprezintă planul de date al unei rețele, care sunt însă independente de tehnologia folosită pentru transportul datelor. Acestea pot fi apoi folosite pentru dezvoltarea de modele informaționale specifice pentru anumite tehnologii (de exemplu tehnologii fără fir - microunde sau unde milimetrice, tehnologii optice, etc.). Modelul conține și obiecte care pot fi folosite în aplicații specifice, însă toate acestea sunt independente de protocoalele care ar putea fi folosite în planul de control. El este propus în limbajul UML (Unified Modeling Language) și se bazează și pe alte modele informaționale, propuse de alte organizații care dezvoltă standarde.

O imagine de ansamblu simplificată, folosind UML se poate vedea în Figura 3.1. Blocurile relevante pentru simulatoarele dezvoltate vor fi detaliate în paragrafele următoare. Nu se vor detalia toate obiectele care alcătuiesc modelul informațional de bază deoarece, așa cum este sugerat în recomandarea ONF, modelul poate fi redus și simplificat, în funcție de nevoile pentru care este folosit.

#### 3.1.1 Obiectul Element de Rețea (*NetworkElement - NE*)

Obiectul *Element de Rețea* - NE (Network Element) - reprezintă un dispozitiv de rețea, adică, în cazul SDN, un echipament de dirijare din planul de date, sau, în cazul în care există virtualizare, un element virtual de rețea vizibil de către interfața care folosește acea virtualizare.

Pentru o interfață directă de la echipamentul de control SDN către un echipament de rețea, obiectul *NE* delimitează domeniul de control pentru resursele echipamentului, specificând funcții cum ar fi încapsularea folosită, multiplexarea, demultiplexarea, funcțiile asociate operațiilor de administrare și de mentenanță, etc. De asemenea, obiectul *NE* definește domeniul spațiilor de adresare pentru identificarea obiectelor care reprezintă resursele conținute de echipamentul respectiv.

În cazul în care se folosesc metode de virtualizare, obiectul *NetworkElement* reprezintă un element virtual de rețea. Asocierea unui element virtual cu unul real

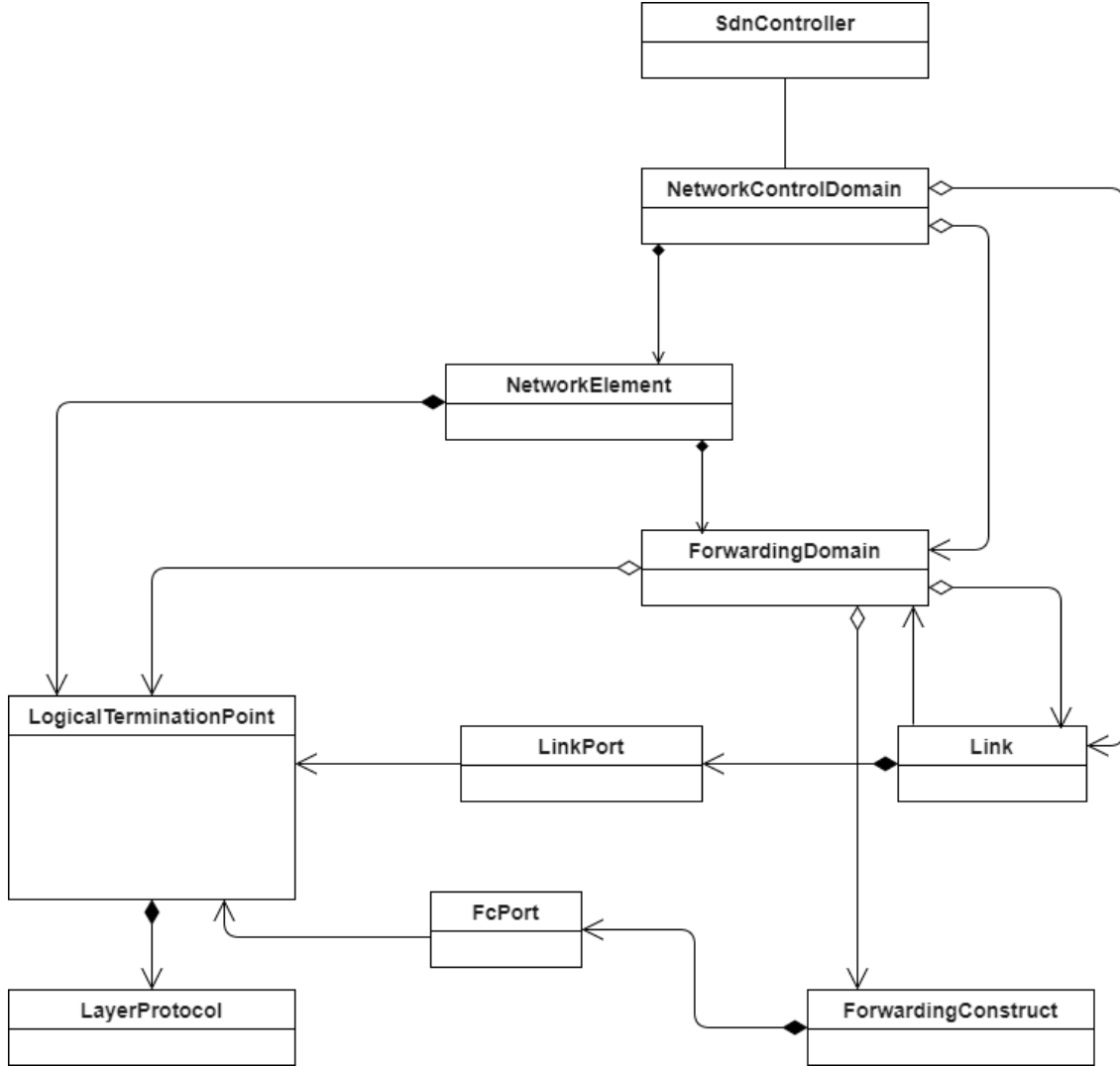


Fig. 3.1: Reprezentare UML simplificată a *CoreModel* [71].

este responsabilitatea echipamentului de control SDN. Cu ajutorul interfeței de Sud acesta poate crea sau șterge în mod dinamic astfel de obiecte pentru a oferi diferite imagini asupra rețelei, în funcție de nevoile aplicațiilor care se află deasupra echipamentului de control.

### 3.1.2 Obiectele Punct Logic de Terminație (*LogicalTerminationPoint* - *LTP*) și Nivelul Protocolului (*LayerProtocol* - *LP*)

Obiectul *Punct Logic de Terminație* - *LTP* (*Logical Termination Point*) - cuprinde terminațiile, adaptările sau funcțiile asociate operațiilor de administrare și de mentenanță ale unuia sau mai multor niveluri de transport. Prin natura sa, acest obiect suportă toate tipurile de protocoale, inclusiv cele pentru comutare de pachete

sau pentru comutare de circuite. Fiecare nivel de transport este reprezentat de o instanță a unui obiect de tip *Nivelul Protocolului - LP (Layer Protocol)* -, instanță care poate fi folosită pentru a controla terminațiile sau funcțiile de administrare și mentenanță ale nivelului respectiv, sau pentru adaptarea (încapsularea sau multiplexarea semnalului client).

Dacă relația client-server între resursele echipamentului reprezentate de obiectele *LTP* și *LP* are un grad de asociativitate de 1:1 și este imuabilă, atunci un obiect *LTP* poate conține mai multe obiecte *LP*, de niveluri diferite de transport. Altfel, obiectele *LP* de pe niveluri diferite trebuie să aibă asociate obiecte *LTP* diferite.

Scopul acestor obiecte este acela de a oferi suport din perspectiva controlului și a administrării fără a fi nevoie de a defini atribute specifice unei anumite tehnologii, permițând astfel extinderea modelului fără a depinde de proprietățile tehnologiei respective, oferind deci o flexibilitate sporită.

Un atribut important al obiectelor *LP* este reprezentat de către *numele nivelului protocolului - layerProtocolName*. Acesta reprezintă nivelul de transport al obiectului și poate avea următoarele valori, conform [71]:

- Nivel 0: Comutator Optic de Protecție - OPS (Optical Protection Switch), Secțiune Optică de Transmisie - OTS (Optical Transmission Section), Secțiune Optică Multiplex - OMS (Optical Multiplex Section), Canal Optic - OCh (Optical Channel);
- Nivel 1: Unitate a Canalului Optic de Transport - OTU (Optical channel Transport Unit), Unitate de Date Optică - ODU (Optical Data Unit);
- Nivel 2: Ethernet Purtător - *Carrier Ethernet*: Nivelul Fizic Ethernet - ETY (Ethernet Physical Layer), Nivelul MAC Ethernet - ETH (Ethernet MAC Layer); Profilul de Transport al protocolului MPLS - MPLS-TP (Multi-Protocol Label Switching Transport Profile);
- Proprietăți specifice nivelului de transport asociate cu obiectul *LP*.

### 3.1.3 Obiectul Construct de Dirijare (*ForwardingConstruct - FC*)

Obiectele de tip Construct de Dirijare - *FC (Forwarding Construct)* sunt folosite pentru a realiza dirijarea informației caracteristice nivelului de transport dat de obiectul *LP* și oferă posibilitatea de a permite dirijarea între două sau mai multe obiecte *LTP*. Astfel, obiectele *FC* sunt independente de nivelul de transport folosit și suportă orice formă de pachete sau circuite.

Asocierea între *FC* și *LTP* se face prin port-uri, în care fiecare dintre acestea are un rol în contextul obiectului *FC*. Dirijarea traficului între port-urile asociate se face în funcție de tipul de obiect *FC*. Un astfel de obiect poate fi asociat unui singur obiect de tip Domeniu de Dirijare - *FD (Forwarding Domain)*. Ele pot fi definite recursiv (un obiect *FC* poate fi parte din alt obiect *FC*), însă la cel mai mic nivel

de recursivitate acesta reprezintă de fapt o legătură în matricea de comutatoare a elementului de rețea.

Obiectele *FC* pot fi folosite pentru a reprezenta orice fel de conexiune, cum ar fi punct la punct, punct la multi-punct sau multi-punct la multi-punct [71].

### 3.1.4 Obiectul Port al Constructului de Dirijare (*FC Port*)

Obiectele *port FC* sunt folosite, așa cum a fost prezentat anterior, la asocierea dintre obiectele *FC* și *LTP*. Dirijarea traficului între aceste obiecte se face conform tipului de *FC*. De exemplu, *port FC* poate reprezenta un punct protejat (de încredere) sau un punct care protejează (de rezervă), în cazul în care rolul obiectului *FC* este unul de protecție.

### 3.1.5 Obiectul Domeniu de Dirijare (*ForwardingDomain - FD*)

Domeniul de dirijare - *FD* - este un obiect care modelează componenta topologiei care permite dirijarea pachetelor între diferite puncte ale resurselor echipamentului de rețea (reprezentate prin alte obiecte, de tipul *LTP*). Lista punctelor logice de terminatie care pot fi folosite de către un domeniu de dirijare este parte a acestui obiect. *FD* poate conține zero sau mai multe obiecte de tip *FC*, indiferent de nivelul la care se află acestea (Ethernet, MPLS, optic, etc.).

Acest domeniu de dirijare oferă contextul pentru crearea, modificarea sau ștergerea obiectelor de tip *FC* [71]. Obiectul *FD* dintr-un element de rețea poate reprezenta comutatorul sau gruparea de comutatoare din echipamentul de dirijare.

## 3.2 ONF TR-532 - Modelul informațional pentru microunde (*Microwave Information Model*)

Modelul informațional pentru microunde [72] a apărut în decembrie 2016 ca o recomandare formulată de grupul OTWG din cadrul ONF. Scopul acestuia este de a modela un echipament de transport de date fără fir, pentru a putea fi folosit de echipamentele de control SDN, în încercarea de a asigura o independență față de producătorii de echipamente. Chiar dacă este denumit *model informațional pentru microunde*, acesta poate fi aplicat fără probleme nu numai echipamentelor ce funcționează în spectrul microundelor, ci și echipamentelor care funcționează în benzi de frecvență mai înalte (lungimi de undă milimetrice), care încep să își facă tot mai mult simțită prezența în rețelele actuale de transport.

TR-532 este de fapt o extensie specifică tehnologiei WT a modelului informațional de bază, versiunea 1.2 (TR-512.1). Legătura cu acesta se face prin extinderea clasei de obiecte *LP*. Astfel, modelul informațional pentru microunde conține șase pachete condiționale caracteristice tehnologiilor folosite pentru transport, care au în nume extensia *\*\_Pac*:

- *MW\_AirInterface\_Pac*;
- *MW\_AirInterfaceDiversity\_Pac*;
- *MW\_PureEthernetStructure\_Pac*;
- *MW\_HybridMWStructure\_Pac*;
- *MW\_EthernetContainer\_Pac*;
- *MW\_TdmContainer\_Pac*

O imagine de ansamblu simplificată a acestui model, în limbajul UML, care conține doar obiectele relevante pentru simulatoarele dezvoltate, împreună cu legătura acestuia cu modelul informațional de bază este ilustrată în Figura 3.2.

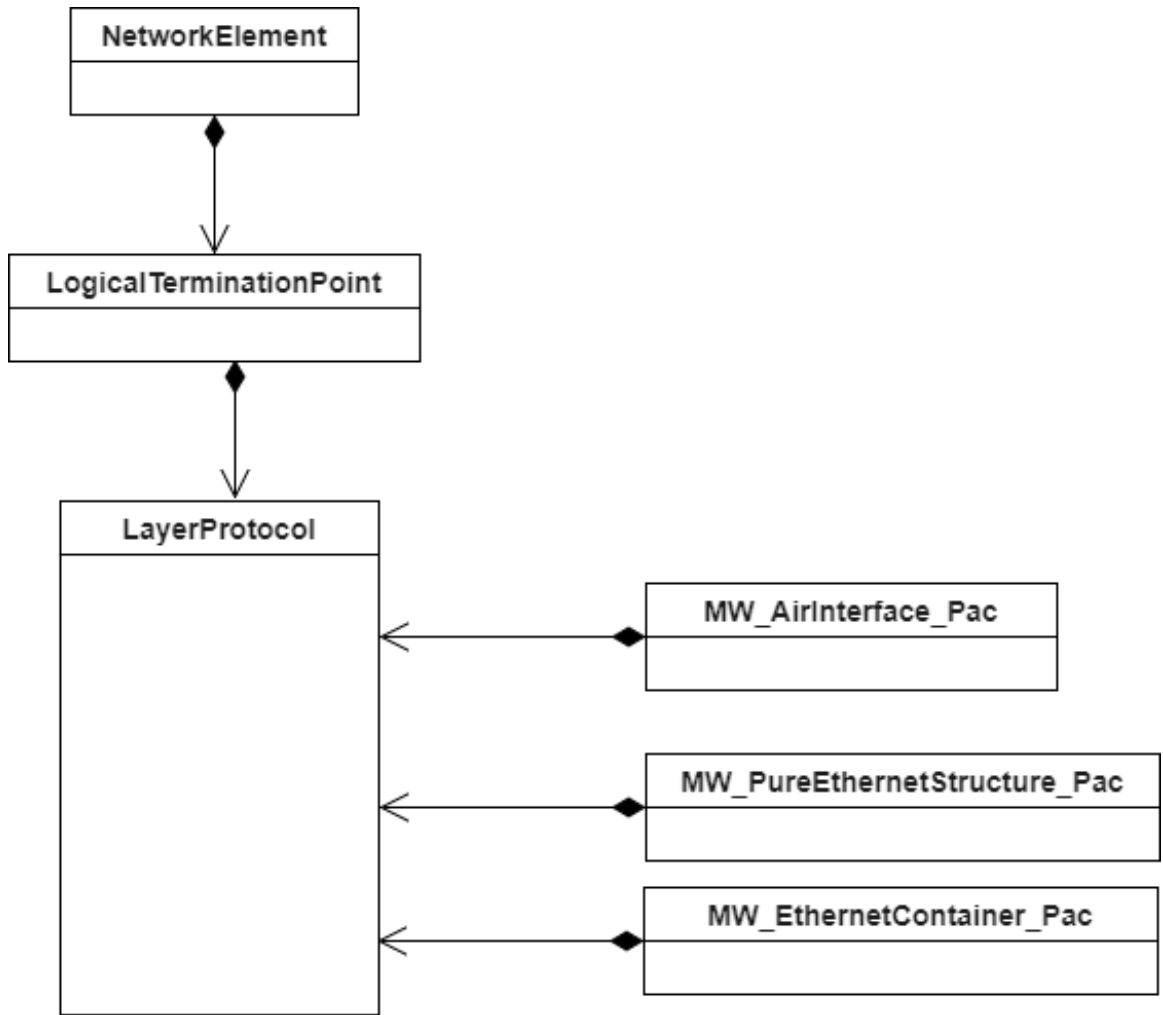


Fig. 3.2: Reprezentare UML simplificată a *MicrowaveModel* și legătura acestuia cu *CoreModel* [72].

În următoarele paragrafe se vor detalia obiectele acestui model care sunt importante din punctul de vedere al simulatoarelor dezvoltate în această lucrare.

### 3.2.1 Obiectul *MW\_AirInterface\_Pac*

Obiectul *MW\_AirInterface\_Pac* reprezintă o interfață radio fizică a unui echipament. Este denumit în recomandare ca *punct de terminație a traseului secțiunii fizice de microunde* - MWPS-TTP (Microwave Physical Section - Trail Termination Point), astfel că nivelul de transport al obiectului *LP* asociat este Secțiunea Fizică de Microunde - *MWPS (Microwave Physical Section)* [72]. O reprezentare simplificată în limbajul UML a *MW\_AirInterface\_Pac* se poate observa în Figura 3.3.

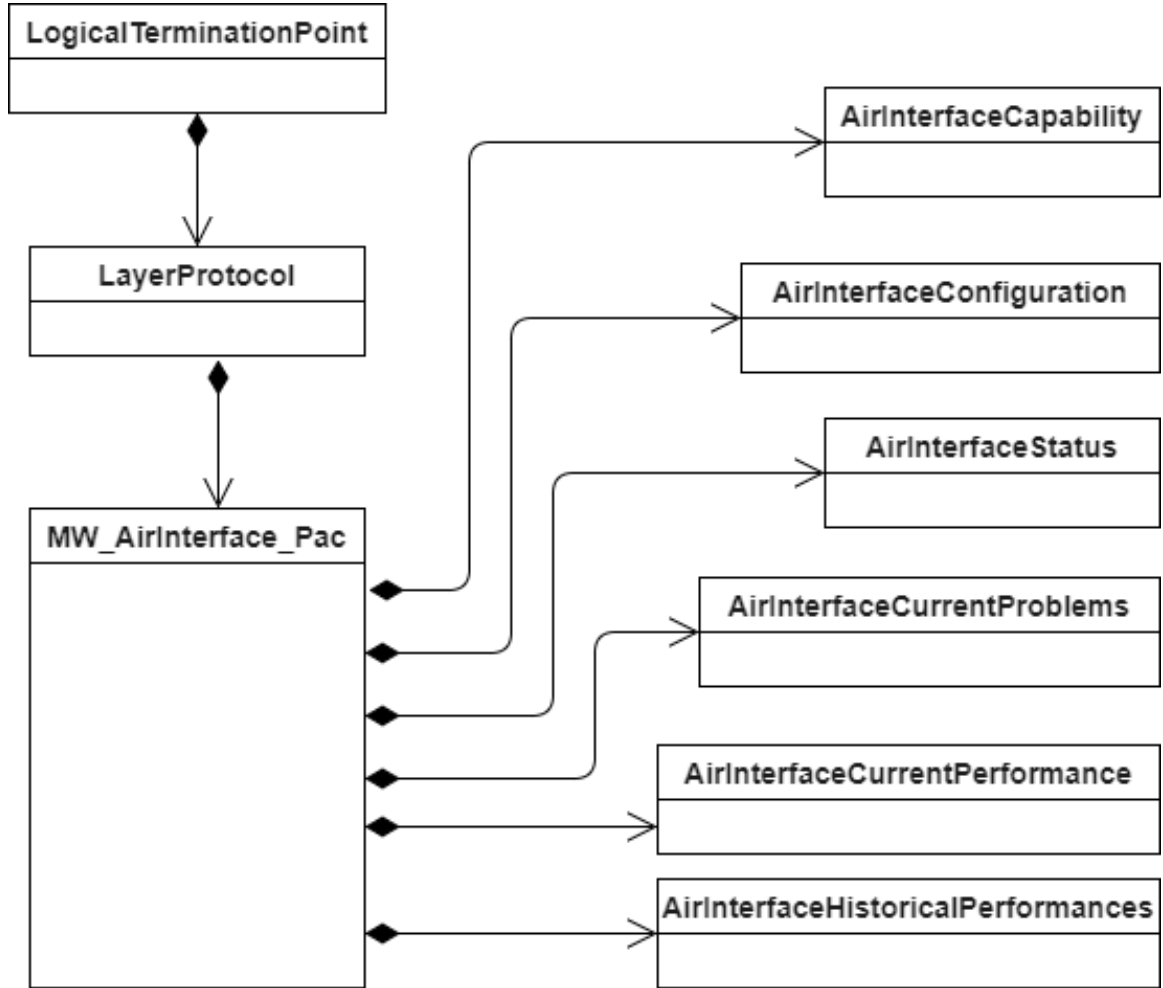


Fig. 3.3: Reprezentare UML simplificată a obiectului *MW\_AirInterface\_Pac* [72].

Acest obiect conține alte câteva obiecte care modelează caracteristice unei interfețe radio fizice, cum ar fi: (i) capacități ale modemului și ale transmițătorului interfeței radio asociate (de exemplu tipurile de modulație suportate pentru transmisie, valorile admisibile ale puterii de transmisie, intervalul de frecvențe suportate de emițător sau de receptor, alarmele expuse de interfață, suportul interfeței pentru modulație adaptivă, etc.), (ii) parametrii configurabili ai interfeței radio (de exemplu numele interfeței, lărgimea de bandă a canalului de transmisie/de recepție, frecvențele folosite pentru transmisie/recepție, puterea de transmisie, intervalul în care modulația poate



lua valori, diferite alte caracteristici configurabile ale interfeței, cum ar fi Anularea Interferenței dintre Polarizări - XPIC (Cross Polarization Interference Cancellation), Intrări Multiple - Ieșiri multiple - MIMO (Multiple-Input Multiple-Output), criptarea datelor, etc.), (iii) parametrii care descriu starea interfeței la un anumit moment de timp (de exemplu frecvențele actuale de transmisie/recepție, nivelurile actuale de putere a semnalului de transmisie/recepție, modulația actuală folosită, raportul semnal-zgomot măsurat de către modem, temperatura actuală a unității radio, etc.), (iv) problemele actuale ale interfeței radio (adică alarmele care apar pe interfață la un moment dat), (v) valorile actuale ale parametrilor de performanță a interfeței și (vi) valorile istorice ale parametrilor de performanță a interfeței [72].

### 3.2.2 Obiectul *MW\_PureEthernetStructure\_Pac*

Obiectul *MW\_PureEthernetStructure\_Pac* este o reprezentare logică a unei interfețe radio capabilă să transporte doar trafic Ethernet. Acest obiect este reprezentat într-un mod simplificat, în limbajul UML, în Figura 3.4. Asocierea cu o interfață fizică radio se face la nivelul modelului informațional de bază, printr-o relație de tip client-server.

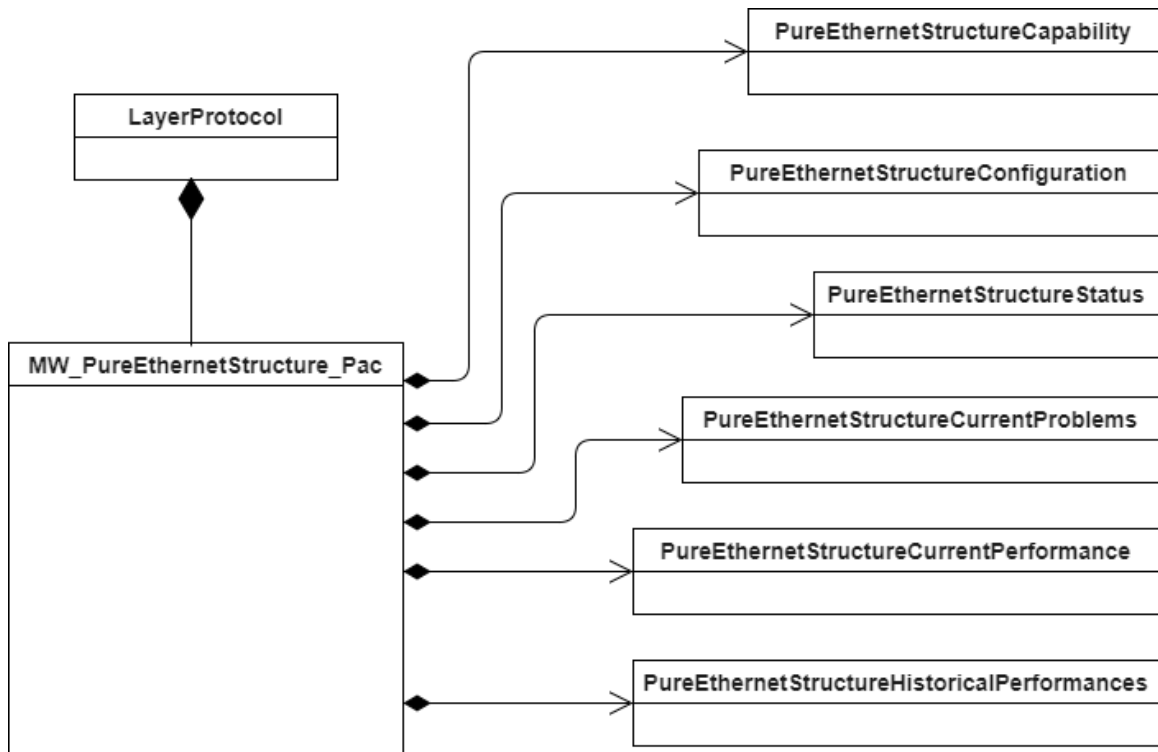


Fig. 3.4: Reprezentare UML simplificată a obiectului *MW\_PureEthernetStructure\_Pac* [72].

Acest obiect este denumit în recomandare ca *punct de terminare a traseului secțiunii microunde* - MWS-TTP (Microwave Section - Trail Termination Point),

astfel că nivelul de transport al obiectului *LP* asociat este Secțiunea de Microunde - *MWS* (*Microwave Section*).

Structura obiectelor conținute de către *MW\_PureEthernetStructure\_Pac* este similară cu cea a obiectului *MW\_AirInterface\_Pac*. Conține obiecte care reprezintă (i) capacitățile acestei interfețe logice (de exemplu alarmele aplicabile ei sau identificatorul structurii respective, care poate fi folosit de alte obiecte), (ii) parametrii configurabili ai interfeței logice (de exemplu gradul de severitate a alarmelor pe care această interfață le expune), (iii) parametrii care descriu starea interfeței logice la un anumit moment de timp, (iv) problemele actuale ale interfeței logice, (v) valorile actuale ale parametrilor de performanță a interfeței logice și (vi) valorile istorice ale parametrilor de performanță a interfeței logice [72].

### 3.2.3 Obiectul *MW\_EthernetContainer\_Pac*

Obiectul *MW\_EthernetContainer\_Pac* reprezintă de asemenea o interfață logică și este denumit în recomandare *punct de terminare a conexiunii unui client de microunde*, pentru un semnal Ethernet client. Practic, este o interfață logică ce are rol de container pentru traficul Ethernet care este transmis de echipament prin radio. În raport cu obiectul *LP* acesta are un nivel de transport denumit Container Ethernet - *ETC* (*Ethernet Container*). O reprezentare grafică simplificată în limbajul UML a obiectului *MW\_EthernetContainer\_Pac* poate fi găsită în Figura 3.5.

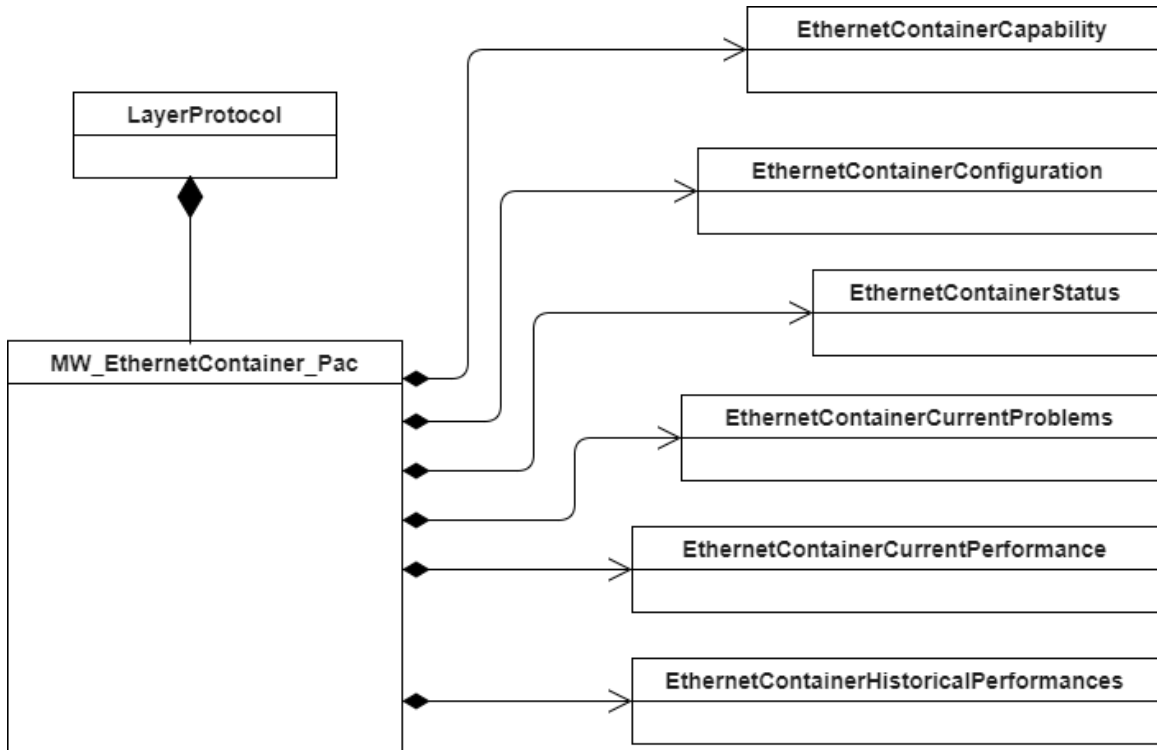


Fig. 3.5: Reprezentare UML simplificată a obiectului *MW\_EthernetContainer\_Pac* [72].

Și în cazul obiectului *MW\_EthernetContainer\_Pac* se păstrează aceeași structura a obiectelor pe care le conține, ca în cazul celorlalte două obiecte detaliate anterior. Astfel, acesta prezintă obiecte care reprezintă (i) capacitățile containerului (de exemplu dacă există compresie la diferite niveluri, criptare a datelor sau alarmele pe care această interfață le expune), (ii) parametrii configurabili ai containerului (de exemplu un identificator al containerului, identificatoarele segmentelor folosite pentru a transporta traficul Ethernet asociat acestui container, etc.), (iii) parametrii care descriu starea containerului, (iv) alarmele la momentul actual de timp pe care containerul le raportează, (v) valorile actuale ale parametrilor de performanță a containerului și (vi) valorile istorice ale parametrilor de performanță a containerului [72].

### 3.3 Protocolul NETCONF și limbajul YANG

Protocolul de Configurare a Rețelei - *Network Configuration Protocol (NETCONF)* - este un protocol de administrare a echipamentelor de rețea care oferă posibilitatea de a instala, manipula și șterge configurații ale echipamentelor de rețea. A fost prima oară definit în RFC 4741 [73], în decembrie 2006, ca mai apoi să fie revizuit în RFC 6241 [74], în iunie 2011. A apărut ca o nevoie a industriei rețelisticii, care până atunci utiliza alte modalități de administrare, precum Interfața prin Linie de Comandă - CLI (Command Line Interface), Protocolul Simplu pentru Administrarea Rețelelor - SNMP (Simple Network Management Protocol) sau Arhitectura Agentului care Solicită Obiecte Comune - CORBA (Common Object Request Broker Architecture) [75].

Chiar dacă tehnicile folosite până acum pentru administrarea echipamentelor sunt utilizate de mai bine de douăzeci de ani, acestea și-au dovedit limitările. De exemplu, prima abordare, cea folosind interfețe cu linie de comandă, implică o foarte mare dependență de producătorii de echipamente [76], în timp ce protocolul SNMP este folosit îndeosebi pentru colectarea de alarme și de valori ale parametrilor de performanță de la echipamente și mai puțin pentru configurarea acestora. NETCONF încearcă să diminueze aceste limitări propunând o abordare nouă și inovativă. În acest scop a fost dezvoltat și limbajul YANG (Yet Another Next Generation), în RFC 6020 [77], care să modeleze datele folosite de acest protocol.

În următoarele secțiuni se vor detalia modul de funcționare a protocolului NETCONF, punând accent pe aspectele relevante din punctul de vedere al simulatoarelor dezvoltate în această lucrare și limbajul de modelare YANG.

#### 3.3.1 NETCONF

Protocolul NETCONF definește un mecanism simplu de administrare a echipamentelor de rețea, prin care parametrii configurabili pot fi ceruți dispozitivelor, manipulați și apoi retrimiși pentru configurarea echipamentelor. Cu ajutorul acestuia dispozitivele pot expune interfețe diferite de programare care să poată fi utilizate de aplicații software pentru administrare.

Mecanismul propus de NETCONF constă în utilizarea paradigmei Apelurilor de Proceduri la Distanță - RPC (Remote Procedure Call). Un server rulează pe fiecare dispozitiv, așteptând conexiuni de la clienți prin canale securizate și folosind protocoale orientate spre conexiune, precum Terminalul Securizat - SSH (Secure Shell), Securitatea Nivelului Transport - TLS (Transport Layer Security), Protocolul de Interschimbare a Blocurilor Extensibile - BEEP (Blocks Extensible Exchange Protocol), Protocolul de Acces cu Obiecte Simple - SOAP (Simple Object Access Protocol) [78]. Clienții își codează acel apel de procedură la distanță cu ajutorul Limbajului de Marcare Extensibilă - XML (Extensible Markup Language), iar serverul răspunde printr-un mesaj codat cu același limbaj. În timpul stabilirii conexiunii între un client și un server NETCONF, acesta din urmă își expune capabilitățile pe care le are, astfel încât clientul va fi informat asupra acestora, având posibilitatea de a-și schimba comportamentul pentru a profita de caracteristicile serverului.

NETCONF propune folosirea a două tipuri de date: (i) date care descriu starea unui dispozitiv, adică valorile parametrilor operaționali sau valori de contorizare a parametrilor de performanță, toate acestea fiind valori care pot fi doar citite din echipamente și (ii) date configurabile pe dispozitiv, care pot fi atât scrise cât și citite. Pentru operarea acestora se oferă diferite comenzi, cele mai importante fiind:

- *get* - această operație permite unui client să ceară unui dispozitiv atât date de stare, cât și parametri configurabili. Evident, se pot aplica filtre pentru a întoarce doar anumiți parametri dintre cei pe care îi expune echipamentul;
- *get-config* - această operație oferă unui client posibilitatea să ceară unui server doar parametri configurabili, excluzându-i pe cei de stare;
- *edit-config* - aceasta este o operație prin care un client poate schimba valorile unor parametri configurabili pe un dispozitiv.

Pentru stocarea acestor date pe echipament, protocolul NETCONF propune la nivel conceptual un loc în care să se stocheze și prin care să se acceseze informația, denumit *bază de stocare a datelor* - *datastore*. Reprezentarea acestuia este la alegerea fiecărui dispozitiv, putând fi implementat prin fișiere, baze de date, locații de memorie flash, etc. Există trei tipuri de astfel de locuri pentru stocarea datelor:

- *baza de stocare a datelor de inițializare* - acest tip reprezintă o bază de stocare a datelor ce poate conține valori implicite ale parametrilor configurabili ai dispozitivului de rețea și poate fi încărcată în echipament în momentul inițializării acestuia. El este prezent doar în dispozitivele care suportă această separare între configurația curentă, care rulează în echipament și configurația care se încarcă în momentul inițializării;
- *baza de stocare a datelor de operare* - acest tip reprezintă toți parametrii configurabili activi la momentul curent, care sunt prezenți în dispozitiv. Acest tip există întotdeauna pe un echipament;

- *baza de stocare a datelor de rezervă* - acest tip reprezintă o copie a tuturor parametrilor configurabili ai unui dispozitiv. Modificarea acestora nu influențează configurația curentă a dispozitivului, însă aceasta se poate aplica, prin copierea cu ajutorul unei tranzacții, înlocuind baza de stocare a datelor de operare. Nu toate tipurile de echipamente suportă această capacitate.

NETCONF propune și un mecanism prin care să protejeze accesul concurent la scrierea parametrilor configurabili ai dispozitivelor. Astfel, un client poate bloca o parte sau chiar toată baza de stocare a datelor de operare cât timp execută operații prin care schimbă valorile acestor parametri, oferind astfel protecție datelor pe care le modifică. Dispozitivul însă trebuie să se asigure că aceste valori nu pot fi modificate în același timp prin alte căi, cum ar fi SNMP sau CLI.

Acest protocol oferă și posibilitatea serverelor NETCONF să trimită informații către utilizatori cu privire la anumite evenimente care se petrec în dispozitiv. Spre deosebire de celelalte mesaje, aceste notificări sunt inițiate de către server și sunt trimise tuturor utilizatorilor și-au exprimat dorința de a le primi (prin trimiterea către server a unui mesaj de abonare).

### 3.3.2 YANG

Încă o Generație Următoare - *Yet Another Next Generation* - *YANG* este un limbaj de modelare a informației dezvoltat specific pentru protocolul NETCONF. Acesta descrie atât datele de configurație și de stare pe care un dispozitiv le poate expune pentru a fi folosite de către protocol, cât și apelurile de proceduri la distanță sau notificările. A apărut în octombrie 2010 ca RFC 6020, fiind dezvoltat de IETF.

În momentul apariției NETCONF, dat fiind faptul că se baza pe limbajul XML, soluția naturală pentru definirea modelului de date folosit de protocol era utilizarea soluțiilor XML existente pentru modelarea informației, precum schemele XML sau Relax NG [78]. Aceste soluții aveau însă dezavantajele de a fi greu de folosit și de a avea o lizibilitate redusă.

Astfel, un nou limbaj a fost dezvoltat: YANG. Au fost considerate mai multe aspecte în dezvoltarea acestuia: lizibilitatea, o abordare orientată pe obiecte și o oarecare similaritate cu limbajele de programare. YANG îndeplinește aceste condiții, fiind folosit pentru a descrie ierarhii de noduri, care pot reprezenta notificări, apeluri de proceduri la distanță sau parametri de stare sau de configurație și pot fi folosite de operațiile NETCONF. Informațiile sunt stocate în modele YANG și precum în limbajele de programare, un model poate include date din alt model, oferind astfel posibilitatea de a crea seturi de modele de date generice și reutilizabile [79]. Acestea descriu atât nodurile într-un mod concis și clar, cât și interacțiunile dintre ele [80].

YANG descrie informația într-un mod ierarhic, astfel că un fiecare nod are, pe de o parte, un nume și pe de altă parte o valoare sau un set de noduri copil. Se pot descrie și constrângeri ce pot fi aplicate asupra apariției sau valorii unor noduri, bazându-se pe prezența sau valoarea altor noduri ale ierarhiei.

Există mai multe tipuri de noduri definite în limbajul YANG. Cele relevante din punctul de vedere al simulatoarelor implementate în această lucrare sunt:

- *grouping* - acesta este, așa cum o sugerează și numele, un nod care reprezintă o grupare de noduri. După ce este definit el poate fi utilizat în același sau în alte module sau sub-module;
- *list* - acest tip de nod YANG definește o listă de noduri, iar intrările în această listă sunt identificate prin noduri care reprezintă cheia intrării respective;
- *typedef* - este folosit pentru definirea unui tip de date care poate fi utilizat ulterior de alte noduri;
- *rpc* - este folosit pentru modelarea apelurilor de proceduri la distanță, prin definirea numelui procedurii și a parametrilor de intrare și de ieșire;
- *notification* - acest tip de nod se folosește pentru descrierea unei notificări NETCONF pe care un server o poate genera, prin modelarea conținutului acesteia;
- *leaf* - nodurile frunză reprezintă nivelul cel mai jos al ierarhiei și descriu un parametru al dispozitivului, care poate fi de stare (poate fi doar citit) sau de configurare (poate fi și scris și citit).

Așa cum a fost prezentat anterior, ONF dezvoltă modelele informaționale pe care le recomandă cu ajutorul limbajului UML, care este mult mai general și mai puțin specializat decât YANG. Însă, pentru a putea fi folosite într-un mod facil, ONF a dezvoltat și o unealtă software care să transforme modelele din limbajul UML în limbajul YANG, împreună cu o recomandare despre cum această transformare ar trebui făcută [81].

## 3.4 Alegerea unei soluții software pentru serverul NETCONF

Există numeroase soluții care lucrează cu protocolul NETCONF, atât pe partea de client, cât și pe cea de server. O listă a acestora este menținută de către grupul de lucru NETCONF și poate fi găsită online [82]. Conține și soluții software proprietare, dar și soluții cu sursă deschisă. Pentru implementarea simulatoarelor prezentate în această lucrare au fost considerate trei opțiuni de implementare a unui server NETCONF, cu sursă deschisă: *Netopeer*, *OpenYuma* și *NETCONF Test Tool* (unealtă oferită de proiectul ODL). O comparație între acestea va fi prezentată în continuare, justificând astfel alegerea de a folosi una dintre ele în simulatoare [83].

### 3.4.1 Netopeer

*Netopeer* este o soluție ce se bazează pe librăria *libnetconf*, oferind atât o implementare pentru server, cât și una pentru un client al serverului. Această librărie este una cu sursă deschisă, implementată în limbajul C, ce oferă o implementare a protocolului

NETCONF [84]. Este o soluție care poate fi personalizată, oferind numeroase posibilități pentru implementările de server și de client și suportă toate caracteristicile protocolului NETCONF.

*Netopeer* oferă câteva unelte, de exemplu pentru a facilita integrarea modelelor informaționale YANG în module ale serverului, denumită *administrator-netopeer* - *netopeer-manager*, sau pentru a configura caracteristicile serverului NETCONF. Orice model de date YANG poate fi adăugat ca un modul al serverului, însă acesta trebuie prelucrat înainte. Astfel, fișierul *\*.yang* este transformat de către această soluție în fișiere pe care serverul le poate recunoaște, inclusiv un fișier *\*.c*, care conține un schelet de cod C, reprezentând așa-numite funcții cu apel invers (*callback functions*) ce pot fi implementate pentru ca serverul să ofere comportamentul dorit, în raport cu modelul YANG folosit. Apoi, codul C este compilat, rezultând o bibliotecă partajată (*shared library*) care poate fi utilizată de către codul de bază al serverului.

### 3.4.2 OpenYuma

*OpenYuma* este o soluție software care se bazează pe proiectul *Yuma*, care a devenit proprietar în anul 2011. Propune de asemenea implementări pentru server și client NETCONF, scrise în limbajul C, oferind chiar posibilitatea de a încorpora acest cod în dispozitive al căror software folosește tot limbajul C.

*OpenYuma* are o filosofie asemănătoare cu *Netopeer*, oferind unelte pentru transformarea modelelor YANG în cod C schelet, care să fie apoi implementat pentru ca serverul NETCONF să ofere facilitățile propuse. Unealta propusă de această soluție software se numește *yangudmp* și transformă fișierele *\*.yang* în fișiere *\*.h* și *\*.c*, conținând, la fel ca în cazul *Netopeer*, funcții de apel invers ce trebuie rescrise.

Codul C obținut după transformarea modelelor YANG se compilează, rezultând tot o bibliotecă partajată care să poată fi folosită de către codul de cază al serverului. Aceasta poate fi încărcată în momentul inițializării serverului sau chiar în mod dinamic, în timp ce acesta rulează.

### 3.4.3 Unealta de Test Netconf - *Netconf Testtool*

Unealta de Test Netconf este o soluție software oferită în cadrul proiectului OpenDaylight [85]. Este o soluție simplă, care nu poate fi personalizată foarte mult, oferind doar o implementare Java pentru un server NETCONF. Aceasta este folosită de proiectul ODL pentru a-și testa interfața de Sud care implementează protocolul NETCONF.

Scopul acestei soluții este puțin diferit de al celorlalte, deoarece *Testtool* nu își propune oferirea unei soluții software care implementează un server NETCONF care apoi să poată fi integrat cu echipamentele de rețea, ci oferirea unei soluții simple și rapide care să încarce un model YANG specific, cu scopul de a-l testa. Cu toate acestea, acest software a fost considerat pentru comparație, deoarece și scopul simulatoarelor este de a testa modelele YANG și de a crea topologii specifice rețelelor de transport de date fără fir, expunând modelele informaționale descrise anterior.

### 3.4.4 Comparație între soluțiile care oferă server NETCONF

Soluțiile descrise anterior au fost evaluate atât prin compararea documentației relevante pe care acestea o pun la dispoziție, cât și prin experimente practice care iar în considerare diferite scenarii.

Comparația bazată pe lucrurile descrise în documentație este rezumată în Tabelul 3.1, în timp ce un sumar al comparației bazată pe experimentare se găsește în Tabelul 3.2.

Tabelul 3.1: Comparație a caracteristicilor oferite de cadrele software considerate.

<b>Criteriile de comparație</b>	<b>Soluții servere NETCONF</b>		
	<i>Netopeer</i>	<i>OpenYuma</i>	<i>Testtool</i>
Limbaajul de programare	C	C	Java
Încărcarea modelelor YANG brute	Nu	Nu	Da
Încărcarea dinamică a modulelor în server	Da	Da	Nu
Baza de stocare a datelor NETCONF	toate	toate	de operare
Suport pentru notificări	da	da	da
Port configurabil	da	da	da
Mai multe instanțe de server	nu	da	da
Mai multe conexiuni în același timp	da	da	da
Capabilități pentru depanare	jurnalizare	jurnalizare	jurnalizare

Primul criteriu care poate fi considerat este limbaajul pe programare în care aceste soluții software sunt implementate: *Netopeer* și *OpenYuma* sunt scrise în limbaajul C, pe când *Testtool* este o implementare Java.

Un alt criteriu pentru evaluare constă în abilitatea serverului de a încărca în mod direct (dinamic sau în momentul inițializării) modele YANG. Această posibilitate este oferită doar de implementarea Java. Celelalte două soluții au nevoie de o fază premergătoare de procesare, transformând fișierele *\*.yang* în *\*.c*. *Netconf Test Tool* poate încărca modelul YANG doar în momentul inițializării serverului, dintr-un director anume, pe când celelalte cadre software pot încărca acest model, după ce a fost procesat, în mod dinamic.

Un alt subiect pentru comparație este dat de tipurile de baze de stocare de date propuse de protocolul NETCONF suportate de implementările serverelor. *Netopeer* și *OpenYuma* folosesc fișiere XML în care stochează informațiile și suportă toate cele trei tipuri de baze de stocare a datelor propuse de NETCONF: de inițializare, de rezervă și de operare. Cealaltă soluție software, *Testtool* oferă posibilitatea de a utiliza doar baza de stocare de date de operare stocând valorile parametrilor în variabile de execuție, acestea pierzându-se în momentul în care serverul este oprit. O diferență importantă apare în acest context, între cele două soluții implementate



Tabelul 3.2: Compararea practică a cadrelor software considerate

Scenariul experimentat	<i>Netopeer</i>	<i>OpenYuma</i>	<i>Testtool</i>
Procesarea *.yang în *.c	lnctool	yangdump	N/A
Reprezentarea bazei de stocare a datelor în cadrul serverului	fișier XML	fișier XML	variabile de execuție
Încărcarea datelor în faza de inițializare a serverului	<i>startup</i>	flexibilă, orice se poate suprascrie	N/A
Implementarea notificărilor NETCONF	<i>libnetconf</i>	funcție cu apel invers oferită	fișier XML
Funcții pentru parametrii configurabili	câte una per atribut	câte una per atribut	N/A
Funcții pentru parametrii de stare	doar una, globală	câte una per atribut	N/A
Conexiuni concurente de la clienți	da	suportă, trebuie implementat	da

în C, *OpenYuma* oferind o flexibilitate mai mare. În cazul *Netopeer*, atunci când serverul se inițializează, încarcă valorile parametrilor din bazele de stocare de date de inițializare și de operare în memorie. Apoi, începe să analizeze valorile din baza de stocare de date de inițializare, comparându-le cu valorile corespunzătoare atributelor din cea de operare. Dacă valorile nu sunt egale, sau valoarea din baza de stocare de date de operare nu există (însemnând prima utilizare a serverului), atunci serverul copiază valoarea din baza de stocare de date de inițializare și apelează funcția de apel invers asociată parametrului de configurare. Prin această abordare serverul se asigură ca nu există inconsistențe între bazele de stocare de date de inițializare și de operare și, mai mult, dacă acest server este conectat la un echipament real de rețea, prin apelarea funcției cu apel invers asociate parametrului, dispozitivul va fi configurat astfel încât valorile din server să reflecte valorile de pe echipament. *OpenYuma* are o abordare mai flexibilă, permițând dezvoltatorilor să altereze baza de stocare de date de operare în timpul inițializării modulului, fără a implica baza de stocare de date de inițializare.

O altă caracteristică importantă a serverelor care poate fi comparată este abilitatea de a genera notificări NETCONF. Toate cele trei soluții oferă notificări. În cazul *Testtool* este o implementare simplă, acestea fiind declanșate printr-o comandă NETCONF și trimise dintr-un fișier XML care le conține. *OpenYuma* oferă câte o funcție cu apel invers pentru fiecare notificare pe care o găsește în momentul procesării modelului YANG. *Netopeer* nu are această abilitate în mod implicit, însă suportul este oferit de *libnetconf*.

Din perspectiva funcțiilor cu apel invers generate în momentul procesării modelului YANG putem compara doar cele două soluții implementate în limbajul C, deoarece *Testtool* nu oferă astfel de caracteristici. Soluția *Netopeer* oferă posibilitatea de a alege care dintre parametrii configurabili vor avea o funcție cu apel invers asociată, în timp ce *OpenYuma* va genera o astfel de funcție pentru fiecare din parametrii configurabili pe care îi găsește în modelul YANG. Pentru informațiile de stare *Netopeer* generează o singură funcție cu apel invers care este apelată pentru orice operație NETCONF de tip *get* care ajunge la server, iar *OpenYuma* generează câte o astfel de funcție pentru fiecare parametru de stare.

Din punctul de vedere al abilității de a configura portul pe care serverul acceptă conexiuni, toate soluțiile oferă această posibilitate. *Testtool* are nevoie de un port de plecare, incrementând apoi numărul portului pentru celelalte instanțe ale serverului. Și celelalte soluții oferă posibilitatea de a schimba portul pe care serverul ascultă. Din perspectiva rulării mai multor instanțe de server pe aceeași mașină, atât *Testtool* cât și *OpenYuma* oferă această posibilitate, diferitele instanțe folosind porturi diferite. *Netopeer*, pe de altă parte, nu permite rularea mai multor instanțe pe aceeași mașină.

Un alt criteriu pentru compararea acestor cadre software este dat de suportul pentru mai multe fire de execuție, adică abilitatea serverului de a permite conectarea mai multor clienți în același timp. *Testtool* oferă această posibilitate. *OpenYuma* oferă de asemenea acest suport, cu mențiunea că accesul concurent la resursele comune trebuie rezolvat în funcțiile cu apel invers care vor fi implementate de către dezvoltatori, nefiind rezolvat de către cadrul software. În cazul *Netopeer*, acest acces concurent este rezolvat din faza de proiectare a serverului și este oferit de *libnetconf*.

Și capabilitățile de depanare oferite ar putea constitui un criteriu de comparație, însă toate soluțiile se bazează pe fișiere de tip jurnal, oferind mai multe niveluri de jurnalizare.

Pe baza acestei comparații a fost aleasă soluția *OpenYuma* pentru implementarea serverului NETCONF folosit în simulatoare, ea oferind cea mai mare flexibilitate.

## 3.5 Arhitectura demonstrațiilor de concept WT SDN

Așa cum a fost amintit anterior, proiectul Rețelelor de Transport de Date Fără Fir (*Wireless Transport* - WT) din Grupul de Lucru pentru Transport Deschis (*Open Transport Working Grup*) al ONF se ocupă cu cercetare legată de SDN în contextul rețelelor de transport de date fără fir. După cum este prezentat și în [86, 87], acest tip de rețele sunt o parte importantă atât din perspectiva rețelelor curente, cât și din cea a rețelelor viitorului, precum cele 5G.

După efectuarea unei demonstrații de concept [46] care folosea protocolul OpenFlow pentru a arăta capabilitățile SDN în rețelele de transport de date fără fir, observând dificultățile pe care le presupune lucrul cu OpenFlow în acest context, grupul a trecut la dezvoltarea unui model informațional care să abstractizeze dispozitivele ce fac parte din aceste rețele, model care a fost descris anterior.

Apoi, alte trei demonstrații de concept au avut loc, descrise în [47–49], care au urmărit evoluția acestui model informațional. Chiar dacă în cea de-a doua demonstrație de concept s-a folosit un model de date mult simplificat, iar în cel de-al patrulea a fost folosită cea mai bună și matură variantă a sa, arhitectura acestora a fost similară.

După cum se poate observa în Figura 3.6, în arhitectura acestor demonstrații apare nevoia unui nivel intermediar, de mediator.

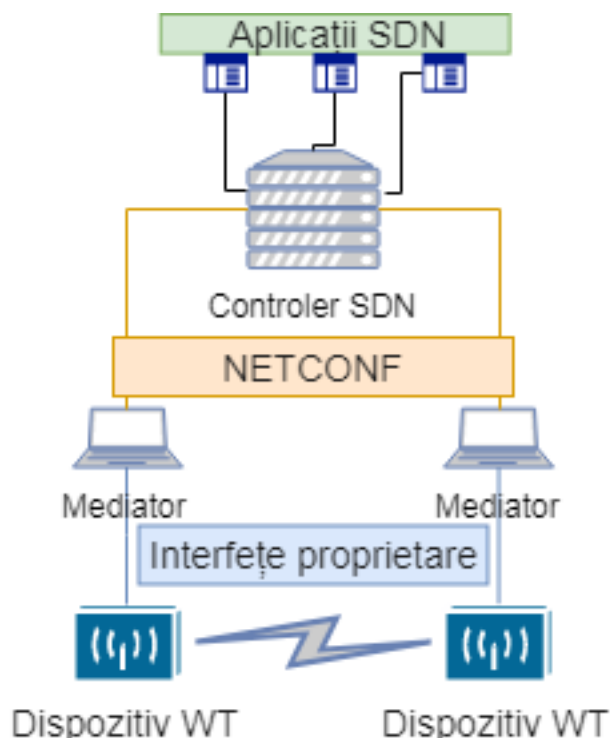


Fig. 3.6: Arhitectura demonstrațiilor de concept WT [46–48].

Acest mediator este reprezentat de o aplicație software care expune o interfață NETCONF către nord, unde se conectează echipamentul de control SDN, folosind modelul informațional descris de TR-532. Rolul mediatorului este de a traduce comenzile NETCONF care vin de la controler în comenzi pe care dispozitivul de rețea le poate înțelege (de exemplu SNMP, CLI sau Transferul Stărilor Reprezentaționale - REST (Representational State Transfer)). Această nevoie apare din cauza faptului că echipamentele din rețelele actuale, care au fost folosite și pentru demonstrațiile de concept, nu suportă încă noul model informațional în mod nativ, deoarece abia a fost dezvoltat. Cel mai probabil, în viitor această nevoie va dispărea, echipamentele putând să încadreze TR-532 în aplicația software proprie de control.

Poziționarea simulatoarelor în arhitectura acestor demonstrații de concept este ilustrată în Figura 3.7. Astfel, ele pot fi folosite de către dezvoltatorii de aplicații SDN pentru testarea aplicațiilor care utilizează interfața NETCONF ce implementează modelul informațional TR-532, eliminând nevoia acestora de a deține dispozitive pentru transportul de date fără fir și mediatorul asociat acestora. Ultima versiune de

simulator dezvoltată, WTE, oferă chiar facilitatea de a simula o întreagă topologie de transport de date fără fir, incluzând legăturile dintre dispozitivele de rețea. Un astfel de scenariu este prezentat în Figura 3.8.

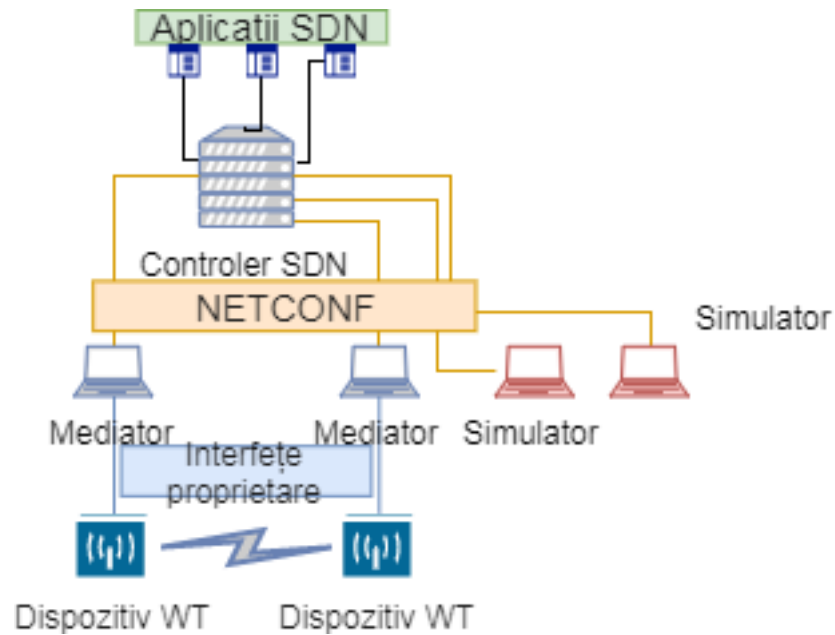


Fig. 3.7: Poziționarea simulatoarelor în arhitectura ONF [46–48].

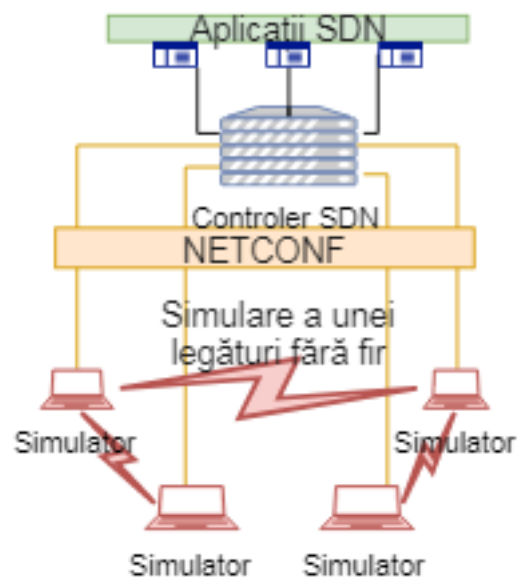


Fig. 3.8: Topologie simulată a unei rețele de transport de date bazată pe WTE.



## Capitolul 4

# Mediatorul cu valori implicite (DVM)

Nivelul mediator a apărut în arhitectura demonstrațiilor de concept, cum a fost prezentat anterior, deoarece dispozitivele de rețea încă nu au suport pentru modelul informațional pentru microunde TR-532. Astfel, această aplicație software trebuie să traducă operațiile NETCONF care vin de la echipamentul de control într-un limbaj care să fie recunoscut de echipamente. Fiecare producător de dispozitive care a participat la cea de-a doua demonstrație de concept WT SDN a fost nevoit să implementeze un astfel de mediator.

Am dezvoltat mediatorul cu valori implicite (*Default Values Mediator - DVM*) special pentru cea de-a doua demonstrație de concept WT a ONF, pentru a oferi posibilitatea dezvoltatorilor de aplicații SDN care nu au acces la echipament de transport de date fără fir și la mediatoarele asociate acestora posibilitatea de a implementa și testa astfel de aplicații, care vor putea apoi interacționa cu mediatoare reale, pentru că folosesc aceeași interfață NETCONF care expune modelul informațional de bază și modelul informațional pentru microunde. Apoi, această implementare a evoluat în cea de-a doua versiune a DVM, care a fost folosită pentru cea de-a doua demonstrație de concept.

DVM este o implementare software cu sursă deschisă a unui server NETCONF, bazându-se pe soluția software *OpenYuma*. Spre deosebire de mediatoarele reale, *DVM* are o singură interfață (cea prin care comunică cu controlerul SDN), neavând nevoie de o interfață care să se lege la dispozitive reale de rețea. Acest simulator va returna valori implicite pentru attributele interogate de către controler. Atât prima, cât și cea de-a doua versiune a DVM sunt implementări în limbajul de programare C și aplicațiile obținute după compilarea codului pot fi rulate sub sistemul de operare Linux.

## 4.1 Arhitectura DVM versiunea 1

Prima versiune a mediatorului cu valori implicite expune către echipamentul de control SDN modelul informațional simplificat pentru microunde, care a fost folosit în cea de-a doua demonstrație de concept efectuată de proiectul WT din ONF [88].

Implementarea acestuia este una simplă și modulară, bazată pe soluția *Open Yuma*, oferind câte o funcție cu apel invers pentru fiecare atribut din modelele YANG expuse, indiferent de natura lor (parametri de configurare sau de stare). În aceste funcții se stabilește valoare atributului respectiv, care este returnată unui client NETCONF. Excepția de la această regulă este dată de câteva atribute ale căror valori sunt definite într-un fișier de configurare folosit de către mediator. Dezavantajul acestei abordări este că dacă un dezvoltator de aplicații are nevoie de o altă valoare a unui atribut care nu face parte din acest fișier de configurare, va trebui să o modifice în funcția asociată parametrului respectiv și apoi să recompileze codul asociat modului de server care conține acel parametru și apoi să încarce din nou acel modul în server.

O imagine de ansamblu a arhitecturii primii versiuni a DVM este ilustrată în Figura 4.1. Acesta se bazează pe un fișier de configurare ce conține câțiva parametri importanți din punctul de vedere al aplicațiilor SDN. Aceștia sunt: (i) numele echipamentului de rețea - *Network Element Name* și (ii) un identificator unic folosit pentru fiecare legătură radio - *Radio Signal ID*.

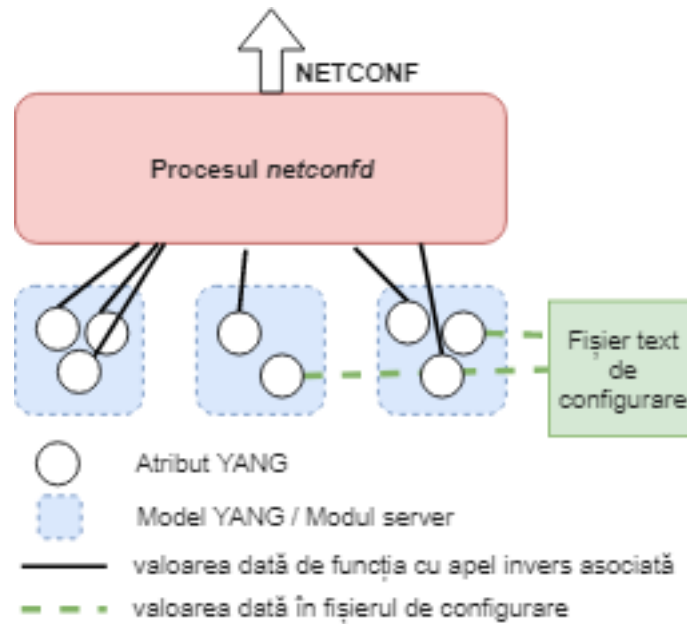


Fig. 4.1: Arhitectura primii versiuni a DVM [88].

Acești parametri sunt importanți pentru aplicații deoarece cu ajutorul lor se pot identifica în mod unic dispozitivele din rețea. Prin pregătirea mai multor fișiere de configurare se pot simula mai multe elemente de rețea, fără a fi nevoie de recompilarea simulatorului. Astfel, vor rula mai multe instanțe ale DVM, fiecare având propriul

fișier de configurare ce conține un alt nume pentru dispozitiv și alte identificatoare pentru legăturile radio.

În această arhitectură, DVM este configurat să trimită și notificări NETCONF fictive către utilizatorii care s-au abonat la primirea acestora. Fișierul de configurare conține astfel și o valoare numerică reprezentând numărul de secunde dintre două notificări fictive consecutive. Dacă această valoare este mai mare decât zero, se vor trimite notificări fictive la intervalul de timp specificat. Altfel, dacă valoarea este zero, DVM nu va trimite astfel de mesaje NETCONF.

## 4.2 Implementarea DVM versiunea 1

Primul pas al implementării a fost reprezentat de alegerea modelelor YANG care să fie expuse de serverul NETCONF. În cea de-a doua demonstrație de concept WT a fost agreată folosirea unor modele informaționale reduse, care să permită demonstrarea cazurilor de utilizare alese [47]. Acestea conțineau aproximativ șaiszeci de atribute care făceau parte atât din modelul informațional de bază, cât și din modelul informațional pentru microunde. Astfel, au fost alese trei modele YANG pentru implementarea în cadrul primei versiuni a DVM: *CoreModel-CoreNetworkModule-ObjectClasses*, *MicrowaveModel-ObjectClasses-MwConnection* și *MicrowaveModel-Notifications*. Asta a însemnat, practic, generarea a trei biblioteci partajate reprezentând module ale serverului NETCONF, care să poată fi încărcate în soluția *OpenYuma* și să ofere capabilitățile dorite.

O organigramă a fazei de dezvoltare și implementare a DVM este ilustrată în Figura 4.2.

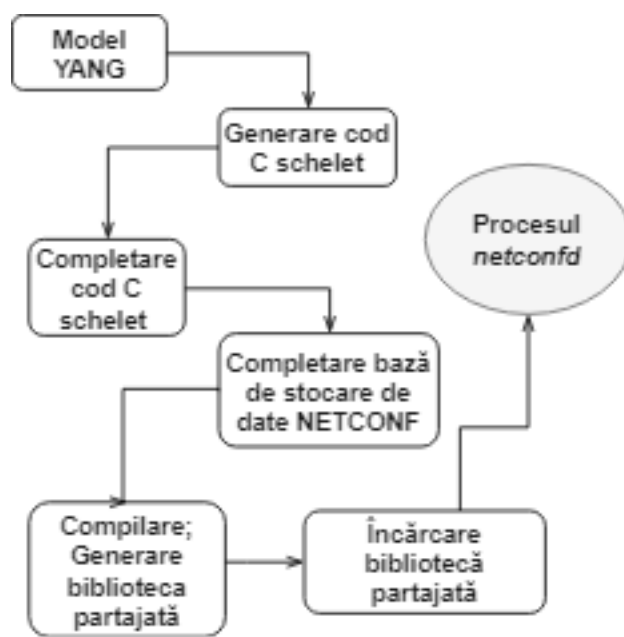


Fig. 4.2: Organigramă a dezvoltării și implementării DVM [88].



Cel de-al doilea pas al implementării a constat în procesarea modelelor YANG alese și generarea codului C schelet al modulelor asociate acestora. Acest lucru a fost realizat cu utilitarul *yangdump* oferit de soluția *OpenYuma*. Pentru a îmbunătăți flexibilitatea DVM și pentru a avea o mai bună separare între codul folosit de server și codul utilizatorului, care trebuie rescris, utilitarul a fost folosit cu opțiunea *-split*. Astfel, pentru fiecare modul au fost generate patru fișiere: câte unul *.c* și *.h* pentru codul de server, respectiv pentru cel de utilizator.

Următorul pas a fost reprezentat de implementarea funcțiilor cu apel invers generate pentru fiecare atribut al modelului YANG. Pentru a asocia câte o valoare implicită fiecărui parametru, funcțiile au fost modificate astfel încât să întoarcă valoarea respectivă în momentul apelării.

Cea mai importantă parte a implementării DVM a constat în construirea bazei de stocare a datelor de operare, reprezentând de fapt arborele atributelor YANG pe care serverul NETCONF le va utiliza atunci când va fi interogat de către echipamentul de control SDN. Nu a fost posibilă construirea automată a acestui arbore de parametri, astfel că attributele au fost implementate manual, de la rădăcină către frunze. Pentru acest lucru a fost nevoie să se altereze funcția de inițializare *init2()* generată automat pentru fiecare model YANG. A fost necesară adăugarea câte unui nod *OpenYuma* pentru fiecare parametru YANG, având asociată o funcție cu apel invers în care se setează valoarea implicită a acelui atribut. În cazul parametrilor anterior menționați, care fac parte din fișierul de configurare, implementarea funcției constă în citirea valorii respective din acel fișier. Această abordare a permis utilizatorilor DVM un acces facil la valorile implicite asociate fiecărui atribut. Dacă un utilizator avea nevoie de schimbarea valorii unui parametru YANG, o putea face ușor prin funcția cu apel invers asociată sau prin fișierul de configurare, fără să fie nevoit să știe detaliile de implementare referitoare la construirea arborelui de valori.

Următorii pași ai implementării sunt simpli și direcți: codul rezultat se compilează, rezultând bibliotecile partajate care apoi sunt încărcate în serverul NETCONF (mai exact în procesul *netconfd* asociat acestuia).

Etapele anterior menționate se aplică în cazul modelului informațional de bază și în cazul modelului informațional pentru microunde, excluzând cazul notificărilor NETCONF, unde abordarea este puțin diferită.

Deoarece modelul asociat notificărilor NETCONF, *MicrowaveModel-Notifications*, are o structură diferită, conținând obiecte YANG ce reprezintă notificări în locul atributelor obișnuite, comportamentul soluției *OpenYuma* este diferit în acest caz. În loc să se genereze funcții cu apel invers pentru obținerea și setarea valorilor atributelor, în cazul notificărilor soluția *OpenYuma* va genera funcții cu apel invers folosite pentru declanșarea acestora (trimiterea lor de către server tuturor utilizatorilor care s-au abonat).

Pentru implementarea notificărilor NETCONF în DVM, un nou fir de execuție s-a creat în funcția de inițializare a modulului, *init2()*. Acesta rulează o singură funcție care implementează o buclă infinită în care se folosește funcția cu apel invers asociată pentru a trimite o notificare fictivă, la un interval de secunde definit în fișierul de configurare. În cazul în care valoarea intervalului este zero, declanșarea notificărilor nu va fi activată. Detaliile conținute în notificarea NETCONF fictivă se

găsesc în interiorul funcției care implementează generarea și modificarea acestora nu este banală pentru un utilizator neexperimentat.

Structura fișierului de configurare este foarte simplă și nu oferă prea multă flexibilitate utilizatorilor DVM. Aceasta este fixă și poate fi observată în Figura 4.3, într-un exemplu în care un dispozitiv conține două interfețe radio. Conține doar trei tipuri de parametri, așa cum a fost menționat anterior: numele echipamentului de rețea (*NeName*), identificatoarele legăturilor radio (*radioSignalId* - există câte un identificator pentru fiecare interfață radio a dispozitivului) și intervalul de timp, exprimat în secunde, dintre două notificări fictive consecutive (*eventFrequency*).

mediatorConfig.txt	
<b>radioSignalId:</b>	<nume obiect LP 1> valoare
<b>radioSignalId:</b>	<nume obiect LP 2> valoare
<b>NeName:</b>	nume echipament de rețea
<b>eventFrequency:</b>	<timpul în secunde dintre două notificări fictive consecutive>

Fig. 4.3: Structura fișierului de configurare al DVM. Exemplu pentru un echipament cu două interfețe radio.

### 4.3 Folosirea în contextul demonstrațiilor de concept a DVM versiunea 1

Prima versiune a mediatorului cu valori implicite a fost o unealtă foarte importantă în contextul celei de-a doua demonstrații de concept a proiectului rețelelor de transport de date fără fir din cadrul ONF. Acesta a ajutat la accelerarea implementării aplicațiilor SDN care au fost dezvoltate pentru cazurile de utilizare propuse în [47].

DVM a oferit interfața de Sud NETCONF care expune modelele informaționale dezvoltate de ONF, în același mod în care ar expune-o un mediator real ce se conectează la echipamente de transport de date fără fir. În acest mod, dezvoltatorii aplicațiilor SDN au putut utiliza acest simulator pentru implementarea și testarea acestora, fără a avea nevoie să dețină dispozitive de rețea, care au un preț foarte ridicat.

Această primă versiune de simulator a accelerat activitățile de pregătire a celei de-a doua demonstrații de concept, permițând lucrul în paralel la aplicațiile SDN și la dezvoltarea mediatoarelor. Interfața NETCONF comună a putut fi testată înainte ca producătorii de echipamente să își implementeze mediatoarele, oferind astfel mai mult timp dezvoltatorilor aplicațiilor SDN pentru depanarea programelor. De exemplu, generarea unei notificări NETCONF se poate face mult mai facil cu ajutorul simulatorului. Pentru un mediator real, trebuie ca dispozitivul să fie făcut să genereze

o notificare către mediator, prin interfața proprietară echipamentului respectiv, apoi mediatorul să traducă acel mesaj într-o notificare NETCONF.

Figura 4.4 reprezintă elementele de bază ale demonstrației de concept, așa cum sunt prezentate în lucrarea apărută după desfășurarea acestuia [47], în care se poate vedea DVM.

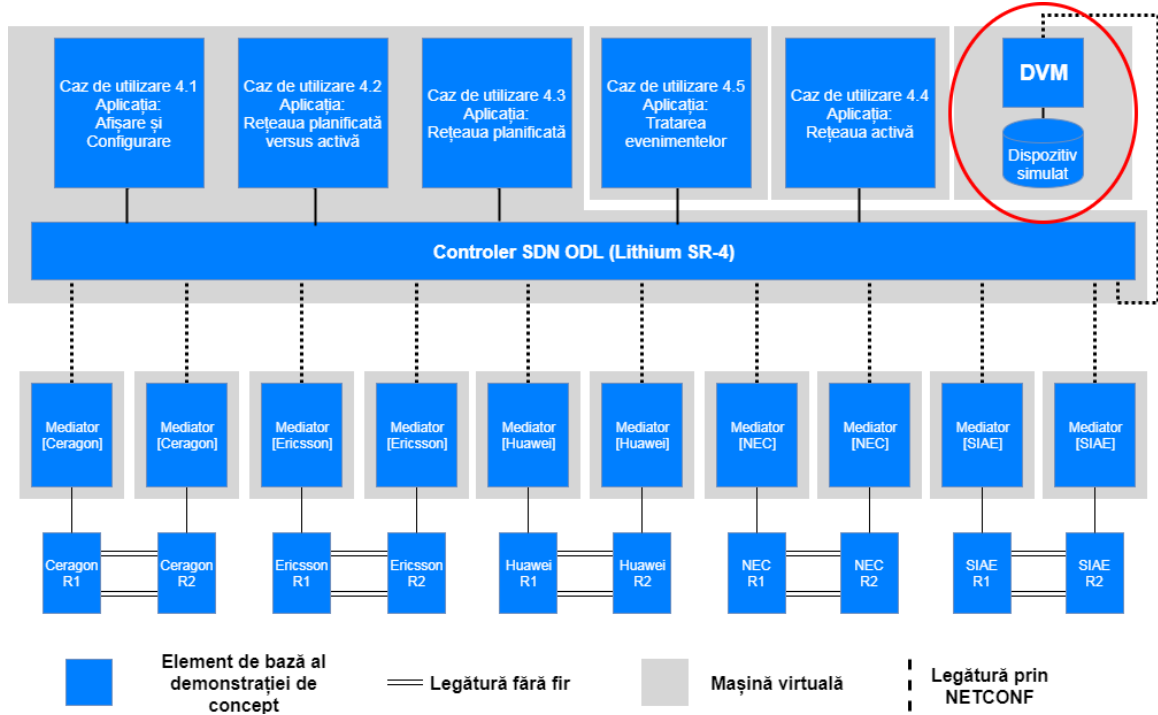


Fig. 4.4: Configurarea rețelei de test SDN utilizând mașini virtuale [47].

Înregistrarea la controlerul SDN a primei versiuni a simulatoarelor DVM, se face la fel ca pentru un mediator real. Astfel, echipamentul de control oferă o interfață de programare a aplicației - API (Application Programming Interface) - prin care o aplicație SDN poate înregistra un astfel de mediator în controlerul SDN. Înregistrarea nu este una automată, utilizatorul fiind nevoit să facă această înregistrare manual. În cea de-a doua demonstrație de concept, acest lucru a fost făcut prin interfața grafică a echipamentului de control folosit (ODL). După înregistrare, controlerul stabilește conexiunea NETCONF cu mediatorul.

Codul asociat primei versiuni a DVM este oferit cu sursă deschisă și se poate găsi în repertoriul asociat ONF de pe platforma GitHub, denumit CENTENNIAL [89].

## 4.4 Arhitectura DVM versiunea 2

Cea de-a doua versiune a DVM a fost implementată pentru a treia demonstrație de concept a rețelelor de transport de date fără fir desfășurată în cadrul ONF. A fost dezvoltată pe scheletul primei versiuni, păstrând aceeași abordare: un server NETCONF care se bazează pe soluția software *OpenYuma* și un fișier de configurare

din care serverul poate citi valori pentru atributele YANG pe care le expune. Din acest punct de vedere, DVM versiunea 2 folosește întregul model informațional pentru microunde TR-532 și o parte semnificativă din modelul informațional de bază, TR-512.1, cumulând aproximativ 300 de astfel de parametri.

Arhitectura celei de-a doua versiuni a DVM este influențată de natura atributelor YANG ce alcătuiesc modelul informațional pentru microunde [90]: parametri de configurare, de stare sau care prezintă capacitățile dispozitivului. Atributele de configurare pot fi citite sau scrise și prin intermediul acestora un utilizator poate influența comportamentul dispozitivului. Parametrii de stare pot fi doar citiți și reprezintă situația curentă a echipamentului, în timp ce capacitățile sunt atribute care pot fi doar citite și descriu abilitățile dispozitivului.

Mediatorul cu valori implicite a fost gândit astfel încât să fie flexibil și să ofere posibilitatea înlocuirii citirii valorilor din fișierul de configurare cu o citire dintr-un echipament real, cu ajutorul unui protocol la alegere. Deoarece mediatorul este o aplicație software externă, care nu face parte din echipamentele de transport de date fără fir, în momentul inițializării ar trebui să reflecte configurația echipamentului la care se conectează. Din cauza faptului că dispozitivul nu are o configurație statică și aceasta poate fi modificată înaintea inițializării mediatorului, nu poate fi utilizată baza de stocare de date de inițializare oferită de serverul NETCONF. Asta înseamnă ca în momentul inițializării mediatorul trebuie să își construiască baza de stocare de date de operare într-un mod arborescent, interogând dispozitivul de rețea asupra valorilor atributelor. Acest lucru se aplică doar în cazul parametrilor configurabili sau în cazul celor care au un caracter static, precum capacitățile echipamentului. Deoarece aceste valori se salvează în baza de stocare de date de operare, înseamnă că acestea nu sunt persistente și vor fi citite la fiecare inițializare din dispozitiv. Fiindcă simulatorul DVM a fost proiectat ca un mediator real, toate aceste aspecte se reflectă și în arhitectura acestuia, diferența fiind că valorile atributelor nu se citesc dintr-un dispozitiv, ci dintr-un fișier de configurare descris în limbajul XML.

Natura atributelor YANG, descrisă anterior, a dus la arhitectura celei de-a doua versiuni a DVM care se poate vedea în Figura 4.5. Atributele YANG au fost împărțite în trei categorii: de inițializare, de execuție și de configurare.

Parametrii de inițializare reprezintă acele atribute statice, care sunt citite din dispozitiv o singură dată, în momentul inițializării simulatorului. În cadrul serverului NETCONF pot fi doar citite, în cazul parametrilor care reprezintă capacitățile echipamentului (adică informație statică, prezentată de către dispozitiv, care nu se poate schimba) sau pot fi și scrise și citite, ca în cazul parametrilor de configurare. În momentul inițializării DVM aceștia sunt considerați parametri de inițializare, ca apoi să devină de configurare, având câte o funcție cu apel invers asociată, care permite modificarea acestei valori în fișierul XML de configurare prin intermediul serverului NETCONF. Nu este nevoie ca aceste atribute de configurare să fie citite din dispozitiv decât în momentul inițializării, deoarece apoi se presupune că orice configurare asupra dispozitivului se va face prin echipamentul de control SDN, deci prin intermediul serverului NETCONF care va fi astfel informat asupra noii valori a parametrilor respectivi.

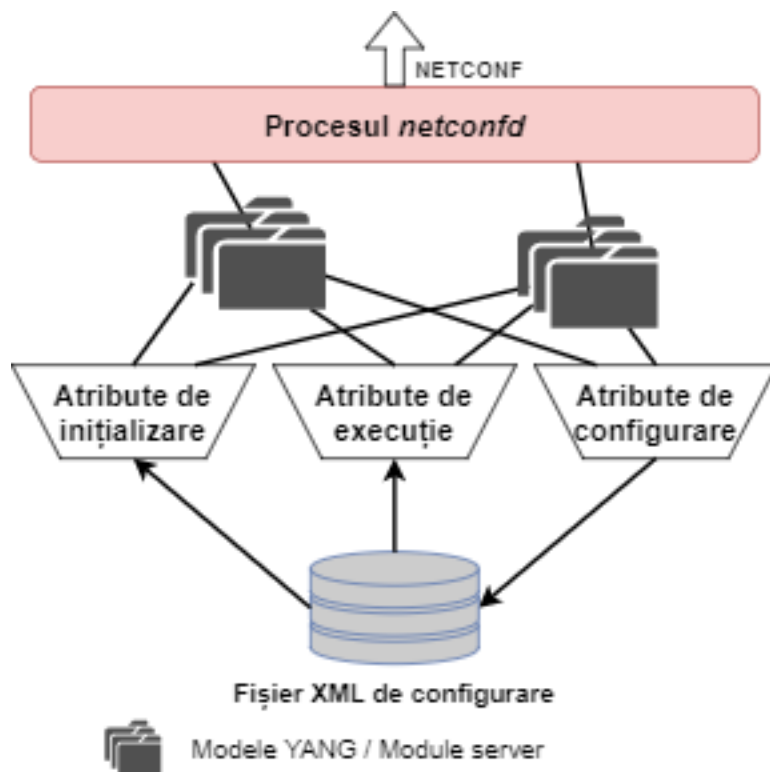


Fig. 4.5: Arhitectura simulatorului DVM versiunea 2.

Atributele de execuție reprezintă parametri dinamici, care pot fi doar citiți ai dispozitivului, precum alarme, informații de stare sau de monitorizare a performanțelor. Acestea trebuie citite din echipament de fiecare dată când controlerul SDN le cere, din cauza naturii lor dinamice. Soluția acestei abordări constă în nodurile virtuale oferite de soluția *OpenYuma*. În loc de a avea o valoare stocată pentru un atribut YANG, sau valori pentru grupuri de atribute, *OpenYuma* oferă posibilitatea de a asocia o funcție cu apel invers pentru astfel de parametri. Aceasta va fi apelată de fiecare dată când valoarea atributului asociat este cerută serverului NETCONF, iar implementarea acesteia va prelua valoarea din fișierul XML de configurare, sau va construi arborele asociat răspunsului, în cazul unui grup de atribute.

Cea de-a doua versiune a simulatorului DVM se bazează pe abordarea *OpenYuma* de a folosi funcții cu apel invers pentru implementarea funcționalității de citire sau scriere asociată atributelor modelelor YANG. Proiectarea DVM urmărește separarea descrisă anterior a parametrilor, oferind diferite funcții cu apel invers pentru diferitele categorii. Fiecare atribut din modelele YANG va fi modelat ca un nod *OpenYuma*, care reprezintă un tip de date pus la dispoziție de această soluție software, conținând diverse informații rezultate din analiza modelului YANG, precum numele, constrângeri asupra valorilor pe care atributul le poate lua și funcția cu apel invers asociată, de citire sau de scriere. Astfel, au fost considerate trei funcții cu apel invers generice, care să fie folosite în cele trei cazuri date de împărțirea atributelor YANG: pentru parametri de inițializare, de execuție și de configurare. Acestea sunt

considerate generice, deoarece o singură astfel de funcție cu apel invers este utilizată pentru fiecare tip de atribut, diferențierea pentru fiecare atribut în parte făcându-se în implementarea acesteia, în funcție de numele atributului. Această abordare oferă flexibilitate și ușurință în implementarea simulatorului DVM.

## 4.5 Implementarea DVM versiunea 2

Implementarea simulatorului DVM versiunea 2 folosește memoria internă și un fișier aflat în sistemul local de fișiere pentru stocarea și întoarcerea valorilor atributelor modelelor YANG pe care le expune. Fișierul de configurare conține valori pentru toți acești parametri și are o structură asemănătoare unui răspuns la operația *get* al unui server NETCONF. Această abordare este foarte avantajoasă, din două motive. În primul rând, dacă un dezvoltator de aplicații software SDN are nevoie de alte valori pentru orice atribut YANG, este suficient să modifice acea valoare în fișier și să repornească simulatorul DVM, fără să fie nevoie să cunoască detaliile de implementare sau să recompileze serverul NETCONF. În cel de-al doilea rând, DVM poate oferi răspunsuri similare cu un mediator real dacă în fișierul de configurare se va introduce răspunsul XML venit de la un astfel de mediator la o operație *get*.

Fluxul de lucru pentru operațiile de aducere a atributelor este ilustrat în Figura 4.6.

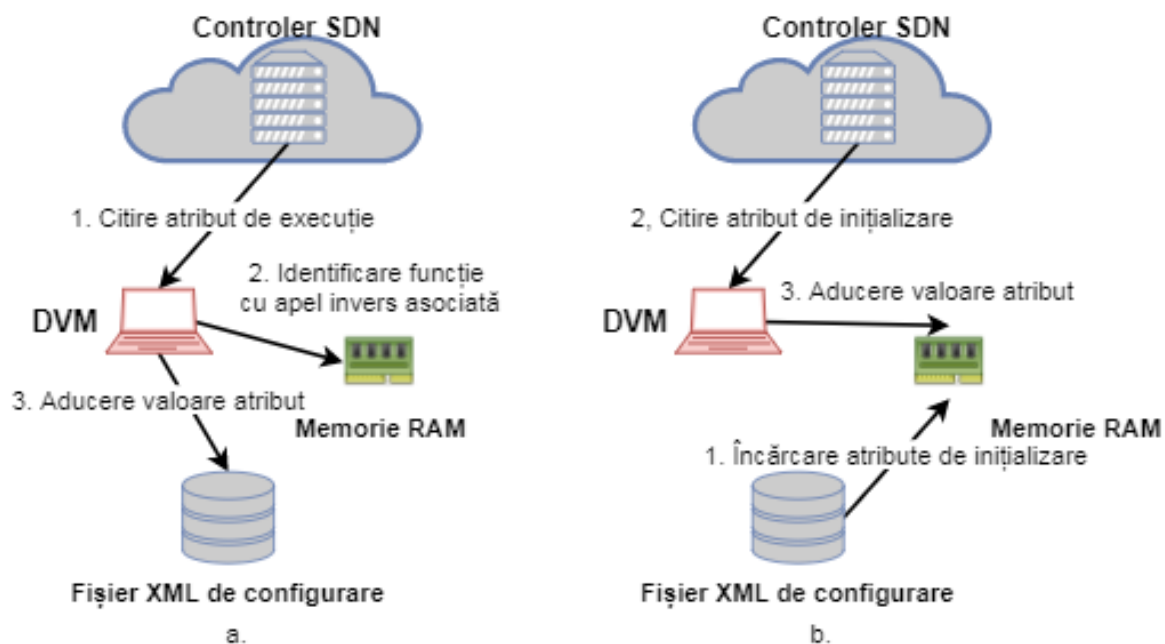


Fig. 4.6: Fluxul de lucru pentru aducerea atributelor: a) de execuție; b) de inițializare [90].

Echipamentul de control SDN care se conectează la simulatorul DVM poate efectua diferite operații asupra parametrilor de execuție sau de inițializare. Acesta asigură funcționalitatea necesară prin intermediul funcțiilor cu apel invers generice

care se află în implementările nodurilor *OpenYuma* pentru fiecare atribut, care își vor da seama pentru ce parametru a fost invocată funcția prin numele atributului. De exemplu, dacă echipamentul de control SDN folosește o operație NETCONF *get* pentru a afla valoarea atributului de stare *txFrequency* (frecvența emițătorului), funcția cu apel invers generică, asociată parametrilor de execuție va fi invocată, își va da seama din numele parametrului că se dorește valoarea pentru atributul respectiv și va întoarce valoarea corespunzătoare din fișierul XML de configurare. Același lucru se întâmplă și pentru atributele de inițializare, cu diferența că valoarea acestora va fi deja încărcată în memoria serverului NETCONF din momentul inițializării simulatorului. Continuând exemplul anterior, dacă echipamentul de control are nevoie de valoarea parametrului *txFrequencyMin* (valoarea minimă pe care frecvența emițătorului o poate avea), o va întoarce din memoria aplicației, deoarece în momentul inițializării DVM aceasta a fost încărcată acolo de funcția cu apel invers generică asociată atributelor de inițializare.

Un caz special este reprezentat de atributele care sunt membre în liste. În această categorie se pot încadra, de exemplu, alarmele de pe dispozitiv, sau valorile indicatorilor de performanță. Fiind parte dintr-o listă, există o valoare cheie care diferențiază intrările din listă. Aceasta este necesară în momentul în care se dorește aflarea valorii unui astfel de atribut. Din această cauză, în momentul invocării funcțiilor cu apel invers generice pentru un parametru dintr-o listă, aceasta va trebui întâi să valideze cheia ca fiind parte din intrările din lista respectivă și abia apoi să apeleze funcția asociată nodului respectiv.

Modelele YANG implementate în cea de-a treia demonstrație de concept ONF, care au fost expuse și de simulatorul DVM sunt parte din TR-532 și TR-512.1: *MicrowaveModel-ObjectClasses-AirInterface*, *MicrowaveModel-ObjectClasses-PureEthernetStructure*, *MicrowaveModel-ObjectClasses-EthernetContainer*, *CoreModel-CoreNetworkModule-ObjectClasses* și *MicrowaveModel-Notifications*. Fiecare dintre acestea a fost transformat într-un modul al serverului NETCONF, mai exact într-o bibliotecă partajată care să poată fi încărcată de codul de bază al serverului, existând o relație de 1:1 între un model YANG și un modul al serverului. Codul în limbajul C al simulatorului DVM folosit în cea de-a treia demonstrație de concept este de asemenea cu sursă deschisă și poate fi găsit în repertoriul ONF de pe platforma GitHub [91].

În implementare a fost aplicat același principiu ca în prima versiune a DVM: în funcțiile de inițializare ale fiecărui modul a fost construită baza de stocare de date de execuție arborescentă, folosind valorile cuprinse în fișierul XML de configurare. Pentru implementarea diferitelor tipuri de parametri YANG (de inițializare, de execuție și de configurare) în DVM a fost construită o bibliotecă statică, tot în limbajul C: ***Base mediator utils***. Printre fișierele mai importante pe care aceasta le cuprinde se numără, după cum este descris și în [90]:

- *boot\_time\_callbacks.c* - acest fișier expune o funcție cu apel invers generică, care este folosită în modulele serverului pentru a găsi valoarea atributelor YANG de inițializare în fișierul de configurare XML. Mai conține de asemenea și funcții folosite pentru găsirea cheilor în cazul elementelor de tip listă care sunt prezente

în model. Este nevoie de acestea în etapa de inițializare, pentru a ști câte elemente sunt conținute în listele respective. Intern (adică fără a fi expuse), acest fișier conține câte o funcție asociată fiecărui parametru de inițializare, a cărei implementare găsește valoarea în fișierul XML pentru atributul respectiv, însă nu direct, ci prin apelarea altor funcții, din rațiuni de lizibilitatea a codului;

- *runtime\_callbacks.c* - acest fișier este asemănător cu cel anterior, diferența fiind făcută de attributele YANG pe care le referă. Astfel, în acest caz este vorba de attributele dinamice, de execuție;
- *configuration\_callbacks.c* - acest fișier sursă expune o funcție generică folosită pentru a seta valoarea unui atribut de configurare în fișierul XML de configurare. Aceasta este invocată în momentul folosirii unei operații de tip *edit-config* de către un client NETCONF (de exemplu echipamentul de control SDN);
- *dvm\_boot\_time\_callbacks.c* - acest fișier a fost creat pentru îmbunătățirea lizibilității codului și conține implementările funcțiilor care citesc valoarea atributelor de inițializare din fișierul XML;
- *dvm\_runtime\_callbacks.c* - acest fișier are același rol cu cel anterior, cu diferența că referă attributele de execuție;
- *utils.c* - acest fișier conține funcții utilitare de care are nevoie biblioteca, cum ar fi funcții pentru citirea din fișierul de configurare sau funcții pentru crearea de noduri *OpenYuma* în baza de stocare de date de execuție.

Fiecare dintre bibliotecile partajate care corespund modelelor YANG sunt legate static de biblioteca ***Base\_mediator\_utils***. În acest mod se asigură o separație între logica de construire a bazei de stocare de date de execuție a serverului NETCONF și găsirea valorii atributelor YANG în fișierul XML de configurare. Este un avantaj important al proiectării DVM, deoarece oferă flexibilitate și posibilitatea de a înlocui ușor citirea din fișierul de configurare cu citirea valorilor dintr-un echipament real de rețea, transformând simulatorul într-un mediator real, fără a fi nevoie de cunoașterea detaliilor de construire a bazei de stocare de date de execuție.

Au fost considerate două abordări pentru citirea valorilor din fișierul de configurare XML: o abordare dinamică și una statică. În abordarea dinamică, în momentul în care o funcție cu apel invers asociată unui atribut YANG dinamic este invocată, fișierul XML este deschis, încărcat în memorie, valoarea respectivă este citită, apoi fișierul este scos din memorie și închis. Scopul acestei abordări era de a simula aspectul dinamic al parametrilor de stare. În cazul în care, între două citiri succesive, valoarea atributelor este schimbată în fișierul de configurare, acest lucru se va reflecta și în valorile trimise către echipamentul de control SDN. Abordarea statică este reprezentată de încărcarea fișierului XML în memorie în momentul primei invocări a funcției cu apel invers și citirea valorii atributului. Apoi, la următoarele citiri, fișierul de configurare va fi deja încărcat în memorie și valorile parametrilor vor fi întoarse de acolo. Astfel, datele vor fi statice, însă va fi nevoie de o singură citire de pe disc. Deoarece timpul necesar citirii de pe disc era foarte mare în momentul în care se



doreau valorile pentru mai multe atribute de execuție, făcând timpul de răspuns al simulatorului extrem de mare, abordarea dinamică nu este prezentă în implementarea simulatorului.

Generarea notificărilor NETCONF este similară cu implementarea din prima versiune a simulatorului DVM. În funcția de inițializare a modulului asociat modelului YANG care descrie notificările se crează un nou fir de execuție care va genera notificări fictive. Conținutul acestora va fi însă dinamic, nu static precum în versiunea anterioară. Detaliile notificării, dar și intervalul de timp dintre două notificări consecutive, vor fi citite din fișierul XML de configurare. Cea de-a doua versiune a simulatorului DVM oferă posibilitatea de a genera două tipuri de notificări, în funcție de conținutul fișierului de configurare: *attributeValueChangedNotification* (notificare care apare în momentul în care valoarea unui atribut se schimbă) și *problemNotification* (reprezintă alarme care sunt generate de către dispozitiv).

A doua versiune a simulatorului DVM oferă o implementare mai flexibilă și mai robustă față de prima versiune. Dacă în versiunea anterioară doar două atribute erau prezente în fișierul de configurare, orice modificare a altui parametru necesitând recompilarea simulatorului, acum DVM le conține pe toate în fișierul XML, oferind posibilitatea dezvoltatorilor de aplicații SDN de a modifica în mod facil aceste valori care vor fi interogate de către echipamentul de control. DVM versiunea 2 expune modelele TR-532 și TR-512.1 în întregime, nu doar un model simplificat, precum prima versiune. Și mecanismul de generare a notificărilor NETCONF a fost îmbunătățit, deoarece detaliile acestora pot fi modificate acum în fișierul de configurare.

## 4.6 Folosirea în contextul demonstrațiilor de concept a DVM versiunea 2

Așa cum prima versiune a DVM a fost foarte importantă în contextul celei de-a doua demonstrații de concept a rețelelor de transport de date fără fir și a doua versiune a acestui simulator a fost o unealtă critică pentru pregătirea celei de-a treia demonstrații de concept. A permis dezvoltatorilor de aplicații SDN să le testeze și să le depaneze într-o manieră facilă, eliminând necesitatea deținerii unor echipamente de rețea și a mediatorului asociat.

SDN nu influențează doar modul în care rețelele sunt controlate și administrate, ci și modul în care sunt organizate proiectele și modul în care anumite cerințe și capacități sunt dezvoltate și implementate. Dacă înainte aplicațiile și dispozitivele de rețea erau oferite împreună, de către producătorii de echipamente, interfețele standard dintre dispozitive și echipamentele de control SDN, sau dintre echipamentele de control și aplicații, oferă posibilitatea companiilor de a oferi doar anumite bucăți din întreaga soluție. Aceste companii pot profita de simulatorul DVM, deoarece le oferă acces la aceste interfețe fără a avea nevoie să dețină echipamente de rețea foarte scumpe.

Avantajul pe care DVM îl oferă și care a fost folosit în cea de-a treia demonstrație de concept ONF este reprezentat de faptul că orice modificare în modelele YANG

dezvoltate poate fi testată cu ajutorul simulatorului, pentru că oferă o modalitate rapidă de a o implementa, simulând astfel, din punctul de vedere al echipamentului de control SDN, interacțiunea cu un dispozitiv de rețea. După dezvoltarea simulatorului și a aplicațiilor SDN, acestea din urmă pot fi testate, chiar înainte ca software-ul din elementele de rețea să fie implementat. Cu toate acestea, este nevoie ca aplicațiile să efectueze teste de integrare cu dispozitivele de rețea, deoarece comportamentul simulatorului nu poate fi identic cu cel al echipamentelor. Simulatorul DVM folosit în cea de-a treia demonstrație de concept nu a simulat dispozitivele de rețea, ci nivelul mediator care oferă interfața NETCONF ce expune modelele YANG dorite. A fost posibilă simularea diferitor topologii de rețea, având dispozitive cu diferite configurări și interfețe de transport de date. Simulatorul DVM a fost și în cazul celor de-a doua și de-a treia demonstrații de concept ONF unealta principală pentru a putea crea o topologie de rețea simulată.

Simulatorul a fost folosit și pentru teste de extensibilitate și de performanță ale aplicațiilor SDN dezvoltate. Fișierul XML de configurare poate fi manipulat foarte ușor, adăugând interfețe de transport de date echipamentelor sau intrări pentru valorile indicatorilor de performanță. De exemplu, un dispozitiv de rețea are nevoie de 24 de ore pentru a genera 96 de intrări ale indicatorilor de performanță de 15 minute și 30 de zile pentru a genera 30 de intrări ale indicatorilor de performanță de 24 de ore. O altă funcție importantă a DVM care a fost folosită în pregătirea demonstrației de concept a fost cea de generare a notificărilor NETCONF. Au putut fi simulate diferite tipuri de notificări care să apară la un interval de timp configurabil.

În mod evident, timpii de răspuns ai simulatorului sunt foarte rapizi, deoarece nu există o comunicație cu un echipament de rețea real și nu există nici nevoia de a procesa valorile atributelor care sunt citite din dispozitivele de rețea. Nevoia de a procesa valorile unor atribute apare în cazul mediatoarelor reale, în momentul în care atributele definite în modelul informațional pentru microunde nu se potrivesc în totalitate cu atributele dispozitivului și o anume procesare a acelei valori este necesară pentru a transforma-o în ce se dorește în modelul YANG.

Tabelul 4.1 ilustrează diferențele între DVM și un mediator real, în diferite situații care au apărut în pregătirea celei de-a treia demonstrații de concept ONF.

Tabelul 4.1: Comparație între comportamentele unui mediator real și al simulatorului în diferite situații [90].

Situația	<i>Mediator real</i>	<i>Simulatorul DVM</i>
Dezvoltarea aplicațiilor SDN	costisitoare	eficientă
Timpul de implementare	greoi	rapid
Topologia de rețea	complex de schimbat	flexibilă, ușor de schimbat
Generarea notificărilor NETCONF	greu de controlat	simplă
Timpi de răspuns	reali	nerealști
Procesare a valorilor atributelor	în unele cazuri	nu este nevoie

Pentru a oferi un suport mai bun testelor de extensibilitate și de performanță ale aplicațiilor, simulatorul DVM ar fi putut implementa timpi de răspuns configurabili la operațiile NETCONF sau generarea de notificări care să urmărească anumite profile temporale, în funcție de nevoile aplicațiilor SDN.

La fel ca în cazul versiunii anterioare a DVM, înregistrarea simulatoarelor la echipamentul de control SDN se face manual, cu ajutorul interfeței grafice a controlerului, care stabilește apoi conexiunea securizată și inițiază o sesiune NETCONF.

## 4.7 Integrarea DVM cu comutatorul software LINC: *LINC-WE*

Așa cum a fost descris anterior, simulatorul DVM permite emularea nivelului mediator care apare în arhitectura SDN a rețelelor de transport de date fără fir. Se pot simula diferite topologii de rețea prin manipularea fișierelor XML de configurare ale diferitelor instanțe ale DVM, făcând simulatoarele să prezinte diferite configurații. Această abordare oferă un bun prim pas pentru simularea rețelelor de transport de date fără fir în contextul SDN, însă nu este suficient, deoarece instanțele DVM sunt independente și rețeaua simulată nu este completă, deoarece nu există legături între dispozitivele simulate. Din acest motiv, această secțiune prezintă încercarea de a integra simulatorul DVM cu un comutator software deja existent, care a fost folosit anterior pentru simulări de rețele optice în contextul SDN: *Legătura Nu Este Închisă* - LINC (Link Is Not Closed) și cu simulatorul de rețele definite prin software, *mininet*. Comutatorul software folosit anterior în simularea rețelelor optice [92–94], bazat pe LINC se numește *Legătura Nu Este Închisă - Extensii Optice* - LINC-OE (Link Is Not Closed - Optical Extensions), astfel că am denumit comutatorul rezultat în urma integrării DVM cu LINC: *Legătura Nu Este Închisă - Emulator Fără Fir* - LINC-WE (Link Is Not Closed - Wireless Emulator) [95].

### 4.7.1 LINC

LINC este un comutator software ce suportă protocolul OpenFlow, scris în limbajul de programare Erlang [96]. A fost proiectat să fie modular, oferind o metodă rapidă de a face prototipuri SDN și de a testa noi caracteristici ale protocolului OpenFlow. Este oferit ca o implementare cu sursă deschisă de către *FlowForwarding.org* și are ca scop oferirea unei soluții prin care utilizatorii pot evalua rapid protocoalele OpenFlow și OF-Config.

Un singur comutator este implementat ca un nod Erlang și cuprinde mai multe comutatoare logice. Acestea, porturile și legăturile dintre porturi sunt descrise de către utilizator prin intermediul unui fișier de configurare, așa cum este ilustrat și în Figura 4.7. Datorită acestei arhitecturi modulare, comutatorul LINC este capabil să ofere mai multe implementări pe care utilizatorul le poate alege, fiecare având asociată o versiune diferită a protocolului OpenFlow (versiunile 1.2, 1.3 sau 1.4). Aceste implementări sunt responsabile de comutarea pachetelor și alterarea tabelor de fluxuri de date.

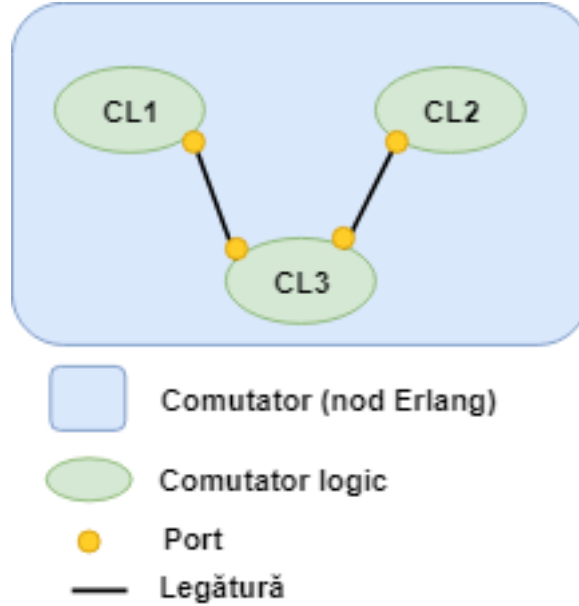


Fig. 4.7: Arhitectura comutatorului software LINC [97].

După cum este prezentat în [97], LINC prezintă mai multe blocuri software: comutatorul capabil OpenFlow, modulul protocolului OpenFlow, modulul protocolului OF-Config. Acestea sunt dezvoltate ca aplicații Erlang separate, respectând principiile *Platforma Telecom Deschisă* - OTP (Open Telecom Platform) ale limbajului Erlang. Există și o componentă separată care se ocupă de conexiunea la un echipament de control SDN. Tabelele de fluxuri, porturile sau tabelele de grup sunt administrate de implementările separate reprezentând diferitele versiuni ale protocolului OpenFlow.

## LINC-OE

Datorită naturii modulare a comutatorului LINC, a apărut o nouă implementare care să acopere cazurile de utilizare ale SDN din domeniul optic: LINC-OE. Acesta reprezintă un simulator de comutatoare optice care suportă extensiile optice ale protocolului OpenFlow. Diferența dintre LINC și LINC-OE este că, în cazul celui din urmă, comutatoarele logice care sunt simulate reprezintă comutatoare optice, astfel că spre deosebire de interfețele electrice, porturile optice oferă mai multe canale de comunicație independente, diferențiate prin lungimea de undă. În implementarea LINC-OE, mesajele ce se transmit printr-un astfel de port optic nu mai sunt pachete Ethernet, ci mesaje Erlang, ce conțin informații adiționale, pe lângă pachetul Ethernet propriu-zis. O altă diferență este dată de faptul că LINC-OE oferă interfețe pentru simularea defectării legăturilor dintre porturi.

## Interfața NETCONF

Comutatorul software LINC expune de asemenea și o interfață NETCONF. Este folosită de către protocolul OF-Config și este disponibilă pentru un comutator doar dacă acel protocol este folosit. În cazul în care este permisă utilizarea acestuia, o nouă aplicație Erlang este pornită în cadrul nodului Erlang: *enetconf*, care pornește un server NETCONF ce așteaptă conexiuni pe adresa IP și portul selectate de utilizator.

Această abordare prezintă un dezavantaj major: un singur server NETCONF este pornit pentru comutatorul LINC, astfel că nu se pot accesa comutatoarele logice care alcătuiesc comutatorul LINC individual prin interfața NETCONF. Acestea pot fi configurate doar prin intermediul protocolului OpenFlow.

De asemenea, modelul informațional YANG care este expus de către server nu este configurabil, astfel încât utilizatorul nu poate alege un model YANG pe care comutatorul software să îl poată folosi. Dacă acest lucru ar fi fost oferit, ar fi fost banală transformarea LINC-OE în LINC-WE, prin înlocuirea modelului YANG expus cu modelul informațional pentru microunde și apoi corelarea atributelor YANG cu parametrii comutatorului LINC.

### 4.7.2 LINC-WE

LINC-WE a apărut ca o încercare de a îmbunătăți simulatorul DVM dezvoltat anterior, prin oferirea unei soluții care să simuleze nu doar dispozitive de rețea independente, ci și legăturile dintre acestea. Astfel, am luat în considerare integrarea DVM cu un comutator software deja existent, LINC, având în vedere și faptul că acesta a fost folosit anterior pentru simularea de rețele optice în contextul SDN, prin LINC-OE.

Spre deosebire de LINC și LINC-OE, LINC-WE nu ilustrează utilizarea protocolului OpenFlow pentru configurarea comutatorului, ci pune accent pe protocolul NETCONF pentru administrarea dispozitivelor de rețea. Oferă totuși suportul pentru protocolul OpenFlow, prin modulele LINC care implementează versiunile 1.2, 1.3 și 1.4 ale acestuia. Deoarece configurarea și administrarea dispozitivelor de transport de date fără fir a migrat de la protocolul OpenFlow (prin dezvoltarea de extensii specifice) la NETCONF, așa cum demonstrează activitatea de cercetare din cadrul ONF, prin apariția modelului informațional pentru microunde, această abordare oferind mai multă flexibilitate. LINC-WE încearcă să construiască pe această bază prin oferirea unui comutator software care să expune modelele YANG propuse de ONF și să poată fi folosit în simulări.

Abordarea aleasă în acest caz a fost înlocuirea aplicației Erlang *enetconf*, care implementa serverul NETCONF pentru comutatorul software LINC cu o aplicație externă, dezvoltată în C, care să expună modelele YANG specifice rețelelor de transport de date fără fir: DVM.

Implementarea serverului NETCONF este dată de simulatorul descris anterior, DVM. Pentru integrarea acestuia cu comutatorul software LINC am înlocuit aplicația Erlang ce implementa suportul pentru acest protocol, *enetconf*, cu o aplicație Erlang nouă, dezvoltată în acest scop: *erl\_yuma*. Aceasta se ocupă de pornirea și oprirea serverului NETCONF implementat pentru DVM, prin deschiderea unui port Erlang

care va avea această sarcină. Codul acestei soluții este oferit cu sursă deschisă și poate fi găsit pe platforma GitHub [98].

Din punctul de vedere al echipamentului de control SDN, acesta va comunica prin interfața NETCONF cu DVM, care a fost pornit anterior din mediul Erlang (prin aplicația *erl\_yuma*). Comunicația dintre aplicațiile dezvoltate în Erlang și în C este administrată de către sistemul de execuție Erlang. Din perspectiva *erl\_yuma*, comunicația cu DVM se face prin mesaje standard Erlang, care sunt transmise prin portul Erlang asociat acesteia. Din cealaltă perspectivă, a DVM, comunicația se realizează prin fluxurile standard de intrare și de ieșire, *stdin* și *stdout*. O imagine de ansamblu a arhitecturii LINC-WE este prezentată în Figura 4.8.

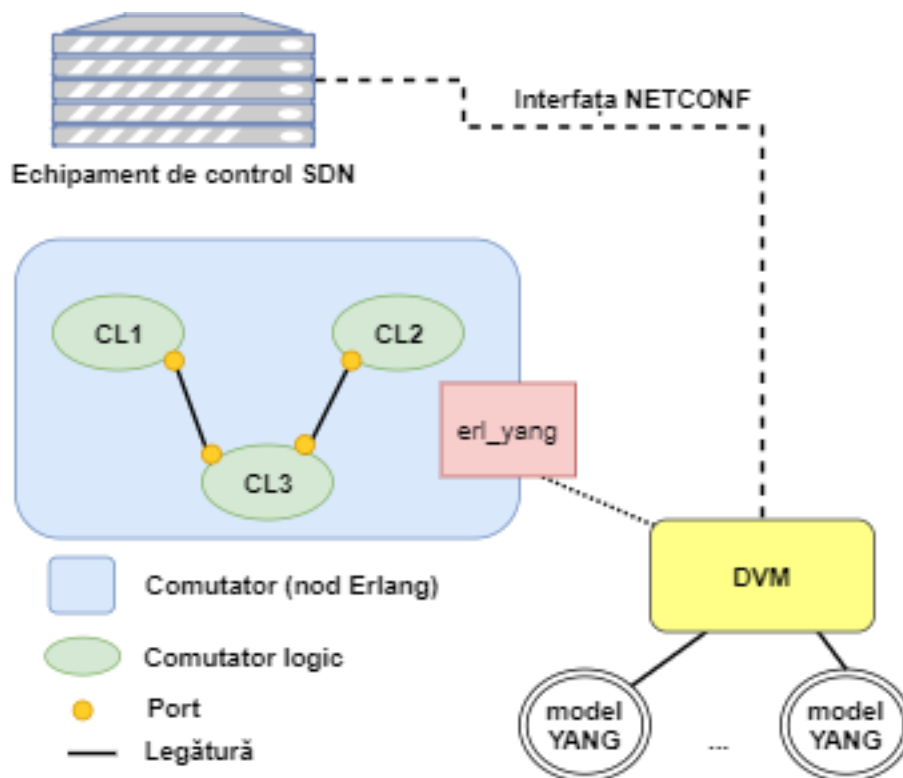


Fig. 4.8: Arhitectura comutatorului software LINC-WE [97].

Restul arhitecturii LINC nu este influențată: există în continuare comutatorul software, care expune o interfață NETCONF (doar că acum este oferită prin DVM) și conține mai multe comutatoare logice, împreună cu porturile acestora și legăturile dintre ele, așa cum a fost definit în fișierul de configurare LINC. Aplicația ce oferă interfața NETCONF, *erl\_yuma*, este pornită doar în cazul în care protocolul OF-Config este activat.

### Interfața NETCONF - Aplicația Erlang

Aplicația care transformă comutatorul software LINC în LINC-WE, prin integrarea cu DVM este *erl\_yuma*. Aceasta este o aplicație Erlang tipică, ce respectă principiile OTP.

Structura aplicației respectă recomandările OTP. În dosarul rădăcină *erl\_yuma* al aplicației sunt create următoarele directoare: *ebin*, ce conține fișierele compilate \*.beam, dar și fișierul de resurse ale aplicației OTP (ce conține informații precum descrierea, versiunea, etc.). Directoarele *include* și *priv* ar trebui să conțină fișierele de incluziune \*.hrl și respectiv aplicațiile externe de care depinde aplicația Erlang. Încă două dosare alcătuiesc aplicația: *src*, unde se găsesc fișierele sursă și *test*, ce conține fișiere de test.

O imagine de ansamblu a aplicației Erlang poate fi văzută în Figura 4.9. Există trei fișiere sursă importante: *ey\_app.erl*, *ey\_sup.erl* și *ey\_server.erl*.

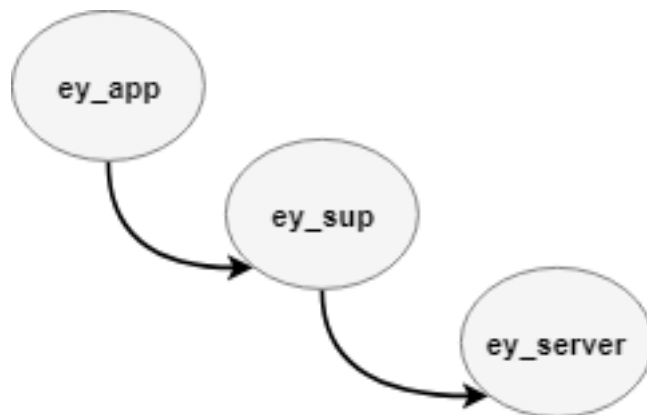


Fig. 4.9: Structura aplicației Erlang *erl\_yuma* [97].

Primul fișier sursă, *ey\_app.erl*, reprezintă implementarea unui comportament *aplicație*, așa cum este definit în OTP. Este o implementare simplă și directă a acestui comportament OTP, care oferă doar două funcții cu apel invers: *start* și *stop*. Prima este folosită pentru a porni procesul care este responsabil pentru implementarea aplicației. Acesta este un proces OTP supervisor.

Cel de-al doilea fișier sursă, *ey\_sup.erl*, implementează un comportament *supervisor* definit de OTP. Acesta este procesul central al aplicației *erl\_yuma* și este responsabil pentru supravegherea acestuia. Este necesară implementarea unei singure funcții cu apel invers oferite de acest comportament: *init*. În momentul în care procesul supervisor este pornit, această funcție este invocată, acesta fiind și locul în care ar trebui să pornim procesul *ey\_server.erl*, conform cu structura aplicației noastre. Nu trebuie să pornim mai multe procese copil în acest punct, deoarece dorim să expunem o singură interfață NETCONF. Astfel, procesul supervisor va porni un singur modul *ey\_server*.

Ultimul fișier sursă care alcătuiește aplicația *erl\_yuma* este *ey\_server.erl*. Din punctul de vedere OTP, acest modul implementează un comportament *gen\_server* (server generic). Acest proces este responsabil de crearea unui port Erlang, care pornește aplicația externă C, reprezentată de DVM. Astfel, procesul va stoca portul care a fost creat. În funcția cu apel invers *init*, practic însemnând atunci când procesul este pornit, un port Erlang este creat. Intern, acest lucru înseamnă crearea unui nou proces, bazată pe o anumită comandă. Comanda este definită ca o variabilă de mediu și este stocată în fișierul de resurse ale aplicației și reprezintă de fapt comanda

necesară pornirii DVM. A fost folosită această abordare pentru a fi ușor de modificat în cazul în care comanda cu care se pornește DVM se va schimba pe viitor. Astfel, aplicația Erlang nu va trebui recompilată.

Un aspect interesant îl constituie modalitatea în care portul Erlang trebuie închis. Când utilizatorul dorește închiderea LINC-WE, trebuie ca și DVM să fie închis, astfel încât să fie pregătit pentru o eventuală viitoare rulare. Acest lucru se face prin implementarea funcției cu apel invers *terminate* a modulului *ey\_server*. Este important de menționat faptul că pentru ca această funcție să fie invocată în momentul închiderii aplicației, aceasta trebuie să proceseze mesaje capcană (*trap message*). Implementarea funcției *terminate* va trebui să închidă instanța DVM care rulează (prin trimiterea unui semnal de închidere a procesului către sistemul de operare).

În prima versiune a LINC-WE nu a fost implementată comunicația dintre aplicațiile Erlang și C. Cu ajutorul acesteia s-ar fi putut manipula parametrii comutatorului software, prin operații NETCONF. Practic, LINC-WE doar crează o instanță un simulator DVM, expunând o interfață NETCONF, care însă nu implementează și modificarea atributelor comutatorului LINC.

### 4.7.3 Concluziile integrării

O unealtă ce oferă posibilitatea de a simula rețele ce conțin dispozitive de transport de date fără fir, care să expună și modelul informațional pentru microunde propus de ONF este importantă atât pentru membrii ONF, care pot testa, revizui și îmbunătăți modelul, cât și pentru dezvoltatorii de aplicații SDN care pot începe implementarea și testarea de software pentru astfel de rețele. Deoarece TR-532 a fost propus de curând, în decembrie 2016, nu există încă suport pentru acesta în simulatoare de rețea consacrate, precum OPNET sau OMNeT++, făcând LINC-WE prima încercare de simulator care să ofere o astfel de interfață unui echipament de control SDN.

LINC-WE oferă posibilitatea unei lansări rapide a unui simulator ce suportă modelul TR-532 și, spre deosebire de DVM, care era oferea doar serverul NETCONF ce expunea modelele informaționale ONF, oferă și posibilitatea de a transmite trafic de date prin rețeaua simulată. Acest lucru este posibil deoarece LINC-WE se bazează pe soluția deja existentă și folosită în rețele optice, LINC-OE. Parte din cercetarea anterioară a fost și integrarea LINC-OE cu simulatorul de rețele definite prin software, *mininet*, astfel că și LINC-WE va fi integrat implicit cu această unealtă.

Această abordare a permis o integrare rapidă a LINC cu DVM. După parcurgerea etapei destul de greoaie de învățare a limbajului de programare Erlang, în care este implementat comutatorul software LINC, integrarea cu DVM a fost facilă, aplicația Erlang ce se ocupă de acest aspect, descrisă anterior, fiind destul de simplă. Aceasta respectă principiile OTP și pornește instanța DVM ce expune modelele YANG dorite.

Un alt avantaj oferit de această soluție este faptul că LINC-WE oferă atât suport pentru protocolul NETCONF, cât și pentru protocolul OpenFlow. Astfel, un echipament de control SDN poate folosi, pe de o parte, interfața NETCONF pentru a configura parametrii dispozitivelor simulate și, pe de altă parte, interfața OpenFlow pentru a administra traficul din rețea prin manipularea tabelor de fluxuri ale echipamentelor.



Principalul dezavantaj al abordării folosite în cadrul LINC-WE este faptul că poate oferi o singură interfață NETCONF, care este asociată cu comutatorul software și nu câte o interfață NETCONF pentru fiecare comutator logic din interiorul topologiei. Soluția evidentă ar fi pornirea câte unei aplicații *erl\_yuma* pentru fiecare astfel de comutator, însă aceasta nu este banală și ușor de implementat, deoarece DVM nu este proiectat să ruleze mai multe instanțe pe aceeași mașină, având anumite lucruri ce nu pot fi împărțite (de exemplu portul pe care așteaptă conexiuni serverul NETCONF, sau fișierul XML de configurare).

Un alt dezavantaj îl constituie faptul că LINC-WE nu este ușor de utilizat. Trebuie urmați mai mulți pași înainte ca acesta să poată fi pornit: instalarea comutatorului software LINC, care implică instalarea mediului Erlang în prealabil; apoi, simulatorul DVM trebuie instalat, după ce a fost instalată anterior soluția software OpenYuma.

Fiind doar prima versiune a LINC-WE, a fost implementată rapid pentru a putea fi folosită pentru teste. Asta înseamnă că se pot aduce multe îmbunătățiri acestei soluții. Un aspect important ce ar putea fi luat în considerare este asocierea parametrilor din modelul informațional pentru microunde cu parametri reali din comutatorul software. Asta implică, însă, o înțelegere mai aprofundată a limbajului de programare Erlang. Implementarea comutatorului LINC ar putea ține cont de acești parametri în simulare. De exemplu, dacă frecvențele emițătoarelor și receptoarelor unor porturi care comunică nu se potrivesc, comunicarea între acestea ar putea fi oprită în implementarea comutatorului.

Data fiind dificultatea limbajului de programare Erlang și limitările descrise anterior, am renunțat la dezvoltarea unei a doua versiuni a LINC-WE și am căutat o alternativă la această abordare care să fie mai facilă și mai ușor de dezvoltat și implementat. Soluția găsită a fost folosirea unei unelte de creare a unor containere software care pot rula într-un mod izolat anumite aplicații: *docker*. Aceasta va fi detaliată în capitolul următor.

# Capitolul 5

## Simulatorul rețelelor de transport de date fără fir (WTE)

Simulatorul rețelelor de transport de date fără fir (*Wireless Transport Emulator - WTE*) a apărut ca o evoluție firească a simulatoarelor prezentate anterior, DVM. Acestea din urmă aveau dezavantajul de a simula elemente de rețea independente, prin implementarea unor servere NETCONF care să expună modelele informaționale dezvoltate de ONF, TR-532 și TR-512. Nu era simulată topologia în întregimea ei, adică să cuprindă și legăturile dintre dispozitivele de rețea. WTE propune o arhitectură care să implementeze și acest aspect, oferind astfel o soluție completă, prin care se poate genera și trafic de date.

Secțiunile următoare vor prezenta arhitectura WTE, detalii despre implementare, dar și modalitatea în care acesta poate fi folosit în prezentarea cazurilor de utilizare propuse în demonstrațiile de concept ONF, chiar după ce acestea s-au încheiat.

### 5.1 Arhitectura

Simulatorul rețelelor de transport de date fără fir a fost proiectat pentru a rula pe o singură mașină Linux și să simuleze acolo o topologie de rețea, folosind diferite unelte. Scopul simulatorului a fost să expună în continuare modelele informaționale de bază și pentru microunde, precum versiunea precedentă, DVM. Astfel, există un fișier de configurare în care se specifică topologia ce se vrea a fi simulată, în limbajul *Notăție de Obiecte JavaScript* - JSON (JavaScript Object Notation). Formatul acestui fișier de topologie este unul fix și este influențat de către nivelurile de transport ale obiectelor LTP definite în modelul informațional de bază. Mai multe detalii despre acest format vor fi date în secțiunea următoare.

Există mai multe unelte care sunt folosite în arhitectura WTE, care împreună alcătuiesc simulatorul. Fiecare dispozitiv de rețea este simulat printr-un container *docker*, în care rulează o imagine *Linux* și serverul NETCONF reprezentat de DVM. Această abordare a fost aleasă pentru a obține o izolare la nivelul sistemului de fișiere, astfel încât mai multe instanțe ale DVM să poată funcționa fără probleme pe aceeași mașină. Interfețele prezente în fiecare echipament sunt reprezentate prin interfețe de

rețea în imaginea Linux, iar legăturile dintre dispozitive se fac cu ajutorul acestora. Fiecare element de rețea are o interfață de administrare, prin care se conectează la echipamentul de control SDN. Pentru a obține o izolare a acestor interfețe (mai exact pentru ca traficul de date care ar putea fi transmis prin interfețele unui dispozitiv să nu tracă prin interfața de administrare), astfel încât echipamentele să nu comunice între ele prin aceste interfețe, au fost folosite *rețelele docker*. Toate aceste elemente sunt ilustrate în Figura 5.1 și vor fi detaliate în continuare.

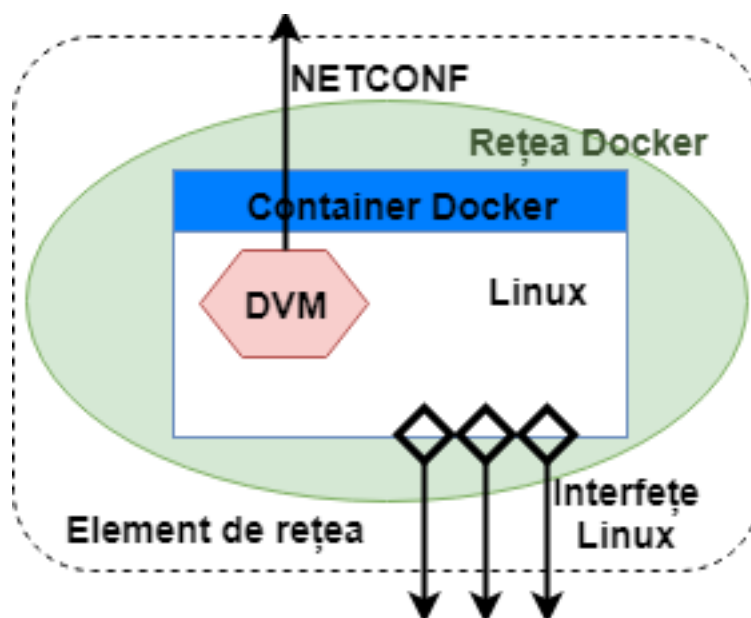


Fig. 5.1: Arhitectura WTE.

*Docker* este o unealtă care permite crearea de containere software în care pot rula aplicații într-un mod izolat față de aplicațiile sistemului de operare gazdă, care lansează aceste containere [99]. Acestea permit împachetarea unei aplicații într-un container *docker*, împreună cu toate aplicațiile software de care ea depinde. Consumul de resurse al unui astfel de container este mult mai redus decât al unei mașini virtuale, deoarece nu se replică tot sistemul de operare, ci doar bibliotecile și procesele necesare aplicației care este virtualizată. După cum este prezentat și în [100], *docker* poate fi utilizat și pentru a produce activitate de cercetare reproductibilă. Astfel, această unealtă este folosită și în cazul WTE pentru a obține izolarea, la nivelul sistemului de fișiere al mașinii gazdă, a aplicației ce implementează serverul NETCONF ce expune modelele informaționale dorite: DVM.

*Rețelele docker* reprezintă o facilitate oferită de soluția *docker* prin care se poate izola și stiva de rețea asociată unei imagini *docker*. Astfel, un container poate fi asociat unei *rețele docker* și imaginile ce aparțin unor *rețele docker* diferite sunt izolate și din punctul de vedere al comunicației dintre ele. Un utilizator își poate crea diferite *rețele docker*, având adrese de rețea sau spații de adrese la alegere.

Din punctul de vedere al legăturilor ce se pot face între aceste containere, mai exact între interfețele Linux ce fac parte din imaginile *docker* care reprezintă elementele de

rețea, au fost considerate două abordări, care vor fi descrise în secțiunea următoare. În prima abordare, s-a încercat folosirea unui comutator software prin intermediul căruia să se facă conexiunile, *Comutatorul Virtual Deschis* - OVS (Open Virtual Switch). Cea de-a doua abordare constă în crearea unei conexiuni între aceste interfețe printr-o pereche *Ethernet virtuală* (*virtual ethernet pair* - **veth**).

În urma proiectării WTE a rezultat o abordare simplă: în momentul inițializării, simulatorul ar trebui să analizeze fișierul care conține topologia ce trebuie simulată. Apoi, ar trebui să construiască *rețelele docker* asociate fiecărui dispozitiv de rețea definit în topologie și să pornească imaginile *docker* necesare, să creeze interfețele Linux asociate cu diferitele niveluri de transport ale obiectelor LTP definite în topologie, după care să construiască legăturile dintre aceste interfețe.

Componentele care alcătuiesc WTE sunt prezentate în Figura 5.2 și sunt următoarele: DVM, care a fost adaptat pentru a putea funcționa în mediul propus de WTE, fișierul JSON care conține topologia ce trebuie simulată, fișierul care conține detalii despre cum ar trebui să fie configurat WTE, comutatorul software OVS (doar în cazul primei abordări propuse pentru reprezentarea legăturilor dintre două elemente de rețea) și un cadru software care să pună toate componentele împreună și să implementeze logica simulatorului. Această ultimă componentă este scrisă în limbajul Python și reprezintă nucleul WTE.

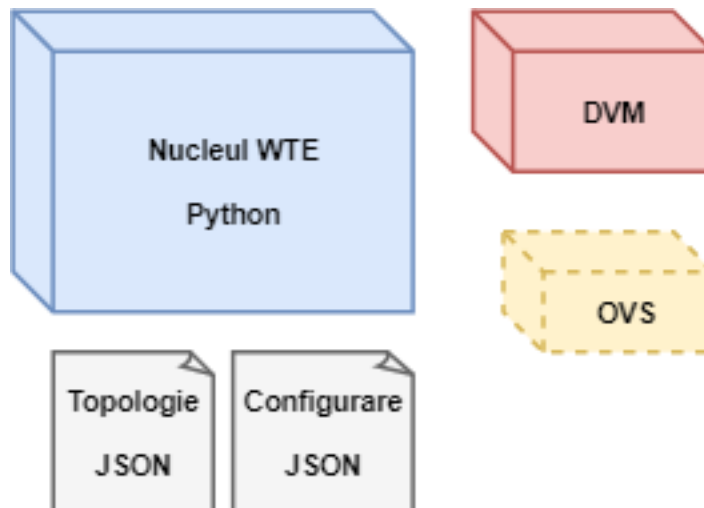


Fig. 5.2: Componentele majore ale WTE.

Nucleul WTE care este scris în limbajul Python este responsabil pentru implementarea infrastructurii de care simulatorul are nevoie și este proiectat să fie modular și flexibil. Este dezvoltat într-o manieră orientată pe obiecte, având clase pentru fiecare componentă importantă de care este nevoie: cadrul general al simulatorului, elementele de rețea, legăturile de date, topologia, etc. Aceasta oferă posibilitatea de extindere folosind, de exemplu, altă soluție care să implementeze un server NET-CONF.

## 5.2 Implementarea

Implementarea WTE se bazează pe cod scris în limbajele Python, pentru nucleul simulatorului, ce implementează infrastructura și pe cod scris în limbajul C, pentru serverul NETCONF, soluția DVM ce a fost dezvoltată anterior fiind adaptată pentru noua abordare. Codul este oferit cu sursă deschisă și poate fi găsit pe platforma GitHub [101]. Paragrafele următoare vor prezenta detalii despre implementare, evidențiind avantajele acestei abordări și motivele pentru care soluția poate fi considerată un adevărat simulator de rețele de transport de date fără fir.

### 5.2.1 Fișierele de configurare

WTE folosește fișiere de configurare, pentru a oferi o interfață simplă de configurare utilizatorilor. Există două fișiere scrise în limbajul JSON, *topology.json* și *config.json*, primul pentru descrierea topologiei care va fi simulată și celălalt pentru a descrie caracteristicile configurabile ale simulatorului. Din aceeași categorie fac parte și modelele informaționale. Fișierele YANG asociate acestora sunt folosite de către simulator pentru a fi expuse de serverul NETCONF.

Fișierul JSON care descrie topologia are un format static, ce va fi prezentat în continuare. Acesta este ilustrat și în Figura 5.3.

Pentru descrierea topologiei unei rețele este suficient să cunoaștem dispozitivele și legăturile dintre acestea. Fiind rețele de transport de date fără fir, în cele mai simple cazuri, putem avea două tipuri de legături între echipamente: fără fir, sau legături Ethernet. Astfel, fișierul de topologie conține două obiecte JSON de tip listă: (i) **elemente de rețea** (*network-elements*), ce descrie dispozitivele de rețea și (ii) **topologii** (*topologies*), reprezentând legăturile dintre echipamente.

Fiecare element de rețea va avea mai multe detalii prezente în fișierul de topologie. Identificatorul unic reprezintă un nume unic ce va fi dat fiecărui dispozitiv de rețea, pentru a-l putea recunoaște în topologia simulată. Tipul elementului se referă la soluția ce va fi folosită pentru implementarea serverului NETCONF ce va expune modelele informaționale dorite. În cazul alegerii DVM, valoarea acestui element este *OpenYuma*. Pentru cea de-a patra demonstrație de concept ONF, o companie a folosit simulatorul WTE pentru dezvoltarea de aplicații SDN, dar a preferat o soluție mai simplă, dar cu mai puține capabilități pentru implementarea serverului NETCONF, astfel că a folosit acest parametru pentru a informa nucleul simulatorului să folosească celălalt server.

Elementul cel mai important al unui dispozitiv de rețea este reprezentat de obiectul JSON de tip listă *interfețe*, fiind responsabil pentru descrierea interfețelor acelui dispozitiv. O interfață a unui echipament are ca echivalent un obiect LTP din modelul informațional de bază. Obiectul *nivel* din lista de interfețe reprezintă nivelul de transport asociat obiectului LTP. Valorile pe care acesta le poate lua sunt descrise în modelul informațional de bază, în Capitolul 3: MWPS, MWS, ETC, ETY și ETH. Fiecare interfață prezintă câteva detalii de care simulatorul are nevoie: identificatorul obiectului LTP, alarmele suportate (simulatorul are nevoie de această valoare, deoarece în modelul informațional pentru microunde, pe baza acestui atribut este definită lista

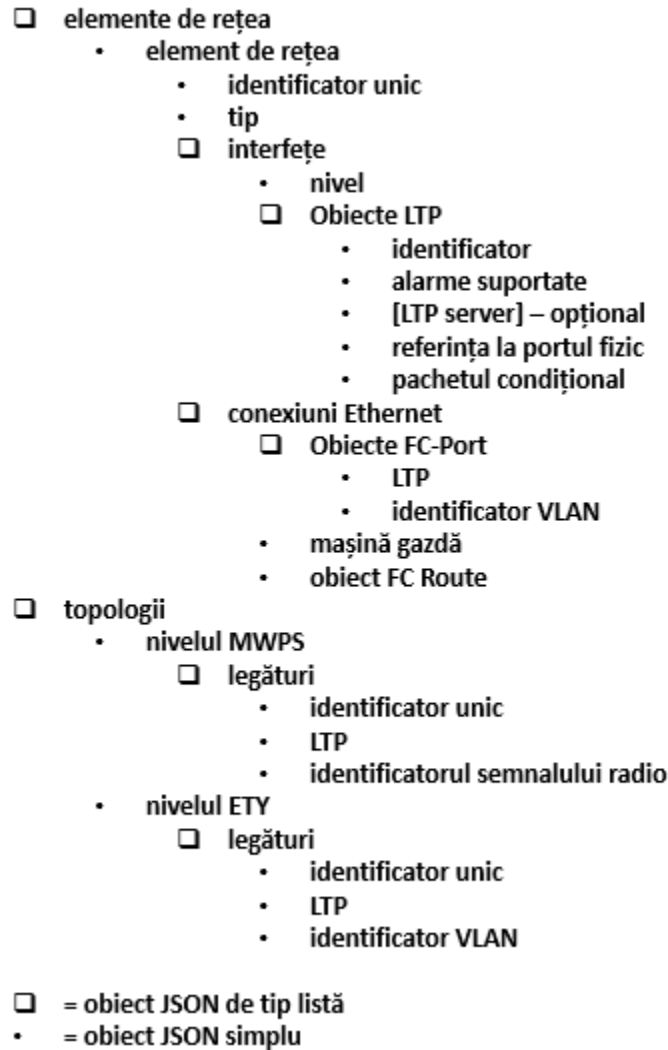


Fig. 5.3: Formatul fișierului JSON care descrie topologia rețelei simulată cu WTE.

alarmelor suportate și aceasta are un număr minim elemente; dacă nu este implementat acest număr minim, modelul YANG prezentat de serverul NETCONF nu va fi valid), obiectul LTP care reprezintă serverul pentru obiectul LTP curent, dacă există (pentru reprezentarea relațiilor de tip client-server dintre interfețe), referința la portul fizic și pachetul condițional folosit de acest obiect LTP.

Următorul obiect JSON care face parte din fișierul de descriere a topologiei este tot un obiect de tip listă, *conexiuni Ethernet*. Acestea reprezintă obiectele FC definite în modelul informațional de bază. Practic, acestea exprimă conexiunile interne între interfețele dispozitivului (de exemplu, traficul de pe o interfață radio R1 va fi dirijat către o interfață Ethernet E2). De aceea, obiectul *conexiuni Ethernet* conține două obiecte de tipul porturi FC, care descriu exact această legătură internă. Obiectul JSON *mașină gazdă* care este prezent aici este folosit pentru simularea unei gazde care folosește această conexiune internă și este utilizat pentru generarea sau recepționarea de trafic în cadrul simulatorului.

Obiectul JSON *topologii* descrie legăturile dintre dispozitivele de rețea. Există două niveluri de transport la care pot exista aceste conexiuni: MWPS, adică legăturile fără fir și ETY, reprezentând conexiunile Ethernet dintre echipamente. Fiecare legătură este reprezentată ca un obiect JSON de tip listă, având doar două elemente, ce descriu cele două capete ale conexiunii, prin identificatorul unic al dispozitivului și identificatorul unic al interfeței.

Cel de-al doilea fișier JSON folosit de WTE este *config.json*. Formatul acestuia poate fi observat în Figura 5.4.

- **controler SDN**
  - **adresă IP**
  - **port**
  - **utilizator**
  - **parolă**
- **adresă rețea de administrare**
- **adresă rețea pentru mașinile gazdă**
- **intervalul de timp dintre notificări NETCONF**
- **înregistrare automată în ODL**
- **simulare bazată pe port**
- **portul NETCONF de bază**
- **portul SSH de bază**
- **adresa IP a mașinii simulatorului**

Fig. 5.4: Formatul fișierului JSON de configurare a simulatorului WTE.

Acesta conține detaliile de conectare la echipamentul de control SDN, în cazul în care se dorește înregistrarea automată a dispozitivelor simulate la controler, constând în adresa IP, portul, numele de utilizator și parola folosite pentru autentificare. Următorul parametru descrie adresa rețelei de administrare care să fie folosită pentru alocarea adreselor IP pentru interfețele de administrare ale echipamentelor simulate. În același mod se poate configura și plaja de adrese IP care să fie alocată mașinilor gazdă, folosite pentru generarea și recepționarea traficului de date prin rețeaua simulată.

Echipamentele simulate pot genera, la fel ca DVM, notificări NETCONF fictive. Intervalul de timp dintre două astfel de notificări este prezent în fișierul JSON de configurare. Parametrul *înregistrare automată în ODL* este folosit de către utilizator pentru a configura simulatorul să facă sau nu înregistrarea automată a dispozitivelor în echipamentul de control specificat prin detaliile de conectare anterioare. Metoda de înregistrare folosită este specifică echipamentului de control ODL, dar poate fi extinsă și pentru altele.

Următorii parametri sunt folosiți în cazul în care utilizatorul nu dorește adrese IP diferite pentru administrarea dispozitivelor simulate. Astfel, echipamentele vor putea fi accesate prin adresa IP a mașinii pe care rulează simulatorul, dar prin porturi diferite. De aceea, este nevoie ca utilizatorul să configureze portul de bază (portul de la care începe numărătoarea în momentul asignării porturilor pentru fiecare element de rețea simulat), pentru protocoalele NETCONF și SSH.

### 5.2.2 Adaptarea DVM pentru WTE

Folosirea DVM fără o adaptare prealabilă la cerințele WTE nu ar fi fost posibilă. În primul rând, DVM a fost transformat astfel încât să poată fi folosit într-un container *docker*. Cerința pentru DVM a fost să permită rularea mai multor instanțe, fiecare în containerul *docker* asociat, însă având o configurație proprie la dispoziție, în funcție de detaliile cuprinse în fișierul de descriere a topologiei.

Am decis astfel împărțirea atributelor prezente în modelele informaționale de bază și pentru microunde în două tipuri: de configurare și de stare. Dacă în versiunile anterioare, baza de stocare de date de execuție se construia manual, pentru toate attributele, în codul C al serverului NETCONF, această abordare nu ar mai fi fost posibilă, deoarece configurația era variată pentru diferitele dispozitive de rețea ce trebuie simulate. Am dezvoltat astfel o soluție automată care să construiască această bază de stocare de date de execuție NETCONF. În cazul atributelor de configurare, soluția a fost folosirea unui fișier XML care să le conțină, împreună cu conceptul bazei de stocare de date de inițializare. Astfel, în momentul în care serverul NETCONF pornește, va încărca în baza de stocare de date de execuție conținutul acelui fișier de inițializare. Dacă structura fișierului XML este corectă, serverul NETCONF va expune interfețele și celelalte detalii oferite de attributele de configurare către echipamentul de control SDN. Această soluție înseamnă, din punctul de vedere al nucleului WTE, construirea acelui fișier XML astfel încât să aibă o structură corectă și să reflecte informațiile din fișierul care descrie topologia ce trebuie simulată.

În cazul atributelor de stare, am renunțat la abordarea folosită anterior pentru DVM, și anume împărțirea acestora în attribute de execuție și de inițializare, deoarece nu s-ar fi putut aplica într-o manieră automată. Soluția în acest caz a fost modificarea uneltei oferite de cadrul software *OpenYuma*, care generează scheletul codului C din fișierul \*.yang asociat modelului informațional. Acesta a fost alterat astfel încât să preia valoarea oricărui atribut de stare prezent în model dintr-un fișier XML care conține valorile acestor attribute. Astfel, din punctul de vedere al nucleului WTE, acest lucru înseamnă construirea unui fișier XML care să conțină attributele de stare prezente în modelele YANG și valorile acestora.

Fiecare instanță de server NETCONF va avea la dispoziție două fișiere XML: unul care conține valorile atributelor de configurare, reprezentând baza de stocare de date de inițializare și celălalt cuprinzând attributele de stare, oferind utilizatorilor diferite valori implicite. Structura acestora este bazată pe modelele informaționale de bază și pentru microunde și este generată automat. Detalii despre această generare vor fi oferite în secțiunea următoare, în contextul prezentării nucleului WTE.

Mecanismul de generare a notificărilor NETCONF fictive a fost îmbunătățit în această versiune și este prezentat în Figura 5.5.

Până acum, generarea acestora se baza pe intervalul dintre două notificări consecutive definit în fișierul de configurare al DVM și pe detaliile notificării ce se preluau din același loc. Acum, abordarea este diferită. În cazul notificărilor de tip *valoarea atributului s-a schimbat* - *attributeValueChanged*, am modificat unealta care generează automat scheletul de cod C asociat serverului NETCONF, astfel încât fiecare funcție cu apel invers asociată unui atribut configurabil va invoca funcția ce trimite această



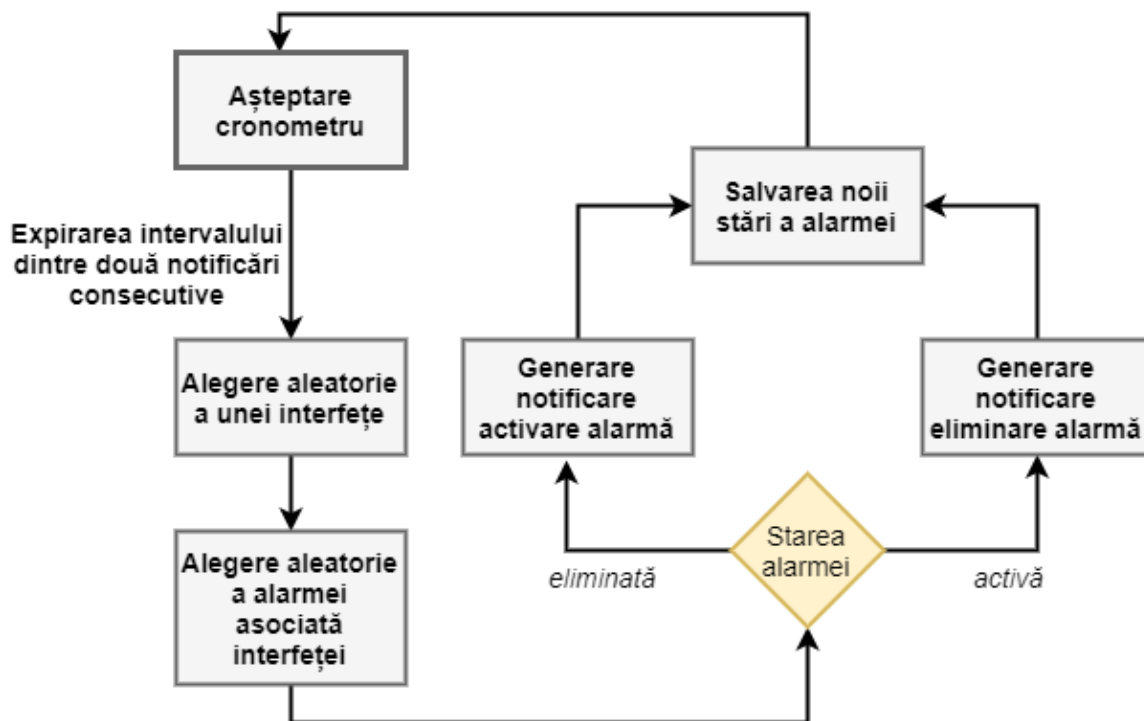


Fig. 5.5: Organigrama mecanismului de generare a notificărilor NETCONF fictive al WTE.

notificare, conținând valorile reale de care este nevoie: obiectul cărui i s-a schimbat valoare și noua valoare a acestuia. În acest mod, notificările de schimbare a valorii unui atribut nu mai sunt fictive, ci vor fi recepționate de clienții NETCONF care s-au abonat la primirea acestora, chiar în momentul în care valoarea se schimbă. În cazul notificărilor de tip *problemă* - *problemNotification*, sau alarmele pe care le generează dispozitivele de rețea, a fost păstrat principiul declanșării acestora în mod fictiv. Prin natura modelului informațional pentru microunde, fiecare interfață are asociată o listă de probleme pe care le poate avea și este definit și un număr minim de elemente pe care această listă să le aibă. WTE profită de acest aspect, implementând următorul mecanism de generare a alarmelor fictive: pentru fiecare problemă se retine starea acesteia: *activă* sau *eliminată*. Când intervalul dintre două notificări fictive consecutive s-a scurs, simulatorul se va pregăti să trimită o nouă notificare, astfel: se va alege în mod aleator o interfață a dispozitivului, iar din lista de probleme asociată interfeței se va alege, din nou în mod aleator, o alarmă. În cazul în care starea acesteia este *activă*, se trimite o notificare care conține eliminarea acestei alarme și se schimbă starea asociată acestei probleme în *eliminată*. Dacă starea acesteia era *eliminată*, se va trimite o notificare care conține activarea acestei probleme și starea ei va fi schimbată în *activă*. Acest mecanism oferă dezvoltatorilor de aplicații SDN un oarecare realism al simulatorului, alarmele fiind generate aleatoriu și, în cazul în care o alarmă era deja prezentă în echipamentul de control, există și posibilitatea ca această problemă să fie eliminată acum.

### 5.2.3 Nucleul simulatorului WTE

Nucleul simulatorului WTE reprezintă codul de bază ce oferă infrastructura care, împreună cu celelalte unelte software, furnizează funcționalitatea propusă: emularea unei rețele de transport de date fără fir, în același timp expunând modelele informaționale dezvoltate de ONF, TR-532 și TR-512. Este implementat în limbajul Python, folosind o metodă orientată pe obiecte, fiind astfel flexibil și modular, oferind posibilitatea unei extinderi facile. Această abordare este similară cu cea folosită în cadrul celui mai important simulator pentru rețele definite prin software, *mininet*. Asemănările, dar și motivele pentru care WTE nu a fost dezvoltat pe infrastructura oferită de *mininet* vor fi detaliate în capitolul următor. Diagrama de clase a nucleului WTE este prezentată în Figura 5.6.

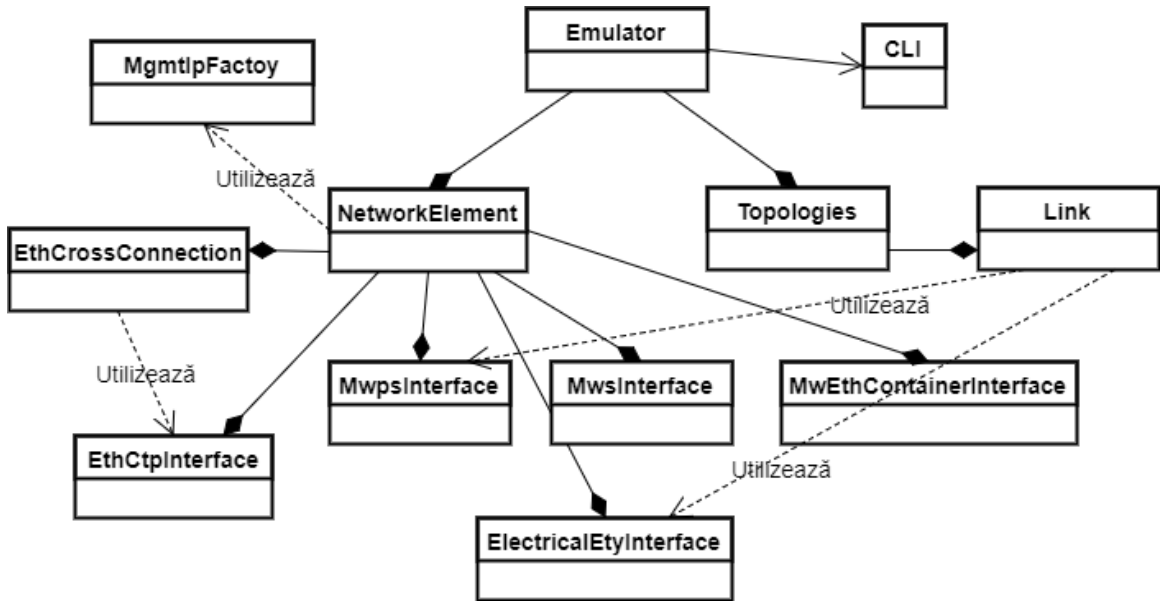


Fig. 5.6: Diagrama de clase a nucleului WTE.

Clasele proiectate pentru WTE sunt folosite în scopul reprezentării obiectelor care au o anumită semnificație în interiorul simulatorului. Astfel, clasa *Emulator* este cea mai importantă, asigurând mediul în care toate celelalte instanțe ale obiectelor WTE vor fi create. Respectă modelul de programare obiect orientată *Singleton*, însemnând că un singur obiect de acest tip va exista în momentul execuției programului. Este responsabilă de încărcarea și analiza fișierelor JSON de configurare a simulatorului. Parametrii din fișierul *config.json* vor fi salvați ca variabile de execuție, pentru a putea fi folosiți ulterior. În cazul fișierului *topology.json*, clasa *Emulator* este responsabilă de analiza acestuia și de crearea obiectelor reprezentând elementele de rețea, respectiv legăturile de date dintre interfețele acestora. Această clasă oferă și metode pentru executarea unor comenzi în mașina Linux sub care rulează WTE.

Clasa *NetworkElement* este folosită pentru reprezentarea unui dispozitiv de rețea. În funcție de fișierul care descrie topologia, vor fi create mai multe instanțe ale acestei clase de către obiectul *Emulator*. Atributele clasei sunt reprezentate, printre altele,

de numele elementului de rețea, tipul acestuia (acest tip se referă de fapt la soluția aleasă pentru implementarea serverului NETCONF asociat, care poate fi DVM sau altă variantă ce poate fi dezvoltată de către utilizator), lista conexiunilor Ethernet interne (reprezentând practic obiectele FC din modelul informațional de bază), scheletul fișierului XML care conține atributele YANG configurabile, necesar pentru inițializarea serverului NETCONF și scheletul fișierului XML care conține atributele de stare, din care DVM va returna valorile asociate acestora. Clasa *NetworkElement* este responsabilă și pentru menținerea listei de obiecte reprezentând interfețele pe care acel dispozitiv le prezintă. Oferă metode pentru crearea acestora, dar și pentru alterarea fișierelor XML astfel încât să reflecte configurația definită de către utilizator în topologie. Celelalte metode importante pe care această clasă le oferă sunt în legătură cu lucrul cu containerul *docker* asociat elementului de rețea: executarea de comenzi în interiorul containerului, inițializarea containerului, copierea fișierelor XML asociate DVM în interiorul containerului *docker*, etc.

Există mai multe clase care definesc obiecte de tip interfață a unui echipament de rețea, în funcție de tipul acestora. Tipul lor este dat de nivelul de transport specificat în modelul informațional de bază. Astfel, clasele asociate interfețelor de rețea sunt:

- *MwpsInterface* - asociată obiectelor LTP de pe nivelul de transport MWPS;
- *MwsInterface* - asociată obiectelor LTP de pe nivelul de transport MWS;
- *MwEthInterface* - asociată obiectelor LTP de pe nivelul de transport ETC;
- *ElectricalEtyInterface* - asociată obiectelor LTP de pe nivelul de transport ETY;
- *EthCtpInterface* - asociată obiectelor LTP de pe nivelul de transport ETH.

Toate aceste clase oferă funcționalitatea asociată unei interfețe, cu elementele specifice date de nivelul de transport la care operează (de exemplu relația client-server dintre obiectele LTP, metode pentru alterarea fișierelor XML asociate DVM cu informații despre interfața respectivă, etc.).

Clasa *Topology* reprezintă topologiile care se definesc în fișierul de configurare. Acestea pot fi prezente la două niveluri, MWPS și ETY. Aceste obiecte conțin de fapt obiectele de tip legătură, ce definesc topologia de rețea.

Clasa *Link* reprezintă o legătură dintre două interfețe de rețea. Este responsabilă de stocarea celor două capete ce formează această legătură, dar și să valideze faptul că aceste capete sunt valide (există obiecte de tip interfață create anterior, între care se va putea face legătura efectivă). O metodă importantă pe care această clasă o oferă este cea responsabilă de crearea efectivă a legăturii (prin OVS sau prin perechi Ethernet virtuale, așa cum va fi descris ulterior) dintre interfețele de rețea Linux prezente în containerul *docker*.

Figura 5.7 prezintă o diagramă de secvență simplificată ce arată inițializarea WTE.

Construirea fișierelor XML folosite de către DVM este responsabilitatea fiecărui obiect dispozitiv de rețea, împreună cu interfețele sale asociate. Fiecare obiect va altera modelul informațional adăugând informațiile care îl reprezintă, astfel încât serverul NETCONF să le pună la dispoziție aplicațiilor SDN. Apoi, aceste fișiere vor

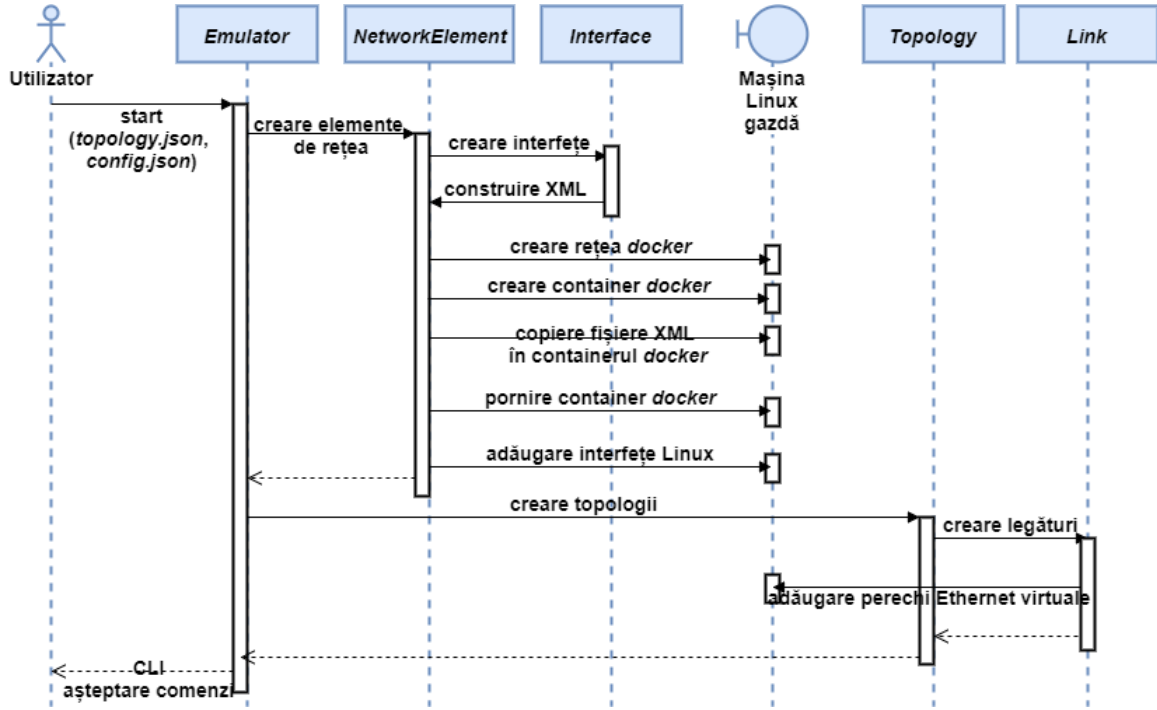


Fig. 5.7: Diagrama de secvență simplificată a inițializării WTE.

fi copiate în interiorul containerului *docker* asociat și vor fi folosite de către DVM pentru inițializarea serverului NETCONF și citirea valorilor atributelor de stare.

### 5.2.4 Reprezentarea interfețelor unui dispozitiv de rețea

Fiecare interfață de rețea a echipamentelor, cu alte cuvinte fiecare obiect LTP din modelul informațional de bază, indiferent de nivelul de transport pe care se află (MWPS, MWS, ETC, ETY, ETH), va fi reprezentată ca o interfață Linux, în interiorul containerului *docker* asociat fiecărui dispozitiv de rețea. Legăturile interne din acest container, între interfețele de rețea, vor fi descrise în paragrafele următoare. Pentru o mai bună înțelegere, acestea sunt prezentate în Figurile 5.8 și 5.9.

Toate obiectele LTP vor fi adăugate ca interfețe Linux având drept nume identificatorul unic universal asociat acestuia, așa cum este definit de către utilizator în fișierul de topologie al simulatorului. Adăugarea se va face cu ajutorul utilitarului *ip*, oferit de Linux. Obiectele LTP care se află la nivelurile MWPS sau ETY reprezintă interfețe ale dispozitivului de rețea cu exteriorul (radio, respectiv Ethernet). Astfel, dacă nu sunt definite legături în topologie care să utilizeze obiectul respectiv, acesta va fi reprezentat în containerul *docker* drept o *interfață fictivă de rețea (dummy interface)*. Adăugarea obiectului în Linux, în cazul în care acesta face parte dintr-o legătură va fi descrisă în sub-sectiunea următoare.

Obiectele LTP de pe celelalte niveluri de transport vor avea, obligatoriu, o relație de tip client-server. Relațiile, așa cum sunt definite în modelul informațional pentru microunde, sunt următoarele: obiectele de pe nivelurile MWPS și ETY nu își pot

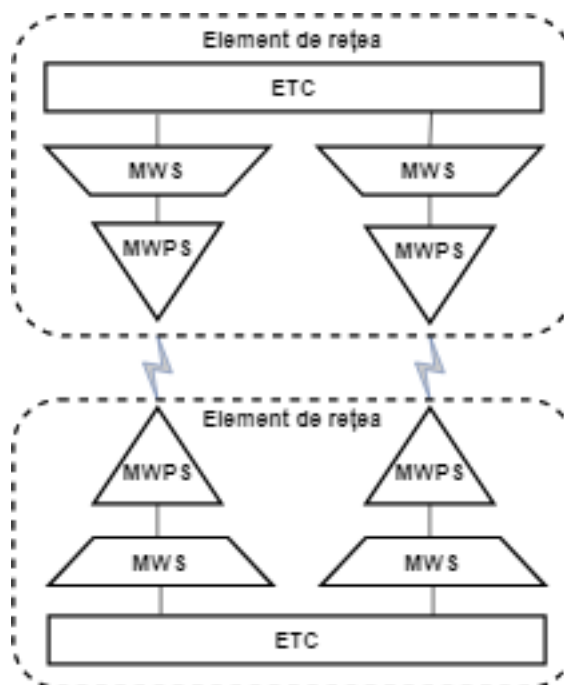


Fig. 5.8: Obiectele LTP în contextul legăturii dintre două dispozitive de transport de date fără fir.

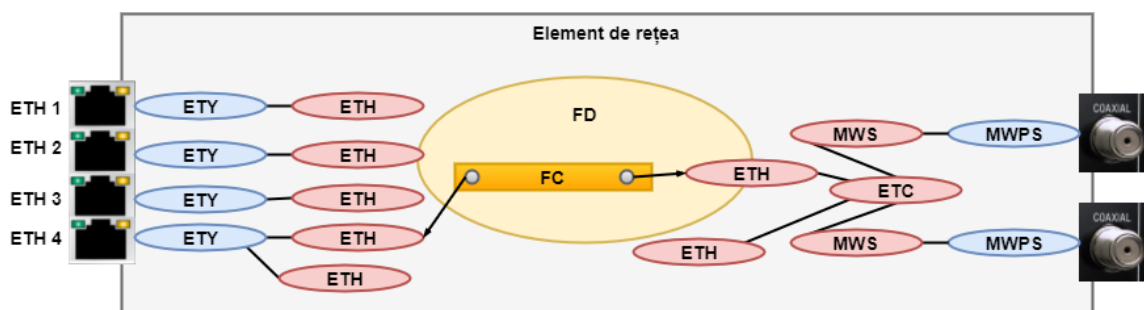


Fig. 5.9: Obiectele LTP în interiorul unui dispozitiv de transport de date fără fir.

asuma rolul de client. Un obiect LTP de pe nivelul MWS va avea drept server un obiect LTP de pe nivelul MWPS, în timp ce un obiect LTP de pe nivelul ETC va avea drept server un obiect LTP de pe nivelul ETC. Obiectele LTP de pe nivelul ETH pot avea ca server obiecte LTP de pe nivelurile ETC sau ETY. Figura 5.9 prezintă aceste legături. Soluția pentru reprezentarea acestor relații de tip client-server în Linux este folosirea interfețelor de tip *legătură* (*bond*). O interfață de tip *legătură* este o conexiune logică ce poate agrega mai multe interfețe. În acest mod, putem privi conexiunea logică, sau agregatorul, ca fiind clientul, iar interfețele care sunt agregate ca fiind serverul. Considerând exemplul din Figura 5.8, pentru un element de rețea, obiectele LTP de pe nivelul MWS vor fi reprezentate ca interfețe de tip *legătură*, care agregă o singură interfață fiecare, dintre cele asociate obiectelor LTP de pe nivelul MWPS, în timp ce obiectul LTP de pe nivelul ETC va fi reprezentat

printr-o interfață de tip *legătură* ce va agrega cele două interfețe logice create pentru obiectele LTP de pe nivelul MWS. Există mai multe moduri de funcționare pentru interfețele de tip *legătură*, dar cel ales pentru WTE este acela în care distribuția traficului pe conexiunile ce fac parte din această interfață logică este una simplă: conexiunile agregate sunt iterate și li se transmite câte un pachet de date, pe rând (*round-robin*).

Obiectele LTP de pe nivelul ETH sunt reprezentate ca interfețe virtuale de tip *vlan*. Deoarece acestea au obligatoriu un identificator VLAN asociat, unei interfețe care reprezintă obiecte LTP de pe nivelurile ETY sau ETC i se va putea asocia o interfață virtuală ce va semnifica obiectul LTP de pe nivelul ETH.

Reprezentarea obiectelor de tip conexiuni Ethernet (FC) este diferită față de celelalte. În acest caz, legătura dintre două obiecte de tip LTP de pe nivelul ETH se face printr-o interfață logică de tip *bridge*. Acestea i se adaugă cele două interfețe de tip *vlan* ce reprezintă cele două capete ale conexiunii Ethernet, astfel traficul va fi dirijat, prin interfața de tip *bridge*, între acestea.

### 5.2.5 Legăturile dintre interfețele elementelor de rețea

Au existat două abordări pentru reprezentarea legăturilor dintre elementele de rețea: prima metodă consta în realizarea unor conexiuni între două interfețe printr-un comutator software OVS, iar cea de-a doua în crearea unei legături logice între două interfețe prin perechi Ethernet virtuale (*veth pairs*). Cea din urmă a fost aleasă pentru implementare, însă vor fi prezentate ambele metode, pentru a înțelege de ce am făcut această alegere. Abordările considerate sunt ilustrate în Figura 5.10.

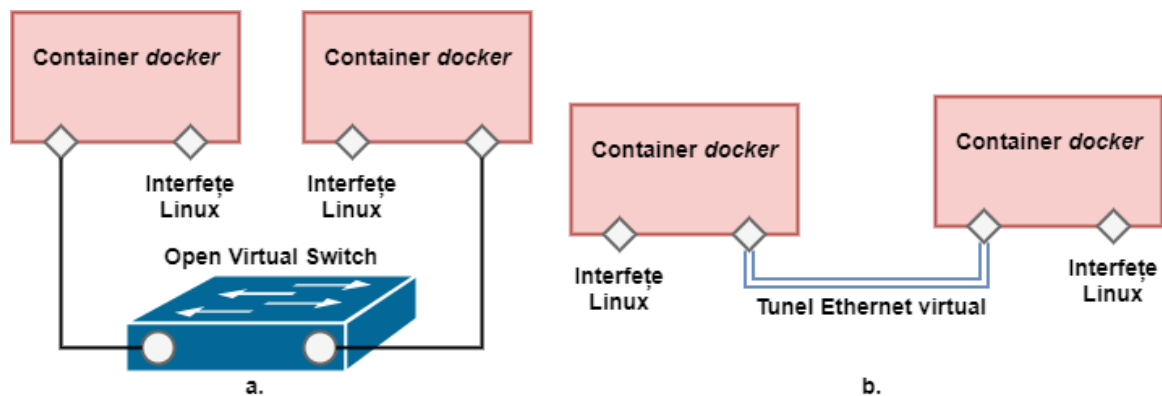


Fig. 5.10: Legăturile dintre interfețele elementelor de rețea considerate pentru WTE: a) prin OVS; b) prin perechi Ethernet virtuale

Prima soluție considerată pentru reprezentarea legăturilor dintre interfețele elementelor de rețea a fost prin intermediul unui comutator software OVS. Astfel, pentru fiecare legătură trebuia creat un comutator OVS, la care cele două capete ale legăturii (câte unul în fiecare container *docker* asociat elementului de rețea) trebuiau legate. Ar fi fost posibilă și crearea unui singur comutator OVS care să cuprindă toate legăturile din topologie, însă apoi trebuiau implementate restricții la nivelul acestuia, pentru

a nu exista legături nedorite (între alte interfețe decât cele specificate în fișierul de topologie). Modelarea legăturii fără fir era posibilă prin aplicarea unor restricții de calitate de servicii - QoS (Quality of Service) asupra conexiunii respective, în interiorul comutatorului OVS (de exemplu lărgimea de bandă să fie una specifică unei legături între echipamente de transport de date fără fir). Această abordare prezenta însă un dezavantaj major. Presupunând că o interfață Linux dintr-un container *docker* ar fi fost dezactivată, era de așteptat ca acest lucru să fie vizibil și în celălalt capăt al legăturii și interfața îndepărtată să raporteze o stare operațională *închisă*. Deoarece legătura era făcută prin comutatorul software OVS, acest lucru nu s-ar fi întâmplat, pentru că stările porturilor din comutator sunt independente și nu s-ar fi propagat între porturile prin care se face conexiunea. Acest dezavantaj major a condus la căutarea unei alte soluții pentru reprezentarea acestor legături. O altă problemă pe care această abordare o ridică este consumul de resurse. Fiecare legătură va avea propriul comutator OVS asociat, astfel că pentru o topologie de rețea cu multe conexiuni, va exista un consum de resurse nejustificat de mare. De asemenea, instalarea WTE pe mașina Linux gazdă va fi mai greoaie, deoarece ar implica și instalarea prealabilă a OVS.

Răspunsul la provocările menționate anterior constă în următorul concept pus la dispoziție de Linux: perechi Ethernet virtuale. Acestea reprezintă perechi de interfețe virtuale Linux de rețea care sunt conectate printr-un *tunel*. Tot traficul care va intra într-o interfață va fi dirijat către interfața pereche, ca și când acestea ar fi conectate printr-un cablu Ethernet virtual. În cazul WTE, în momentul în care se încearcă reprezentarea unei legături din fișierul de topologie, o astfel de pereche este creată. Apoi, un capăt al conexiunii va fi asociat containerului *docker* asociat primului dispozitiv de rețea al legăturii, iar celălalt capăt celui de-al doilea container *docker*. În acest mod, în momentul în care una dintre interfețe este dezactivată, acest lucru se va propaga imediat la interfața îndepărtată, care va raporta o stare operațională *închisă*. Modelarea unei legături fără fir se poate face în mod facil folosind această abordare. Linux oferă utilitarul *tc* - Traffic Control. Acesta poate instrui nucleul sistemului de operare să altereze caracteristicile unei interfețe virtuale, precum: debitul interfeței, latența pe care interfața o introduce în traficul pe care îl transportă, procentul de pachete pe care interfața de poate pierde, etc.

Pe baza considerentelor expuse anterior, a fost implementată ce-a de-a doua soluție pentru reprezentarea legăturilor dintre interfețele echipamentelor de rețea: folosirea perechilor Ethernet virtuale.

### 5.2.6 Modelarea conexiunilor dintre dispozitivele simulate ca legături fără fir

O parte importantă a WTE, care este reprezentativă pentru considerarea acestuia ca fiind un simulator de rețele de transport de date fără fir, nu doar o unealtă care expune modelele informaționale dezvoltate de ONF conform cu fișierul în care este definită topologia, este dată de modelarea conexiunilor dintre dispozitivele de rețea simulate având caracteristicile unei legături fără fir.

Caracteristicile ce pot fi alterate prin unealta **tc** oferită de Linux, care sunt relevante pentru simularea unei legături fără fir, sunt reprezentate de debitul interfeței, întârzierea pachetelor ce tranzitează această conexiune și procentul de pachete pierdute.

Debitul unei interfețe este influențat de mai mulți parametri, dintre care unii sunt configurabili. TR-532 oferă și o formulă de calcul pentru acest debit, tocmai pentru ca aplicațiile SDN ce se folosesc de acest model să fie aliniate asupra acestei metode. Ecuația 5.1 descrie această metodă de calcul.

$$\begin{aligned}
 txCapacity = & AirInterface :: AirInterfaceConfiguration :: txChannelBandwidth \\
 & * \log_2(AirInterface :: AirInterfaceStatus :: modulationCur) \\
 & * AirInterface :: AirInterfaceStatus :: codeRateCur * 0,85
 \end{aligned} \tag{5.1}$$

unde:

$txCapacity$  = debitul interfeței de rețea, în kbps  
 $txChannelBandwidth$  = lărgimea de bandă a emițătorului, în kHz  
 $modulationCur$  = schema de modulație folosită, în număr de simboluri  
 $codeRateCur$  = procentul informației utile din informația transmisă

Rata de transmisie a încărcăturii utile, reprezintă produsul dintre rata de transmisie brută și procentul informației utile din informația transmisă ( $codeRateCur$ ), unde acesta reprezintă numărul de biți ai încărcăturii utile, raportat la numărul total de biți transmiși. Acest procent apare din cauza redundanței introduse pentru corectarea de erori și alte informații care nu fac parte din încărcătura utilă și are o valoare tipică între 0,85 și 0,95 [102]. În cazul WTE, am considerat valoarea acestei rate curente a codului ca fiind 0,9.

Rata de transmisie brută este definită ca produsul dintre rata simbolurilor și numărul de biți per simbol. Valoarea acestuia din urmă este dată de modulația folosită la transmisie. Modulația tipică folosită în echipamentele de transport de date fără fir este Modulația de Amplitudine în Cuadratură - QAM (Quadrature Amplitude Modulation). În acest caz, numărul de biți per simbol este definit ca  $\log_2$  din numărul de constelații (reprezentat în modelul informațional prin  $modulationCur$ ). Rata simbolurilor tipică pentru acest tip de echipamente este de aproximativ 85% din lărgimea de bandă ( $txChannelBandwidth$ ) a canalului de transmisie [102].

Această formulă este implementată în cadrul WTE. În momentul în care unul dintre atributele unei interfețe pe baza căreia se face calculul se schimbă (de exemplu o aplicație SDN modifică lărgimea de bandă folosită de o interfață a unui echipament, sau modulația care se folosește pentru transmisie), instanța DVM asociată dispozitivului de rețea va calcula debitul interfeței respective și apoi va folosi utilitarul Linux **tc** pentru a modifica această caracteristică a interfeței. În acest mod, legăturile fără fir vor fi modelate în simulator astfel încât să semene cu cele din rețelele reale, din punctul de vedere al ratei de transmisie.



În legătură cu latența pe care o introduc echipamentele de transport de date fără fir, tipic aceasta este de ordinul zecilor sau sutelor de microsecunde [102]. Această plajă de întâzieri nu pot fi simulate în cadrul WTE, deoarece utilitarul *tc* oferă modificarea latenței unei interfețe de rețea doar cu o rezoluție de o milisecundă. Pentru a putea beneficia de rezoluții mai mici, de ordinul microsecundelor, imaginea sistemului de operare Linux din containerul *docker* ar trebui modificată și nucleul sistemului de operare recompilat, astfel încât să permită utilizarea unor cronometre de sistem de timp real. Versiunea curentă a WTE nu implementează acest comportament, dar acesta poate constitui subiectul unei alte cercetări, pentru a modela legăturile dintre interfețele dispozitivelor de rețea simulate cât mai aproape de realitate.

Alterarea procentului de pachete pierdute de o interfață de rețea nu este încă folosit în WTE, deoarece presupunem că legătura fără fir este realizată în condiții atmosferice optime, astfel încât să nu se piardă pachete. Ar putea fi totuși utilizat, spre exemplu prin expunerea unei comenzi în linia de comandă a simulatorului, care să crească sau să scadă acest procent pentru o conexiune. Astfel, s-ar putea simula condiții atmosferice potrivnice în anumite părți ale topologiei de rețea. În acest mod ar putea fi testată, de exemplu, o aplicație SDN care să detecteze faptul că există probleme în rețea (din cauza unei furtuni se pierd pachete, sau scade modulația folosită la emisie) și să dirijeze traficul prin altă parte a topologiei.

### 5.2.7 Linia de comandă

WTE dispune de un modul care oferă o interfață prin linie de comandă utilizatorului. După ce acesta pornește simulatorul, la sfârșitul inițializării (adică după ce au fost create toate obiectele asociate cu topologia de rețea specificată în fișierul de configurare) se lansează modulul care implementează interfața prin linie de comandă - CLI. Acesta oferă utilizatorului posibilitatea unei interacțiuni simple cu simulatorul, prin furnizarea unor comenzi clare, simplu de folosit. O captură de ecran a acestei interfețe este prezentată în Figura 5.11.

```
Adding Network element Simulator-1...
Creating docker container Simulator-1...
Adding Network element Simulator-2...
Creating docker container Simulator-2...
Adding Network element Simulator-3...
Creating docker container Simulator-3...
Emulator started successfully!
*** Starting CLI:

WirelessTransportEmulator>
WirelessTransportEmulator>
```

Fig. 5.11: Captură de ecran a interfeței prin linie de comandă a WTE.

Comenzile implementate sunt următoarele:

- *afișare noduri* - afișează pe ecran numele tuturor dispozitivelor de rețea simulate;

- *afișare informații nod* - afișează pe ecran toate informațiile disponibile despre un anumit echipament de rețea simulat: nume, adresa IP a interfeței de administrare, portul folosit de serverul NETCONF, lista interfețelor nodului respectiv împreună cu nivelul de transport pe care acesta se află;
- *afișare legături* - afișează pe ecran toate legăturile dintre interfețe prezente în topologia simulată, inclusiv nivelul de transport la care este realizată aceasta (MWPS sau ETY);
- *pornire terminal* - pornește un terminal care se leagă la containerul *docker* asociat unui element de rețea, oferind acces la mediul Linux din interiorul imaginii. Acesta poate fi folosit de utilizator pentru a interacționa într-un mod facil și direct cu elementul de rețea respectiv.
- *înregistrare / anularea înregistrării ODL* - această comandă oferă posibilitatea de a înregistra / anula înregistrarea în mod automat pentru elementele de rețea la / de la echipamentul de control SDN (în particular ODL). Detaliile de conectare la echipamentul de control SDN sunt cele specificate în fișierul de configurare al WTE;
- *închidere* - această comandă poate fi folosită de către utilizator pentru închiderea simulatorului (împreună cu toate obiectele pe care acesta le-a creat în interiorul mașinii Linux gazdă).

## 5.2.8 Generarea traficului de date

Pentru a oferi o soluție completă de simulare, inclusiv validarea topologiei și a legăturilor create, WTE pune la dispoziție utilizatorilor și posibilitatea de a genera trafic de date între anumite puncte ale rețelei. În acest scop este folosită unealta software *iperf3* [103, 104]. Aceasta este instalată în sistemul de operare Linux din imaginea *docker* folosită de fiecare dispozitiv de rețea simulat.

Toate conexiunile descrise până acum sunt făcute la nivelul 2 (legătură de date) din stiva OSI (Open Systems Interconnection). Astfel, interfețele Linux din containerul *docker* nu folosesc adrese IP (în afară de cea de administrare). Generarea de trafic se face între două interfețe, una jucând rolul de client, iar cealaltă de server. Pentru ca unealta *iperf3* să poată funcționa, este nevoie de adăugarea de adrese IP celor două interfețe. WTE poate face acest lucru în mod automat, dacă în fișierul care descrie topologia, obiectului *conexiune Ethernet* i se adaugă un atribut *gazdă*. Astfel, acelui obiect i se asociază, la nivel teoretic, un port capabil să genereze sau să recepționeze trafic. Practic, o adresă IP este adăugată la interfața asociată acelui obiect. Apoi, utilizatorul va putea genera trafic între oricare două astfel de porturi.

Unealta *iperf3* permite generarea de trafic TCP (Transmission Control Protocol) sau UDP (User Datagram Protocol), oferind în același timp posibilitatea de a măsura latența pachetelor, variația acestei latențe (jitter) sau lărgimea de bandă a canalului de comunicație folosit.

## **5.3 Folosirea în contextul demonstrațiilor de concept**

### **5.3.1 Altă implementare de server NETCONF**

# Capitolul 6

## Rezultate și discuții

### 6.1 Evaluarea soluțiilor propuse

SDN și rețelele actuale..

### 6.2 Comparatie între WTE și alte abordări

Istoria rețelelor definite prin software.

### 6.3 Demonstrarea cazurilor de utilizare cu ajutorul WTE

Standardizare: ONF, etc..



# Capitolul 7

## Concluzii

### 7.1 Rezultate obținute

Istoria rețelelor definite prin software.

### 7.2 Contribuții originale

SDN și rețelele actuale..

### 7.3 Lista contribuțiilor originale

Standardizare: ONF, etc..

### 7.4 Perspective de dezvoltare ulterioară

Standardizare: ONF, etc..



# Bibliografie

- [1] Thomas D Nadeau and Ken Gray. *SDN: Software Defined Networks: An Authoritative Review of Network Programmability Technologies.* " O'Reilly Media, Inc.", 2013.
- [2] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-Defined Networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.
- [3] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to SDN: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98, 2014.
- [4] Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 1–12. ACM, 2007.
- [5] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.
- [6] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.
- [7] Evangelos Haleplidis, Jamal Hadi Salim, Joel M Halpern, Susan Hares, Kostas Pentikousis, Kentaro Ogawa, Weiming Wang, Spyros Denazis, and Odysseas Koufopavlou. Network programmability with ForCES. *IEEE Communications Surveys & Tutorials*, 17(3):1423–1440, 2015.
- [8] OME Committee et al. Software-defined networking: The new norm for networks. *Open Networking Foundation*, 2012.
- [9] Zhijing Qin, Grit Denker, Carlo Giannelli, Paolo Bellavista, and Nalini Venkatasubramanian. A software defined networking architecture for the internet-of-things. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9. IEEE, 2014.



- [10] Lei Zhou, Ligang Dong, and Rong Lin. Research on ForCES Configuration Management Based on NETCONF. *Information Technology Journal*, 13(5):904–911, 2014.
- [11] ONF. TS-016, OpenFlow Management and Configuration Protocol. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf>, 2014. [Online].
- [12] Chunyi Peng, Minkyong Kim, Zhe Zhang, and Hui Lei. VDN: Virtual machine image distribution network for cloud data centers. In *INFOCOM, 2012 Proceedings IEEE*, pages 181–189. IEEE, 2012.
- [13] Teemu Koponen, Keith Amidon, Peter Balland, Martín Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Paul Ingram, Ethan J Jackson, et al. Network Virtualization in Multi-tenant Datacenters. In *NSDI*, volume 14, pages 203–216, 2014.
- [14] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. FlowVisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep.*, 1:132, 2009.
- [15] Aditya Gudipati, Li Erran Li, and Sachin Katti. Radiovisor: A slicing plane for radio access networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 237–238. ACM, 2014.
- [16] Ali Al-Shabibi, Marc De Leenheer, Matteo Gerola, Ayaka Koshibe, Guru Parulkar, Elio Salvadori, and Bill Snow. OpenVirteX: Make your virtual SDNs programmable. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 25–30. ACM, 2014.
- [17] M Sune, A Köpsel, V Alvarez, and T Jungel. xDPd: eXtensible DataPath Daemon. *EWSDN, Berlin, Germany*, 2013.
- [18] Andreas Blenk, Arsany Basta, and Wolfgang Kellerer. HyperFlex: An SDN virtualization architecture with flexible hypervisor function allocation. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 397–405. IEEE, 2015.
- [19] Advait Dixit, Fang Hao, Sarit Mukherjee, TV Lakshman, and Ramana Kompella. Towards an elastic distributed SDN controller. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 7–12. ACM, 2013.
- [20] Dan Levin, Andreas Wundsam, Brandon Heller, Nikhil Handigol, and Anja Feldmann. Logically centralized?: state distribution trade-offs in software defined networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 1–6. ACM, 2012.

- [21] Yury Jimenez, Cristina Cervello-Pastor, and Aurelio J Garcia. On the controller placement for designing a distributed SDN control layer. In *Networking Conference, 2014 IFIP*, pages 1–9. IEEE, 2014.
- [22] David Erickson. The beacon openflow controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 13–18. ACM, 2013.
- [23] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, et al. Onix: A distributed control platform for large-scale production networks. In *OSDI*, volume 10, pages 1–6, 2010.
- [24] Amin Tootoonchian and Yashar Ganjali. HyperFlow: A distributed control plane for OpenFlow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, pages 3–3, 2010.
- [25] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O’Connor, Pavlin Radoslavov, William Snow, et al. ONOS: towards an open, distributed SDN OS. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6. ACM, 2014.
- [26] Jan Medved, Robert Varga, Anton Tkacik, and Ken Gray. Opendaylight: Towards a model-driven sdn controller architecture. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, pages 1–6. IEEE, 2014.
- [27] Fábio Botelho, Alysson Bessani, Fernando Ramos, and Paulo Ferreira. Smartlight: A practical fault-tolerant SDN controller. *arXiv preprint arXiv:1407.6062*, 2014.
- [28] Andreas Wundsam and Minlan Yu. NOSIX: A portable switch interface for the network operating system. Technical report, TR-12-013, Oct. 1, 2012, XP055120641,(URL: <https://www.icsi.berkeley.edu/pubs/techreports/ICSI—TR-12-013.pdf>) Section 3, Figure 2, 2012.
- [29] Blaise Barney. POSIX threads programming. *National Laboratory. Disponível em: <https://computing.llnl.gov/tutorials/pthreads/>* Acesso em, 5, 2009.
- [30] Andy Bierman, Martin Bjorklund, and Kent Watsen. RESTCONF protocol. 2017.
- [31] Kok-Kiong Yap, Te-Yuan Huang, Ben Dodson, Monica S Lam, and Nick McKeeown. Towards software-friendly networks. In *Proceedings of the first ACM asia-pacific workshop on Workshop on systems*, pages 49–54. ACM, 2010.

- [32] Joshua Reich, Christopher Monsanto, Nate Foster, Jennifer Rexford, and David Walker. Modular SDN Programming with Pyretic. *Technical Reprint of USENIX*, 2013.
- [33] Christopher Monsanto, Nate Foster, Rob Harrison, and David Walker. A compiler and run-time system for network programming languages. In *ACM SIGPLAN Notices*, volume 47, pages 217–230. ACM, 2012.
- [34] Cole Schlesinger, Alec Story, Stephen Gutz, Nate Foster, and David Walker. Splendid isolation: Language-based security for Software-Defined Networks. In *Proc. of Workshop on Hot Topics in Software Defined Networking*, 2012.
- [35] Daniel Turull, Markus Hidell, and Peter Sjödin. libNetVirt: the Network Virtualization Library. In *Communications (ICC), 2012 IEEE International Conference on*, pages 5543–5547. IEEE, 2012.
- [36] Andreas Voellmy, Hyojoon Kim, and Nick Feamster. Procera: a language for high-level reactive network control. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 43–48. ACM, 2012.
- [37] Fabian Schneider, Takashi Egawa, Sibylle Schaller, Shin-ichiro Hayano, Marcus Schöller, and Frank Zdarsky. Standardizations of SDN and its practical implementation. *NEC Technical Journal, Special Issue on SDN and Its Impact on Advanced ICT Systems*, 8(2), 2014.
- [38] Joel M Halpern. Standards collisions around SDN. *IEEE Communications Magazine*, 52(12):10–15, 2014.
- [39] David Meyer. The Software-Defined-Networking Research Group. *IEEE Internet Computing*, 17(6):84–87, 2013.
- [40] Attila Csoma, Balázs Sonkoly, Levente Csikor, Felicián Németh, Andràs Gulyas, Wouter Tavernier, and Sahel Sahhaf. Escape: Extensible service chain prototyping environment using mininet, click, netconf and pox. *ACM SIGCOMM Computer Communication Review*, 44(4):125–126, 2015.
- [41] Antonio Felix, Nuno Borges, H Wu, Michael Hanlon, Martin Birk, and Alexander Tschersich. Multi-layer sdn on a commercial network control platform for packet optical networks. In *Optical Fiber Communications Conference and Exhibition (OFC), 2014*, pages 1–3. IEEE, 2014.
- [42] Ying-Dar Lin, Dan Pitt, David Hausheer, Erica Johnson, and Yi-Bing Lin. Software-defined networking: Standardization for cloud computing’s second wave. *Computer*, 47(11):19–21, 2014.
- [43] Christian Esteve Rothenberg, Roy Chua, Josh Bailey, Martin Winter, Carlos NA Corrêa, Sidney C de Lucena, Marcos Rogério Salvador, and Thomas D Nadeau. When open source meets network control planes. *Computer*, 47(11):46–54, 2014.

- [44] ONF. Technical Communities Overview. <https://www.opennetworking.org/technical-communities>, 2017. [Online].
- [45] ONF. TR-539, OpenFlow Controller Benchmarking Methodologies, version 1.0. [https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-539\\_OpenFlow\\_Controller\\_Benchmarking\\_Methodologies\\_v1.pdf](https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-539_OpenFlow_Controller_Benchmarking_Methodologies_v1.pdf).
- [46] ONF. Wireless Transport SDN Proof of Concept White Paper. [https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/ONF\\_Microwave\\_SDN\\_PoC\\_White\\_Paper%20v1.0.pdf](https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/ONF_Microwave_SDN_PoC_White_Paper%20v1.0.pdf), Sep 2015. [Online].
- [47] ONF. Wireless Transport SDN Proof of Concept 2 Detailed Report. [https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Wireless\\_Transport\\_SDN\\_PoC\\_White\\_Paper.pdf](https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Wireless_Transport_SDN_PoC_White_Paper.pdf), Jun 2016. [Online].
- [48] ONF. Third Wireless Transport SDN Proof of Concept White Paper. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Third-Wireless-Transport-SDN-Proof-of-Concept-White-Paper.pdf>, Dec 2016. [Online].
- [49] ONF. Fourth Wireless Transport SDN Proof of Concept White Paper. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Third-Wireless-Transport-SDN-Proof-of-Concept-White-Paper.pdf>, Aug 2017. [Online].
- [50] Avri Doria, J Hadi Salim, Robert Haas, Horzmud Khosravi, Weiming Wang, Ligang Dong, Ram Gopal, and Joel Halpern. Forwarding and control element separation (ForCES) protocol specification. Technical report, 2010.
- [51] Rodolfo Alvizu, Guido Maier, Navin Kukreja, Achille Pattavina, Roberto Morro, Alessandro Capello, and Carlo Cavazzoni. Comprehensive survey on T-SDN: Software-defined Networking for Transport Networks. *IEEE Communications Surveys & Tutorials*, 2017.
- [52] Paul Goransson, Chuck Black, and Timothy Culver. *Software Defined Networks: A Comprehensive Approach*. Morgan Kaufmann, 2016.
- [53] ONF. OpenFlow-Enabled Cloud Backbone Networks Create Global Provider Data Centers. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-cloud-backbone-networks.pdf>, Nov 2012. [Online].

- [54] ONF. SDN Security Considerations in the Data Center. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-security-data-center.pdf>, Oct 2013. [Online].
- [55] Jiaqiang Liu, Yong Li, and Depeng Jin. SDN-based live VM migration across datacenters. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 583–584. ACM, 2014.
- [56] Raul Muñoz, Ricard Vilalta, Ramon Casellas, Ricardo Martinez, Thomas Szyrkowicz, Achim Autenrieth, Víctor López, and Diego López. Integrated SDN/NFV management and orchestration architecture for dynamic deployment of virtual SDN control instances for virtual tenant networks. *Journal of Optical Communications and Networking*, 7(11):B62–B70, 2015.
- [57] Google. Inter-Datacenter WAN with centralized TE using SDN and OpenFlow, White Paper. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/customer-case-studies/csgooglesdn.pdf>, 2012. [Online].
- [58] Stefano Vissicchio, Laurent Vanbever, and Olivier Bonaventure. Opportunities and Research Challenges of Hybrid Software Defined Networks. *ACM SIGCOMM Computer Communication Review*, 44(2):70–75, 2014.
- [59] David Ke Hong, Yadi Ma, Sujata Banerjee, and Z Morley Mao. Incremental deployment of SDN in hybrid enterprise and ISP networks. In *Proceedings of the Symposium on SDN Research*, page 1. ACM, 2016.
- [60] Dan Levin, Marco Canini, Stefan Schmid, Anja Feldmann, et al. Toward transitional SDN deployment in Enterprise Networks. *Proceedings of the Open Networking Summit (ONS)*, 2013.
- [61] Marco Canini, Anja Feldmann, Dan Levin, Fabian Schaffert, and Stefan Schmid. Panopticon: Incremental Deployment of Software-Defined Networking. In *ACM Symposium on SDN Research*, 2016.
- [62] Cheng Jin, Cristian Lumezanu, Qiang Xu, Zhi-Li Zhang, and Guofei Jiang. Telekinesis: Controlling Legacy Switch Routing with OpenFlow in Hybrid Networks. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 20. ACM, 2015.
- [63] Marcel Caria, Tamal Das, and Admela Jukan. Divide and conquer: Partitioning OSPF networks with SDN. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 467–474. IEEE, 2015.
- [64] Marcel Caria and Admela Jukan. Link Capacity Planning for Fault Tolerant Operation in Hybrid SDN/OSPF Networks. In *Global Communications Conference (GLOBECOM), 2016 IEEE*, pages 1–6. IEEE, 2016.

- [65] NEC. Kanazawa University Hospital, Case Study. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/customer-case-studies/cs-nec.pdf>, 2012. [Online].
- [66] Sarvesh Bidkar, Ashwin Gumaste, Puneet Ghodasara, Saurabh Hote, Anirudha Kushwaha, Geetha Patil, Shivprasad Sonnis, Rishav Ambasta, Braja Nayak, and Peeyush Agrawal. Field trial of a software defined network (SDN) using carrier ethernet and segment routing in a tier-1 provider. In *Global Communications Conference (GLOBECOM), 2014 IEEE*, pages 2166–2172. IEEE, 2014.
- [67] György Kálmán. Applicability of Software Defined Networking in Industrial Ethernet. In *Telecommunications Forum Telfor (TELFOR), 2014 22nd*, pages 340–343. IEEE, 2014.
- [68] Masayoshi Kobayashi, Srini Seetharaman, Guru Parulkar, Guido Appenzeller, Joseph Little, Johan Van Reijendam, Paul Weissmann, and Nick McKeown. Maturing of OpenFlow and Software-defined Networking through deployments. *Computer Networks*, 61:151–175, 2014.
- [69] ONF. TR-512, Core Information Model (CoreModel), version 1.0. [https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Core\\_Information\\_Model\\_V1.0.pdf](https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Core_Information_Model_V1.0.pdf), Mar 2015. [Online].
- [70] ONF. TR-512, Core Information Model (CoreModel), version 1.1. [https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/ONF-CIM\\_Core\\_Model\\_base\\_document\\_1.1.pdf](https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/ONF-CIM_Core_Model_base_document_1.1.pdf), Nov 2015. [Online].
- [71] ONF. TR-512.1, Core Information Model (CoreModel), version 1.2. [https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-512\\_CIM\\_\(CoreModel\)\\_1.2.zip](https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-512_CIM_(CoreModel)_1.2.zip), Sep 2016. [Online].
- [72] ONF. TR-532, Microwave Information Model, version 1.0. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-532-Microwave-Information-Model-V1.pdf>, Dec 2016. [Online].
- [73] IETF. NETCONF Configuration Protocol [RFC 4741]. <https://tools.ietf.org/html/rfc4741>, Dec 2006. [Online].
- [74] IETF. Network Configuration Protocol (NETCONF) [RFC 6241]. <https://tools.ietf.org/html/rfc6241>, Jun 2011. [Online].
- [75] James Yu and Imad Al Ajarmeh. An empirical study of the NETCONF protocol. In *Networking and Services (ICNS), 2010 Sixth International Conference on*, pages 253–258. IEEE, 2010.

- [76] Khalid Elbadawi and James Yu. Improving network services configuration management. In *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*, pages 1–6. IEEE, 2011.
- [77] IETF. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF) [RFC 6020]. <https://tools.ietf.org/html/rfc6020>, Oct 2010. [Online].
- [78] Huang Ji, Bin Zhang, Guohui Li, Xuesong Gao, and Yan Li. Challenges to the new network management protocol: NETCONF. In *Education Technology and Computer Science, 2009. ETCS'09. First International Workshop on*, volume 1, pages 832–836. IEEE, 2009.
- [79] Emmanuel Nataf and Olivier Festor. End-to-end YANG-based configuration management. In *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pages 674–684. IEEE, 2010.
- [80] Huiyang Cui, Bin Zhang, Guohui Li, Xuesong Gao, and Yan Li. Contrast analysis of NETCONF modeling languages: XML Schema, Relax NG and YANG. In *Communication Software and Networks, 2009. ICCSN'09. International Conference on*, pages 322–326. IEEE, 2009.
- [81] ONF. TR-531, UML to YANG Mapping Guidelines, version 1.0. [https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-531\\_UML-YANG\\_Mapping\\_Guidelines\\_v1.0.pdf](https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-531_UML-YANG_Mapping_Guidelines_v1.0.pdf), Sep 2016. [Online].
- [82] NETCONF Working Group. NETCONF Implementations, Wiki. <https://trac.ietf.org/trac/netconf/wiki>, 2017. [Online].
- [83] Alexandru Stancu, Alexandru Vulpe, George Suci, and Eduard Popovici. Comparison between several open source Network Configuration protocol Server implementations. In *Communications (COMM), 2016 International Conference on*, pages 173–176. IEEE, 2016.
- [84] Radek Krejci. Building NETCONF-enabled network management systems with libnetconf. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 756–759. IEEE, 2013.
- [85] OpenDaylight. OpenDaylight Controller: Netconf Testtool. [https://wiki.opendaylight.org/view/OpenDaylight\\_Controller:Netconf:Testtool](https://wiki.opendaylight.org/view/OpenDaylight_Controller:Netconf:Testtool), 2017. [Online].
- [86] Dudu Bercovich, Luis M Contreras, Yoram Haddad, Ariel Adam, and Carlos J Bernardos. Software-Defined Wireless Transport Networks for Flexible Mobile Backhaul in 5G Systems. *Mobile Networks and Applications*, 20(6):793–801, 2015.

- [87] Carlos J Bernardos, Antonio De La Oliva, Pablo Serrano, Albert Banchs, Luis M Contreras, Hao Jin, and Juan Carlos Zúñiga. An Architecture for Software Defined Wireless Networking. *IEEE wireless communications*, 21(3):52–61, 2014.
- [88] Alexandru Stancu, Alexandru Vulpe, Octavian Fratu, and Simona Halunga. Default Values Mediator used for a Wireless Transport SDN Proof of Concept. In *Standards for Communications and Networking (CSCN), 2016 IEEE Conference on*, pages 1–6. IEEE, 2016.
- [89] ONF CENTENNIAL. DVM (Default Values Mediator), version 1. <https://github.com/OpenNetworkingFoundation/CENTENNIAL/tree/onf-otwg-wtp-3rd-poc/02-MWTN-PoC/code/mediator>, 2016. [Online].
- [90] Alexandru Stancu, Alexandru Avram, Martin Skorupski, Alexandru Vulpe, and Simona Halunga. Enabling SDN Application Development Using a NETCONF Mediator Layer Simulator. In *Ubiquitous and Future Networks (ICUFN), 2017 Ninth International Conference on*, pages 658–663. IEEE, 2017.
- [91] ONF CENTENNIAL. DVM (Default Values Mediator), version 2. [https://github.com/OpenNetworkingFoundation/CENTENNIAL/tree/onf-otwg-wtp-3rd-poc/03-WTP-PoC/code/Default\\_Values\\_Mediator](https://github.com/OpenNetworkingFoundation/CENTENNIAL/tree/onf-otwg-wtp-3rd-poc/03-WTP-PoC/code/Default_Values_Mediator), 2016. [Online].
- [92] Guru Parulkar, Tom Tofigh, and Marc De Leenheer. SDN control of packet-over-optical networks. In *Optical Fiber Communication Conference*, pages W1G–4. Optical Society of America, 2015.
- [93] Aristotelis Kretsis, Loris Corazza, Kostas Christodoulopoulos, Panagiotis Kokkinos, and Emmanouel Varvarigos. An emulation environment for SDN enabled flexible IP/optical networks. In *Transparent Optical Networks (ICTON), 2016 18th International Conference on*, pages 1–4. IEEE, 2016.
- [94] Victor Mehmeri, Xi Wang, Shrutarshi Basu, Qiong Zhang, Paparao Palacharla, Tadashi Ikeuchi, Idelfonso Tafur Monroy, Juan Jose Vegas Olmos, and Nate Foster. Software-programmed optical networking with integrated NFV service provisioning. In *Optical Fiber Communication Conference*, pages Tu3L–12. Optical Society of America, 2017.
- [95] Alexandru Stancu, Alexandru Vulpe, Octavian Fratu, and Simona Halunga. Wireless Transport Emulator Based on LINC Software Switch. *Wireless Personal Communications*, pages 1–17, 2017.
- [96] FlowForwarding. LINC - OpenFlow software switch. <https://github.com/FlowForwarding/LINC-Switch>, 2015. [Online].
- [97] FlowForwarding. Quick start guide - LINC OpenFlow switch. [https://github.com/FlowForwarding/LINC\\_Switch/blob/master/docs/LINC\\_Switch\\_Quick\\_Start\\_Guide.pdf](https://github.com/FlowForwarding/LINC_Switch/blob/master/docs/LINC_Switch_Quick_Start_Guide.pdf), 2015. [Online].



- [98] Alexandru Stancu. LINC-WE: Wireless Emulator based on LINC software switch. <https://github.com/Melacon/LINC-WE>, 2017. [Online].
- [99] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.
- [100] Ryan Chamberlain and Jennifer Schommer. Using Docker to support reproducible research. DOI: <https://doi.org/10.6084/m9.figshare.1101910>, 2014.
- [101] Alexandru Stancu. Wireless Transport Emulator. <https://github.com/Melacon/WirelessTransportEmulator>, 2017. [Online].
- [102] George Kizer. *Digital Microwave Communication: Engineering Point-to-Point Microwave Systems*. John Wiley & Sons, 2013.
- [103] iPerf - The ultimate speed test tool for TCP, UDP and SCTP. <https://iperf.fr/>, 2017. [Online].
- [104] Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, and Kevin Gibbs. Iperf: The TCP/UDP bandwidth measurement tool. <http://dast.nlanr.net/Projects>, 2005.