UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels

TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Disseny, construcció i avaluació d'una xarxa mesh wireless gestionada per software (SDW)

TITULACIÓ: Grau en enginyeria Telemàtica

AUTORS: Anna Hurtado Borràs
Jordi Palà Solé

DIRECTOR: Sebastià Sallent Ribes

SUPERVISOR: Daniel Camps Mur

DATA: 17 de novembre de 2014

Títol: Disseny, construcció i avaluació d'una xarxa mesh wireless gestionada per software (SDW)

Autors: Anna Hurtado Borràs
Jordi Palà Solé

Director: Sebastià Sallent Ribes

Supervisor: Daniel Camps Mur

Data: 17 de novembre de 2014

Resum

Software Defined Wireless Backhaul for Small Cells (SESAME), el nostre projecte, pretén donar resposta a la imminent saturació de les xarxes mòbils a través de la introducció de xarxes denses formades per *small cells* controlades per software, que permetrien assumir el creixent tràfic de dades. Fent ús de les *small cells*, incrementem la capacitat reduint la mida de la cel·la i densificant les xarxes d'accés. No obstant, un desplegament massiu de *small cells* planteja obstacles tècnics importants a les arquitectures de xarxa actuals. En concret, "*backhauling outdoor small cells*" que poden ésser muntats en fanals o mobiliari urbà, és un problema difícil que ha de ser abordat amb tecnologies *wireless* eficients.

Per tant, el propòsit és dissenyar, implementar i avaluar un sistema que permet a un controlador de SDN (*Software Defined Network*) controlar el pla d'encaminament en una xarxa *Wireless* que està connectada a una xarxa cablejada –o *core*- per diferents punts físics(*Wireless Backhaul Network*).

L'objectiu principal és la millora de la qualitat de servei oferta als usuaris mitjançant l'ús de la tecnologia SDN. S'introdueix la funcionalitat d'un algoritme d'encaminament propi, que permet la gestió de les rutes de sortida d'una xarxa *wireless backhaul*, en funció de les condicions dels enllaços ràdio. Per tal d'implementar aquest control de manera efectiva, cal que el controlador sigui conscient d'aquestes condicions.

Demostrarem la nostra arquitectura en dos entorns de proves diferents, on analitzarem els costos implicats en el processat addicional i la centralització imposada per l'SDN, mentre motivem els beneficis de la nostra arquitectura en termes de millora de la gestió i control de la xarxa.

Per tal de construir el nostre entorn, hem utilitzat: Open vSwitch per poder gestionar els nodes –*switch*-, mitjançant el controlador OpenDaylight. El protocol utilitzat per a l'intercanvi de missatges és OpenFlow.

La solució proposada s'ha simulat sota un entorn Linux degut a la flexibilitat que ens proporciona en quant a l'ús del hardware. S'ha emulat una xarxa *wireless* per tal de poder desenvolupar la nostra arquitectura i així, realitzar les proves pertinents per a verificar el seu rendiment en comparació amb l'extensió *mesh* del protocol 802.11.

Title : Design, build and evaluate a software defined wireless backhaul for small cells

Authors: Anna Hurtado Borràs
Jordi Palà Solé

Advisor: Sebastià Sallent Ribes

Supervisor: Daniel Camps Mur

Date: November 17, 2014

Overview

Software Defined Wireless Backhaul for Small Cells (SESAME), our project, pretends to give an answer to mobile network saturation through the introduction of dense networks formed by software-controlled small cells to address the forecasted exponential traffic increase in future Mobile networks. Using small cells we increase capacity by means of reducing cell size and densifying access networks. However, a massive deployment of this technology poses significant technical hurdles to current network architectures. In particular, backhauling outdoor Small Cells, which may be mounted on lamp posts or street furniture, is a challenging problem that needs to be addressed with efficient wireless technologies.

The main goal is to improve the quality of service offered to the users through the use of SDN technology. We developed a custom forwarding algorithm which allows us the management of the routing of a wireless backhaul network in function of the radio link conditions. In order to implement this control effectively, the controller must be aware of radio conditions.

We demonstrate our architecture in two different testbed prototypes, where we analyze the costs incurred in the additional processing and the centralization of state imposed by SDN, while motivating the benefits of our architecture in terms of improved network management and control.

In order to build our environment we used the following technologies: Open vSwitch to manage the nodes –switchs - in the network controlled by OpenDaylight. The protocol used to exchange the messages is Openflow.

Our solution has been simulated under a Linux environment due to the flexibility that it provides in terms of hardware implementation. We have emulated a wireless network in order to develop our architecture and perform appropriate tests to verify its performance in comparison with the mesh extension of 802.11 protocol.

ÍNDEX

ÍNDEX DE FIGURES

ÍNDEX DE TAULES

INTRODUCCIÓ

La xarxa d'Internet evoluciona a conseqüència d'un canvi en el seu ús. Els protocols van ser dissenyats per a que fossin robustos i escalables en un entorn on el control de la xarxa és rígid i poc dinàmic. Actualment hi ha una necessitat de flexibilitat en l'administració de les xarxes per a fer un ús òptim dels dispositius degut a la gran demanda de serveis que hi ha avui en dia.

El creixement del tràfic previst en les xarxes mòbils és un dels reptes de la indústria mòbil. Densificar les xarxes mòbils mitjançant *small cells* que tenen un rang limitat però altes taxes de transmissió de dades, sembla ser la manera més prometedora de fer front a aquesta creixent demanda de capacitat.

Una manera de gestionar les *small cells* és a través del paradigma de SDN (*Software-Defined-Network*), en el qual la majoria de serveis es converteixen en *software*. Aquesta tecnologia promet aportar una gran flexibilitat, agilitat i eficiència a la infraestructura de la xarxa, simplificant i centralitzant el control dels recursos i les polítiques de la xarxa. El primer punt d'aquest paradigma és que totes les funcionalitats són *software*. El segon, marca una clara tendència a gestionar la xarxa de manera centralitzada, mentre que el tercer punt diu que aquesta concentració dóna més poder a l'usuari. Per tant, s'ha d'optimitzar l'ús del *hardware* físic a través d'un controlador centralitzat on es troba tota la informació necessària per dur a terme la gestió de la xarxa. D'aquesta manera un servei pot fer una demanda en temps real de recursos de xarxa, en funció de la seva necessitat, aconseguint que utilitzi el *hardware* que realment necessita. En el model tradicional de xarxa, possiblement aquesta assignació de recursos estaria sobredimensionada.

SDN és una tecnologia emergent i en constant desenvolupament. L'arquitectura es defineix en tres nivells: infraestructura, control i aplicació. OpenFlow és un dels elements més importants ja que és el primer estàndard de comunicació entre el pla de dades (encaminament), i el pla de control. És el protocol que fa possible la comunicació del controlador amb els diferents dispositius de la xarxa, ja que uneix els dos primers nivells del SDN. Hi ha un estàndard *de facto open source* establert per al controlador. N'hi ha més d'un en desenvolupament però la funció general de tots és la mateixa. La única diferencia entre ells és el llenguatge en el qual estan implementats com ara phyton, java,... En el nostre cas utilitzarem OpenDaylight ja que està implementat en java i és un dels més utilitzats.

SESAME, el nostre projecte, pretén donar resposta a la imminent saturació de les xarxes mòbils a través de la introducció de xarxes denses formades per *small cells* controlades per software, que permetrien assumir el creixent tràfic de dades.

En particular, en aquest projecte presentem una nova arquitectura basada en SDN per "*Wireless Small Cell Backhaul*", que permet reutilitzar fàcilment els controladors SDN disponibles per controlar els nodes sense fils. Minimitzant la càrrega de gestió per configurar-los i extenent el protocol OpenFlow per tal de poder transportar els paràmetres *wireless*. A més, es descriu la implementació d'un prototip i l'avaluació dels beneficis potencials de la nostra arquitectura, utilitzant unes proves de simulació virtual i unes proves en entorn físic.

Aquest projecte està organitzat de la manera següent. Capítol 1, introdueix la motivació i els orígens del projecte, avaluant la situació i la necessitat actual. Capítol 2, descriu la nostra arquitectura SDN. Capítol 3, descriu el prototip implementat i els resultats de

rendiment obtinguts. Finalment, es resumeixen les principals conclusions a les que s'ha arribat amb el projecte.

Aquesta innovació donaria una possible solució a problemes amb futures implementacions de noves tecnologies com el 5G. Pretén també la utilització de qualsevol dispositiu, independentment del fabricant, i de si disposa d'un protocol propietari que el faci compatible amb la tecnologia SDN.

CAPÍTOL 1. MOTIVACIÓ I ORIGEN DE SESAME

1.1. Situació actual: Problema, necessitat i evolució

SESAME és un projecte que ha nascut a partir de la gran demanda actual de tràfic de dades en les xarxes mòbils. En el nostre entorn, aquests últims anys s'ha pogut observar com les tecnologies mòbils s'han anat integrant en la nostra vida quotidiana. En la figura 1.1, podem veure el creixement d'usuaris amb dispositius mòbils que disposen d'accés a Internet en els últims anys.

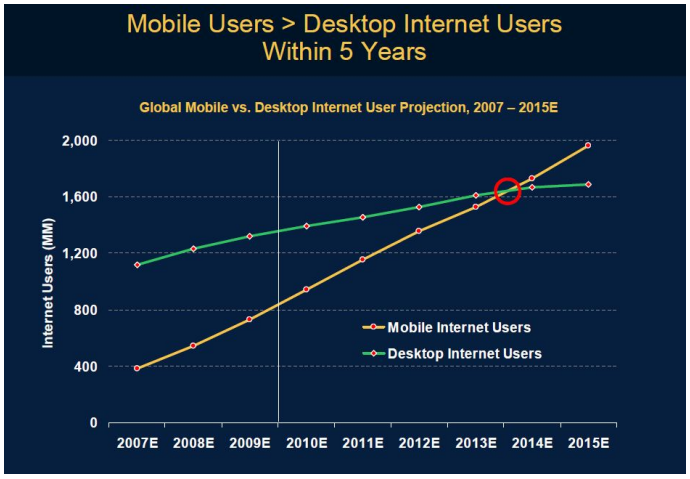

Figura 1.1: Usuaris d'Internet mòbils vs. usuaris d'Internet de sobretaula

Aquesta evolució tecnològica ha esdevingut gràcies a la comoditat de tenir a l'abast tota la informació personal i d'interès de l'usuari al instant: contactes, calendaris sincronitzats, missatgeria instantània, connexió amb xarxes socials, jocs... Són uns dels molts avantatges que ens ofereix tenir un *smartphone*. Fins ara s'ha pogut suportar el tràfic que generaven aquests dispositius, però la previsió de l'augment d'aquests està deixant enrere els protocols actuals, ja que no poden gestionar un tràfic tant dinàmic i de tanta densitat de dispositius.

Per altra banda en els últims anys, s'estan implementant sensors de tot tipus a les grans ciutats, per tenir un control total de la infraestructura urbana i d'aquesta manera, optimitzar els seus recursos. Aquestes ciutats reben el nom de *Smart Cities*. Tots aquests dispositius han de ser gestionats per tal d'obtenir la seva informació i controlar-los remotament des d'altres màquines per poder realitzar-hi accions. Aquestes connexions, o dispositius, es classifiquen amb el nom de M2M (*Machine-to-Machine*).

Com podem veure a la Figura 1.2, donat al gran nombre de sensors que s'estan implementant i l'augment d'ús de dispositius mòbils, segons l'estudi de [1], preveu un creixement molt gran de dispositius intel·ligents i M2M connectats a la xarxa d'Internet fins al 2018.

Aquests grups de dispositius ubicats a la ciutat que la fan intel·ligent, està portant a tenir una visió diferent d'Internet anomenat Internet de les Coses. Aquests tant poden estar connectats per cable com per una infraestructura *wireless* dins de la ciutat. Però aquest gran nombre de dispositius, ja siguin M2M o dispositius mòbils d'usuaris, impliquen un

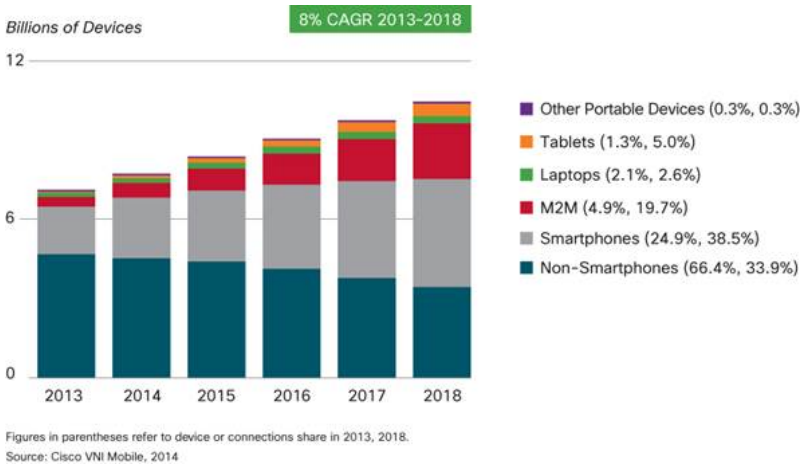

Figura 1.2: Creixement de dispositius mòbils i connexions fins al 2018

control més exhaustiu de la xarxa que amb els protocols actuals no podem assolir. Una de les possibles solucions que es proposen a aquest problema, és utilitzar la tecnologia SDN (*Software-Defined Network*), per poder controlar eficientment el gran nombre de dispositius i gestionar el tràfic dinàmic que generen.

Per altra banda, en quant a dispositius *wireless*, les tecnologies ràdio també s'han vist obligades a evolucionar en funció del gran nombre de dispositius que s'estan implementant, per tal de poder acollir tot aquest excés d'informació a nivell de hardware. En els 10 últims anys s'ha pogut observar com ha evolucionat de manera vertiginosa. S'ha vist una lluita constant entre les operadores mòbils per oferir tecnologies d'última generació, ja que és un motor econòmic.

El nostre projecte pretén donar una solució als problemes anteriors, oferint una convergència de l'arquitectura de la xarxa física amb la mòbil, mentre optimizem la gestió de recursos i serveis de la infraestructura. Donat que són tecnologies pioneres, a més de solucionar els problemes actuals, oferiran noves funcionalitats que permetran evolucionar en l'eficiència de la xarxa entre d'altres canvis.

SESAME fa possible l'encaminament de dades en una "*xarxa wireless backhaul*" per diferents punts físics, en funció de les condicions dels enllaços ràdio dels nodes, amb l'objectiu d'optimitzar la sortida cap a la xarxa *core* oferint rutes amb la millor qualitat possible. S'ha dissenyat un algoritme capaç de detectar canvis d'estats en els enllaços *wireless* (inalàmbrics), i calcular una ruta a partir d'aquestes estadístiques. Els nodes podran estar ubicats en les faroles d'una ciutat o mobiliari urbà similar, i oferiran una connexió de gran capacitat als ciutadans mitjançant la tecnologia SDN (*Software-Defined Network*) i les *Small Cells*.

Cal tenir en compte que les freqüències utilitzades per a la xarxa d'accés i la xarxa *wireless backhaul* són les mateixes, 2.4 Ghz, per simplicitat de muntatge. En un entorn real, les dues xarxes haurien d'estar en freqüències diferents per evitar interferències en el medi, d'aquesta manera, si la *xarxa backhaul* operés a 5 Ghz no perjudicaria el medi de la xarxa d'accés.

1.2. Tecnologies actuals

Un cop avaluada la situació actual de les xarxes i la demanda d'una nova arquitectura, presentarem les tecnologies actuals en les quals ens basarem per a construir-la:

- Small cells
- Software Defined Network (SDN)
- El protocol OpenFlow

1.2.1. Small Cells

La tecnologia *Small Cell* és un tipus de node d'accés ràdio de baixa potencia que opera en espectres amb llicència o sense, amb un rang de 10 metres a 1 o 2 quilòmetres. Són "petites" en comparació a les macrocel·les que poden tenir un rang de desenes de quilòmetres. En aquest concepte s'engloba a les femtocel·les, picocel·les i microcel·les.

Les *Small Cells* poden ser utilitzades per oferir un servei *wireless* tant a dins d'un edifici com a l'exterior. Actualment les utilitzen els operadors de telefonia mòbil per estendre la cobertura del seu servei i/o per augmentar la capacitat de l'enllaç.

Hi ha diverses maneres d'augmentar la capacitat: fent més eficient les modulacions, els protocols MAC, els codificadors de veu...Però, sense dubte, al llarg de la historia la tècnica que més s'ha utilitzat per augmentar la capacitat ha estat el reús espaial.

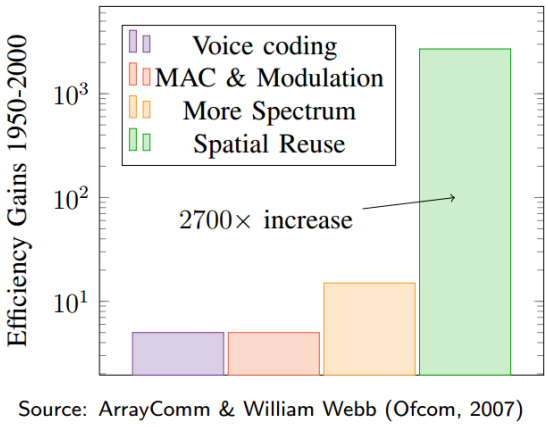

Figura 1.3: Factors que han contribuït a l'augment de la capacitat i l'eficiència

A través de la introducció de xarxes denses formades per *small cells* controlades per *software*, es permetria l'absorció del creixent tràfic de dades en xarxes mòbils. Fent ús de les *small cells*, incrementem la capacitat reduint la mida de la cel·la i densificant les xarxes d'accés. No obstant això, un desplegament massiu de *small cells* planteja obstacles tècnics importants a les arquitectures de xarxa actuals. En concret, "*Backhauling outdoor small cells*", que poden ésser muntats en fanals o mobiliari urbà similar, és un problema difícil que ha de ser abordat amb tecnologies *wireless* eficients. La Figura 1.4 il·lustra

un exemple de desplegament dens de *small cells*, on s'utilitzen unitats de transport *wireless* entre *small cells* per tal d'encaminar el tràfic salt a salt, fins a arribar a una unitat de transport cablejada connectada a la xarxa *core*.

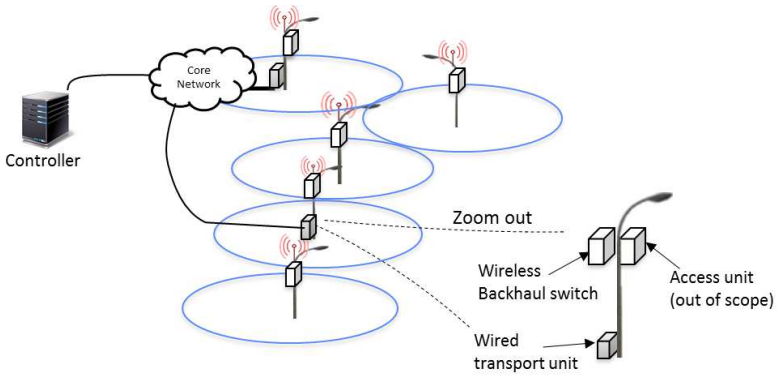

Figura 1.4: Desplegament exterior de *Small Cells* amb *wireless backhauling*

Actualment hi ha empreses que estan apostant fort per la investigació i l'ús d'aquest tipus de cel·les. Alguns fabricants com [2] ja estan promovent l'ús de les *Small Cells* per poder ampliar la capacitat dels enllaços i combinar totes les tecnologies que tenim avui en dia: 3G, 4G, Wi-Fi. Proposen ampliar la capacitat dels enllaços mil vegades més combinant:

- Més bandes en l'espectre radioelèctric.
- L'ús de moltes *Small Cells* per tal de cobrir l'àrea exterior d'una ciutat.
- Fer "*Inside-Out deployments*", degut a que les dades dels dispositius mòbils normalment son generades dins d'edificis.

Per altra banda, diferents fabricants s'han unit per poder oferir un producte innovador. Una solució al problema de densificació de xarxes amb un impacte visual mínim. El projecte s'anomena *Zero Site* [3] i, consisteix en una farola modificada de tal manera que integra a la part superior una unitat ràdio, que ofereix cobertura amb *Small Cells*. A més a més, les llums són de tecnologia LED per promoure l'estalvi d'energia.

Veiem que grans empreses estant invertint en projectes per desenvolupar en aquesta tecnologia, tot i que encara és emergent, provocant certa confiança en el seu ús i per tant, que més empreses decideixin apostar per ella.

Un altre aspecte important del desplegament de *Small Cells*, és l'augment dels fluxos de tràfic imprevisibles. Abans els models de tràfic, des d'un punt de vista estadístic, eren més deterministes ja que només existien les connexions de telefonia i de dades, ara bé, això ha patit un canvi degut a que ara existeixen més tipus de connexions, tornant aquest patró de tràfic més imprevisible. Concretament, reduint la mida de les cel·les resulta en un increment del tràfic, com menys terminals actius de manera simultània hi ha en una cel·la donada. Per tant, caldria un control més estricte de recursos de la xarxa, que aconseguim mitjançant el SDN.

1.2.2. Software Defined Network (SDN)

L'imminent creixement de dispositius mòbils, el tractament dels continguts, l'ús de la virtualització de servidors i l'ús de serveis *Cloud*, han portat a que la indústria de la xarxa analitzi les arquitectures tradicionals.

Moltes xarxes convencionals tenen dissenys jeràrquics. Aquest disseny és eficient sempre i quan la comunicació dominant sigui client-servidor. Però una arquitectura tant estàtica i distribuïda, és poc adequada per a les necessitats tant dinàmiques que requereixen tots aquests serveis. Algunes de les tendències clau que impulsen la necessitat d'un nou paradigma de la xarxa són:

- **Canvi de patrons de tràfic**: els serveis actuals generen molt de tràfic punt a punt com per exemple P2P (*Peer to Peer*) i M2M (*Machine-to-machine*), en lloc d'haver-hi la típica connexió client-servidor. Això fa que hi hagi més tràfic transversal, que va en contra d'una arquitectura jeràrquica.
- **Consum de les tecnologies de la informació**: cada vegada més usuaris utilitzen els seus dispositius personals com *smartphones* i tablets per accedir a la xarxa. Això suposa una densificació de la xarxa que ha de ser controlada eficientment.
- **El creixement dels serveis *cloud***: els serveis *cloud* impliquen una connexió permanent amb els dispositius creant tràfic molt dinàmic.
- **"*Big Data*" significa més ample de banda**: per abarcar tota la informació massiva del *cloud* i d'altres serveis, han d'haver-hi mils de servidors en paral·lel processant aquesta informació, necessitant estar connectats entre ells.

La tecnologia *Software-Defined Network* (SDN) pretén donar solució a totes les necessitats anteriors. És una arquitectura de xarxa emergent on el pla de control està separat del pla de dades, és a dir, intenta extreure totes les entitats cap a *software*. Aquesta migració del control, permet fer una abstracció de la capa d'infraestructura per a les aplicacions i serveis de xarxa, permetent tractar els elements de la xarxa com una entitat lògica o virtual. En la Figura 1.5 podem veure el model lògic que te una arquitectura SDN.

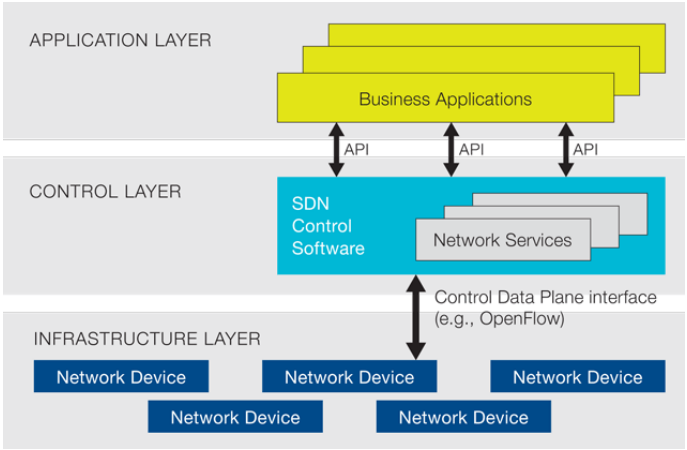

Figura 1.5: Arquitectura SDN

Com podem observar, la lògica de la xarxa està centralitzada mantenint una visió global de la xarxa. Degut a això, la xarxa esdevé per a les aplicacions i els serveis de polítiques, com un únic *switch* lògic que anirà dividint-se en base a la demanda d'aquesta capa d'aplicacions.

Aquestes funcionalitats de virtualització s'anomenen *Network Functions Virtualitzation* (NFV) i permeten un ús més eficient dels recursos de xarxa. Amb aquestes funcionalitats, la xarxa pren un dinamisme que pot anar canviant al mateix instant, depenent del patró de tràfic i dels recursos/serveis que s'ofereixen dinàmicament en funció dels usuaris (proveïdors, consumidors...).

Cal dir que SDN també ens permet dividir els recursos de la infraestructura de tal manera que, cada part pugui ser controlada per un controlador diferent. El FlowVisor s'encarrega de virtualitzar aquestes parts de la xarxa, fent que en qualsevol moment i d'una manera ràpida, es puguin assignar. D'aquesta manera, es fa flexible el traspàs de recursos d'una banda a l'altra.

Com a punts forts del SDN podem destacar que, les empreses i les operadores surten beneficiades degut a que un dels avantatges és el control de la xarxa independentment del proveïdor. El control es realitza en un punt lògic que simplifica el disseny i les operacions en les xarxes.

Per altra banda, també simplifica els dispositius de la xarxa perque ja no necessitaran entendre i processar milers de protocols diferents. Ara que les funcions de xarxa s'han portat al controlador, els dispositius només s'han d'encarregar de les taules d'encaminament.

Un dels altres beneficis que ens aporta és que, els operadors de xarxa i els administradors, poden configurar aquesta abstracció de la xarxa simplificada mitjançant programació, en lloc de configurar manualment milers de dispositius, amb milers de línies de codi diferents.

Actualment, la tecnologia SDN s'aplica en els següents entorns:

Empreses

- Campus: l'arquitectura SDN dóna suport a la convergència de dades, veu i vídeo, essent accessible des de qualsevol lloc i en qualsevol moment, complint les polítiques de manera coherent tant en infraestructura cablejada com *wireless*.
- *Data Centers*: facilita la virtualització de xarxes que ofereix una hiper-escalabilitat en el centre de dades. En un *data center* hi ha milers de servidors en paral·lel connectats entre ells gestionant tot aquest volum de dades.
- *Cloud*: Per facilitar la reserva de recursos de xarxa d'una manera extremadament elàstica, oferint una ràpida provisió de serveis.

Operadores i proveïdors de serveis

L'arquitectura SDN permet transformar la xarxa tradicional *backbone* en plataformes d'ofertes de serveis, ajudant als operadors i proveïdors a suportar la gran demanda dels usuaris. A més, gràcies a la virtualització el *multi-tenancy* es gestiona d'una manera més eficient.

1.2.3. Protocol OpenFlow

Openflow és un protocol de comunicacions emergent i obert dins de l'arquitectura del SDN que opera entre els dispositius de xarxa i la capa de control. Permet a un servidor de *software* (controlador), determinar l'encaminament que els paquets haurien de seguir en una xarxa de *switchs* (commutadors) o *routers*. Gràcies a ell, la xarxa pot estar gestionada amb una visió global i no individual com s'estava fent fins ara. Un del seus avantatges és la centralització de la informació de tota la xarxa en un únic punt, fent més eficient les decisions d'encaminament i, permetent tenir una visió holística de la xarxa. D'aquesta manera augmentem l'efectivitat amb els recursos físics de la xarxa.

Actualment hi ha diferents versions funcionals del protocol com el 1.0, 1.3, 1.4...Però a mida que els protocols s'han anat actualitzant, s'han anat afegint noves funcionalitats com ara:

- Múltiples taules de flux
- Tags i tunnels: MPLS, VLAN, virtual ports
- Encaminament *multipath*: ECMP, *groups*
- Noves extensions dels filtres *Match*
- Noves extensions de *Actions*
- IPv6
- Múltiples controladors
- Gestió d'elements de xarxa òptics

En el nostre projecte utilitzem la versió 1.0 ja que la versió de software Open vSwitch que utilitzem és la 2.1.2, que treballa amb la versió 1.0 del OpenFlow per defecte. No obstant, pot treballar amb la versió 1.3 però s'ha d'activar manualment. D'ara en endavant, les descripcions escrites sobre el protocol estaran basades en la versió 1.0.

El protocol té 3 grups de missatges diferents. Cada grup es composa de diferents missatges amb les seves funcions:

Controlador a *switch*

Aquest tipus de missatges són iniciats pel controlador i s'utilitzen per gestionar l'estat dels *switchs*. Els missatges d'aquest grup són:

- *Handshake*: establiment de la connexió entre switch i controlador.
- *Switch configuration*: permet al controlador enviar paràmetres de configuració als *switchs*.
- *Modify State Message*: són missatges per afegir o treure informació de les taules. Existeixen diferents tipus:
    - *Modify Flow Entry Message*: modifica informació de les taules de flux.

    - *Port Modification Message*: modifica el comportament dels ports físics.

- *Queue Configuraiton Message*: s'utilitza per consultar les cues dels ports físics.
- *Read State Message*: s'utilitza per demanar diferents estats del *switch*. Tenim els següents missatges:
    - *Description Statistics*: informació sobre el fabricant, revisió de *hardware* i *sofware* i número de sèrie.
    - *Individual flow Statistics*: informació d'estadístiques sobre un flux en concret.
    - *Agregate flow Statistics*: informació d'estadístiques sobre un conjunt de fluxos en concret.
    - *Table Statistics*: informació d'estadístiques sobre les taules.
    - *Port Statistics*: informació d'estadístiques sobre els ports físics.
    - *Queue Statistics*: informació d'estadístiques sobre les cues dels ports físics.
    - *Vendor Statistics*: informació d'estadístiques específiques del fabricant.
- *Packet-out Message*: s'envia en resposta d'un *Packet-in* del *switch*.
- *Barrier Message*: s'utilitza per a que el controlador rebi notificacions quan s'acabin operacions a les que s'ha subscrit anteriorment.

Asíncrons

Són iniciats pel *switch* per actualitzar la informació del controlador. Els missatges d'aquest grup són:

- *Packet-in Message*: s'envia cap al controlador quan en el *datapath* no hi ha cap instrucció per a aquell flux.
- *Flow Removed Message*: s'envia quan el controlador ha demanat que s'avisi quan un flux ha caducat en el *switch*.
- *Port Status Message*: s'envia quan un port ha estat afegit, modificat o eliminat del *datapath*.
- *Error Message*: s'envia per notificar errors del *switch*.

Simètrics

Iniciat tant pel controlador com pel *switch* sense sol·licitud. Els tipus de missatges són:

- *Hello*: s'utilitza per notificar la presencia d'un *switch*.
- *Echo Request*: s'envia amb un timestamp per a comprovar la latència, amb diferents longituds per a comporvar l'ample de banda o, sense res per comprovar la connexió amb el controlador.
- *Echo Reply*: s'envia en resposta a un Request amb el seu mateix contingut.

- *Vendor Message*: dissenyat per contenir informació extra d'un fabricant.

El protocol OpenFlow actualment està implementant en diversos dispositius de diferents arquitectures. En la Figura 1.6 podem veure alguns tipus tipus de sistemes en els quals treballa.

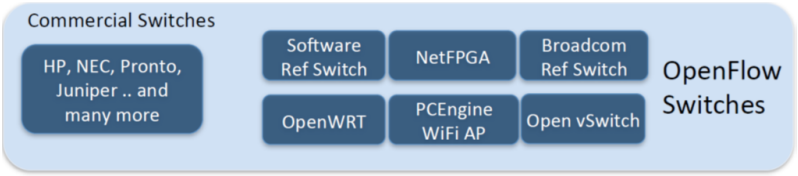

Figura 1.6: Switch comercials d'OpenFlow

En el nostre projecte treballarem amb una arquitectura X86, degut a que podem treballar amb qualsevol PC, facilitant-nos el desenvolupament i la instal·lació en els dispositius. Open vSwitch és el *switch* virtual que incorpora el protocol OpenFlow que utilizarem per muntar la nostra xarxa SDN. És un *software* dissenyat per l'arquitectura x86, allotjat als repositoris de Linux i en constant desenvolupament per la comunitat lliure.

La funcionalitat d'un *switch* OpenFlow és fer l'encaminament a través de les taules de flux generades pel controlador. El *switch* pot treballar amb les taules de flux de manera encadenada creant una *pipeline*.

Els elements per a realitzar aquest encaminament són els següents:

Flow Tables

Hi ha una acció (*Actions*) associada a cada entrada de la taula. Cada entrada és un tipus de flux que compleix una o un conjunt de condicions (*Header fields*). A més, cada entrada conté una sèrie de comptadors (*Counters*), amb els quals es podrà controlar l'estat d'aquest flux. Així doncs, el controlador pot determinar les decisions d'encaminament de manera més eficient, aplicar paràmetres QoS...

Header Fields	Counters	Actions	Priority
If ingress port == 2		Drop packet	32768
if IP_addr == 129.79.1.1		re-write to 10.0.1.1, forward port 3	32768

Figura 1.7: Taula de Fluxos del OpenFlow

Pipeline

És el tracte que rep un paquet al arribar a un *switch* OpenFlow. Quan arribar el paquet, aquest és comparat amb les condicions de les entrades de les *flow tables*. L'entrada que compleixi la condició, se li aplicarà l'acció associada. A mida que es van recorrent les diferents taules, se li apliquen les accions associades al paquet en qüestió "*on-the-fly*".

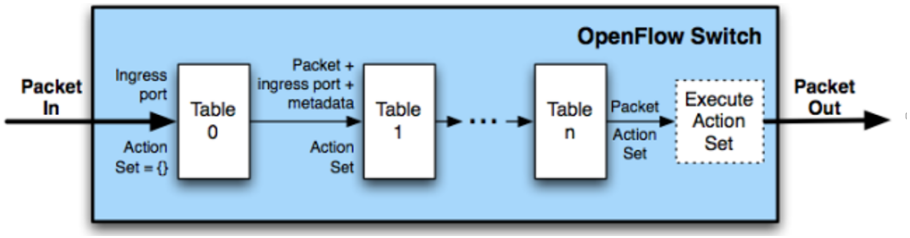

Figura 1.8: Pipeline del OpenFlow

1.3. Solució proposada

SESAME fa convergir la tecnologia de *Small Cells* amb el SDN, de manera que extreu els avantatges de cadascuna per tal de que una xarxa "*wireless backhaul*" instal·lada en una ciutat, pugui assumir totes les necessitats actuals.

Utilitzant les *Small Cells* densificarem la xarxa, per tant, cal un control més estricte dels recursos de la xarxa *wireless backhaul*. En lloc d'aprovisionament d'enllaços estàtics per a velocitats màximes com en les xarxes mòbils actuals, els enllaços han de ser monitoritzats i adaptats a la dinàmica de la xarxa. En aquest context afirmem que SDN, -on l'estat de la xarxa es transmet a un controlador central que obté una visió holística de la xarxa-, és una tecnologia clau per permetre l'adaptació a la dinàmica de xarxa, necessària per a un *wireless backhaul* en *small cells*.

1.4. Treballs relacionats

L'aplicació de les tècniques de SDN a l'entorn de xarxes sense fils, és actualment un tema d'investigació, amb contribucions disponibles en diferents dominis.

En [4] (*An architecture for software defined wireless networking*), els autors proposen una arquitectura basada en SDN per a les futures xarxes mòbils, on un controlador és utilitzat per orquestrar les tecnologies d'accés *wireless* heterogènies, incloent aspectes com ara QoS i la mobilitat. En aquest treball però, només es proposa una arquitectura d'alt nivell sense detalls sobre els protocols o algoritmes emprats.

En [5] (*Towards Programmable Enterprise WLANs with Odin*) s'introdueix el controlador Odin, que implementa certs components MAC superiors, i pot ésser utilitzat per orquestrar punts d'accés a una xarxa d'accés WLAN. Un controlador com l'Odin, es pot utilitzar en xarxes d'*Small Cells* per controlar les unitats d'accés il·lustrades en la Figura 2.2. Aquest treball s'estén més en [6] (*Towards a scalable and near-sighted control plane architecture for WiFi SDNs*).

A diferència de [5] o [6] no obstant això, en aquest projecte –SESAME-, s'investiga l'aplicació de SDN per controlar la xarxa *backhaul* en lloc de la xarxa d'accés.

El treball de [7] (*OpenFlow for Wireless Mesh Networks*) és el més semblant al presentat en aquest projecte, ja que introdueix una arquitectura de processament basat en OpenFlow per a xarxes *mesh*.

No obstant això, el projecte SESAME avança el treball de [7] en diversos aspectes, com

ara minimitzar la càrrega de gestió per configurar els nodes *wireless*, i de manera més significativa, extenent el protocol OpenFlow per tal de poder transportar els paràmetres *wireless*.

Totes les referències anteriors estan aplicades en el món de les comunicacions mòbils. Però l'arquitectura SDN no només s'aplica en aquest àmbit, sinó que també en àmbits on s'ha de gestionar el tràfic entre molts servidors com és el cas dels *Data Centers*.

Diverses empreses estan aplicant l'arquitectura SDN per solucionar les necessitats dels usuaris actuals i, poder fer un ús més eficient del seus recursos que repercutiran amb un estalvi energètic. Altres fabricants, ofereixen una arquitectura SDN explicant els seus avantatges. Podem trobar a [8] un document on s'expliquen diverses curiositats de la implementació i, com es pretén fer l'evolució de l'arquitectura antiga cap a la nova.

Com a novetat actual amb aquesta arquitectura, tenim que Google està plantejant la idea de introduir la tecnologia SDN a nivell de connexions WAN. Amb aquest moviment per part del gran gegant, ens adonem del gran impuls i dels beneficis que comporta aquesta tecnologia. Podem trobar a [9] una interessant presentació de la seva proposta.

CAPÍTOL 2. DISSENY I CONSTRUCCIÓ

En aquest segon capítol plantegem com hem dissenyat l'arquitectura del nostre projecte. Dividirem el capítol en diversos apartats per tal d'anar aprofundint a poc a poc en el seu disseny i construcció. Primerament, avaluarem els problemes amb els que ens hem trobat i la solució que hem donat, per després explicar l'arquitectura global. Seguidament, s'explicaran tots els components utilitzats i modificats per a fer que aquesta arquitectura funcioni correctament.

2.1. Entorn i proposta dels elements de l'arquitectura

Per tal de poder dissenyar l'arquitectura, primer vam haver d'enumerar tots els problemes que ens podríem trobar a l'hora d'implementar la tecnologia SDN en una xarxa *wireless*.

El primer problema que vam identificar va ser que una xarxa *wireless* és una xarxa de difusió (*broadcast*). Això comporta que el tràfic que envia un node el reben tots els que estan a una certa distancia en aquella freqüència. Aquests tipus de connexions s'anomenen connexió punt-multipunt. Donat que la implementació actual de l'arquitectura SDN segueix un model punt-punt, si la implementéssim directament, ens trobaríem amb que la xarxa es modela d'una forma errònia. Això derivaria a un càlcul de rutes erroni, ja que no estaria modelada correctament.

El segon problema deriva del fet que l'arquitectura SDN, al estar pensada per a una xarxa cablejada, no pot gestionar informació *wireless* de les interfícies dels nodes. Aquest problema afecta a tres tecnologies que volem utilitzar de la nostra arquitectura:

- **Open vSwitch** (Agent OpenFlow): no té definits els camps i no té connexió amb les interfícies *wireless* per obtenir les estadístiques.
- **Protocol OpenFlow** versió 1.0: no està dissenyat per transportar les estadístiques *wireless*.
- **Controlador**: no hi ha cap mòdul que gestioni l'encaminament en funció de les estadístiques *wireless*.

A continuació proposem una solució per a cada un dels problemes:

Problema punt-multipunt

S'ha dissenyat una capa que adapta la visió de la xarxa *wireless* com si fos una xarxa cablejada, per a que el controlador la pugui gestionar correctament. Aquesta capa la formen 3 parts:

- Sistema d'etiquetatge (VLAN): es dissenya un sistema d'etiquetes que ens permetrà diferenciar el tràfic *wireless* dels diferents clients connectats. Aquestes etiquetes s'extreuen del *driver* $mac80211$ mitjançant el protocol anomenat PLM.
- $mac80211$: es modifica el *driver* per a que sigui capaç de gestionar el tràfic etiquetat.

- Entitat MUX: s'encarrega de crear un port físic al Open vSwitch per cada connexió *wireless*. El nom del port està mapejat de tal manera que es pugui identificar a quin node *wireless* pertany. El tràfic de cada node va etiquetat per a que el MUX pugui redirigir-lo pel port que li pertoca.

Problema de gestió de la informació *wireless*

S'han implementat solucions per a cada una de les parts afectades:

- Open vSwtich (OVS): s'ha dissenyat una llibreria anomenada OpenFlow-Wireless (OFW) que permet al OVS accedir a la informació de les estadístiques *wireless* dels dispositius físics.
- Protocol OpenFlow 1.0: s'han afegit les estadístiques *wireless* a l'estructura de les estadístiques del port.
- Controlador: s'ha dissenyat un nou mòdul OSGI anomenat $SesameForwarding$ que gestiona les estadístiques *wireless* obtingudes dels nodes i les utilitza per recalcular rutes. El controlador utilitzat és l'OpenDaylight.

Un cop presentats els problemes als quals ens enfrontaríem, i les solucions que voliem donar, procedirem a explicar l'arquitectura global del nostre projecte.

2.2. Arquitectura global

L'objectiu del treball és dissenyar un sistema que permet a un controlador SDN controlar el pla d'encaminament en una xarxa *backhaul wireless*. Per tal de poder implementar aquest control de manera efectiva, cal que el controlador sigui conscient de les condicions dels enllaços ràdio i de l'estat dels dispositius.

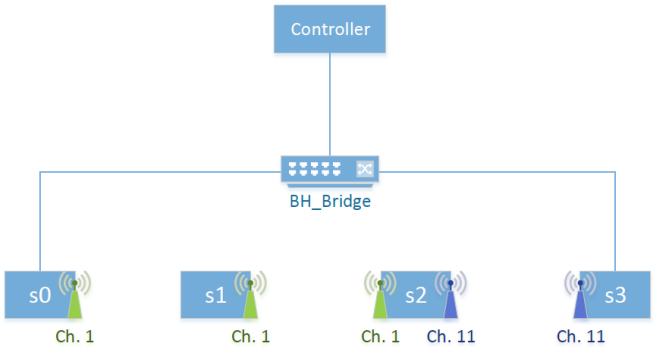

Figura 2.1: Arquitectura física considerada

A la Figura 2.1 podem veure l'arquitectura física corresponent a l'arquitectura lògica de la Figura 2.2. Observem que dos dels nodes ($s0$ i $s3$) estan connectats directament a la xarxa *backhaul* a través d'una connexió cablejada, mentre que els nodes $s1$ i $s2$ només tenen interfícies *wireless* operant en diferents canals. Tenint en compte que no utilitzem

una interfície alternativa operant en una altra freqüència (per exemple a 5GHz), necessitem que la senyalització de control s'enviï a través de la mateixa connexió. Per tant, necessitarem que la nostra arquitectura treballi de manera $in-band$, que s'explicarà en detall a la secció 2.6.

La Figura 2.2 mostra l'arquitectura lògica del nostre sistema on podem observar un controlador de SDN C, que controla el funcionament de diversos *switch wireless* a la xarxa mitjançant una interfície *south-bound* Ext_{SB}. De manera que tots els nodes tenen una connexió definida -un camí-, cap al controlador. Aquesta connexió no serà directa en els casos en els quals els nodes no estiguin connectats directament a la xarxa *backhaul*.

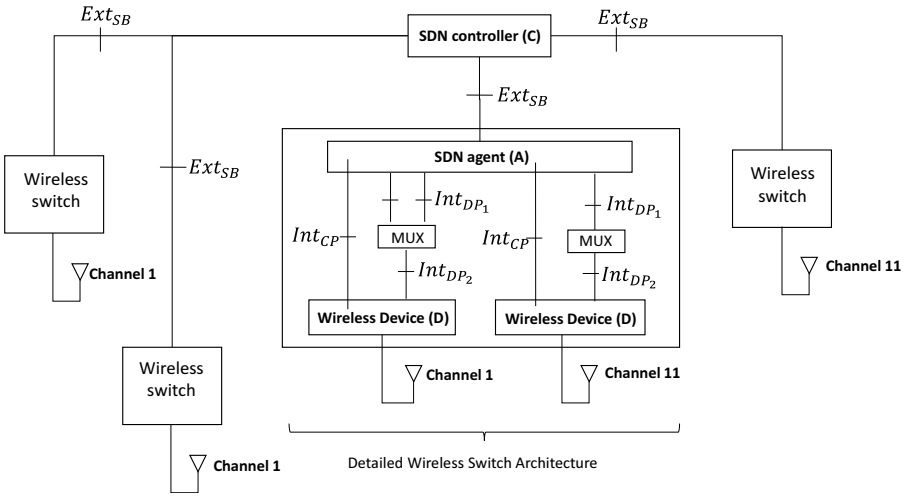

Figura 2.2: Arquitectura lògica considerada

El *switch* central de la Figura 2.2 detalla l'arquitectura interna d'un *switch wireless*, on podem apreciar els següents components principals:

- Un o més dispositius *wireless* D, són interfícies d'accés *wireless*, on cada dispositiu pot ser configurat per operar de manera independent. Sense pèrdua de generalitat, d'ara en endavant considerarem dispositius *wireless* de l'estàndard 802.11 com els dispositius *wireless* de la nostra arquitectura. Observem que, si bé l'aplicació d'alguns mecanismes del nostre sistema han estat optimitzats per 802.11, la nostra arquitectura proposada és prou genèrica com per abastar altres tecnologies ràdio.

- Un agent SDN A, controla el pla d'encaminament dels *switch wireless* i es comunica amb el controlador remot de SDN (C).

- Un MUX, entitat funcional que multiplexa els múltiples enllaços *wireless* sobre un únic dispositiu *wireless* D. Per exemple, a la Figura 2.2 el dispositiu *wireless* que opera en el canal 1 està al rang de comunicació de dos dispositius més, per tant, el MUX multiplexa dues interfícies.

- Un parell d'interfícies Int_{DP_i} i Int_{CP} s'executen entre l'agent de SDN A i el dispositiu *wireless* D, on:

  i) Int_{DP_i} transporta els paquets del pla de dades que han de ser transmesos a través de la xarxa *wireless*

ii) Int_{CP} és una interfície del pla de control que permet al agent SDN demanar al dispositiu *wireless* D sobre les condicions instantànies de l'enllaç ràdio a la xarxa.

Un requisit important de la nostra arquitectura és, que ha de ser fàcilment implementable utilitzant tecnologies SDN i components *software* disponibles, que han sigut desenvolupats sobretot per a entorns cablejats. Aquest supòsit és important, ja que simplifica el desplegament de la nostra arquitectura, que permet reutilitzar els nombrosos desenvolupaments SDN realitzats per a les xarxes cablejades, i que està en línia amb el treball portat a terme per l'ONF [10]. Per a aquest propòsit, la interfície *southbound* Ext_{SB} de la nostra arquitectura està basada en el protocol OpenFlow [11].

2.3. Funcions del pla de dades i interfícies

2.3.1. OpenFlow Switch (Open vSwitch)

Donat que la interfície Ext_{SB} està implementada utilitzant OpenFlow, l'agent de SDN A és essencialment un *switch* OpenFlow. Els *switch* OpenFlow controlen l'encaminament mitjançant un conjunt de ports Ethernet que composen el *datapath* del *switch*, on cada port proporciona una connexió física punt a punt cap a un altre *switch* de la xarxa.

El switch OpenFlow que hem utilitzat en la nostra arquitectura és l'Open vSwitch, abreviat com a OVS. L'OVS és un *switch* virtual multicapa, designat per permetre la automatització massiva de la xarxa a través d'extensions programables, sense deixar de donar suport a les interfícies i protocols de gestió estàndards. A més, també està designat per suportar la distribució a través de múltiples servidors físics, similar a *VMware vNetwork distributed vswitch* o el Cisco Nexus 1000V. OVS pot operar tant com a un *switch* virtual executant-se en un *hypervisor*, com una pila de control per *switching silicon*[1]. OVS s'ha portat a múltiples plataformes de virtualització i *chipsets* de commutació. És el *switch* per defecte de XenServer 6.0, Xen Cloud Platform i també suporta Xen, KVM, Proxmox VE i VirtualBox. També s'ha integrat en diversos sistemes de gestió virtuals incloent OpenStack, openQRM, OpenNebula i oVirt.

Opera tant en *user-space* com en *kernel-space*. Les eines i el *control path* operen en user-space, mentre que el *datapath* pertany a la part de *kernel-space*. Els conceptes bàsics de l'OVS són:

- Un *switch* –o *bridge*-, conté ports
- Un port pot tenir una o més interfícies
- Els paquets són encaminats per flux.

[1] Switching Silicon Engine (SSE) és un mecanisme de commutació i encaminament que compara les dades de l'enllaç o la capçalera de la capa de xarxa d'un paquet entrant, amb una *silicon-switching cache*. Determina l'acció apropiada (encaminament o commutació), i reenvia el paquet a la interfície apropiada. El SSE està programat directament en el *hardware* del processador del *silicon switch* (SSP) de la sèrie de routers 7000 de Cisco. Per tant, pot portar a terme la commutació independentment del processador del sistema, fent que l'execució de les decisions d'encaminament siguin molt més ràpides que si fossin programades en *software*.

OVS suporta un nombre de característiques que permeten a un sistema de control de xarxa respondre i adaptar-se a mesura que canvia l'entorn. Això inclou la compatibilitat i visibilitat senzilla com ara NetFlow, IPFIX i sFlow. Però el més útil en el nostre cas, és que OVS suporta una base de dades de l'estat de la xarxa (OVSDB) que suporta *triggers* remots. De manera que, una peça de *software* d'orquestració, –el nostre controlador de SDN-, pot "veure" diversos aspectes de la xarxa i respondre si canvien, o bé, quan canvien.

L'OVS també suporta l'OpenFlow com a mètode per exportar l'accés remot al control del tràfic. Hi ha nombrosos usos per això, incloent descobriment global de la xarxa com per exemple LLDP, CDP, OSPF, etc. OVS inclou múltiples mètodes per especificar i mantenir les regles d'etiquetatge, essent accessibles per a un controlador remot. A més, OVS tant pot controlar una implementació pura en *software* com un *switch* físic.

Utilitza l'estàndard 802.1Q per al model d'etiquetes VLAN. A part de suportar també túnels GRE o IPSEC entre d'altres.

A la Figura 2.3 es poden apreciar els principals components de la versió 2.1.2 que són els següents:

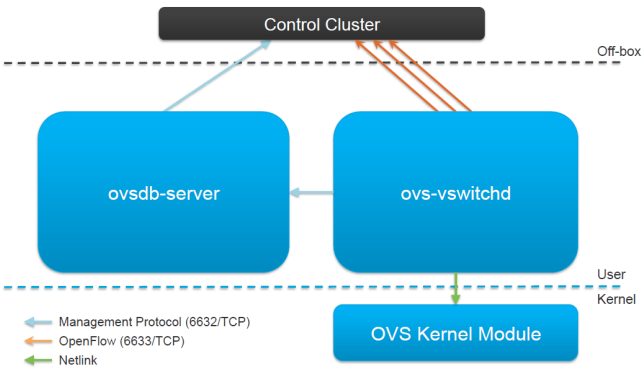

Figura 2.3: Components principals del OVS

ovs-vswitchd
: Un dimoni que implementa el *switch*, juntament amb un mòdul del *kernel* de Linux per a la commutació basada en fluxos. Hi parla mitjançant Netlink. Permet fusionar múltiples *switch* virtuals en un únic *datapath*.

ovsdb-server
: Un servidor de base de dades lleuger que $ovs-vswitchd$ consulta per obtenir la configuració.

ovs-dpctl
: Eina per configurar el mòdul del *kernel* del *switch*.

ovs-vsctl
: Una utilitat per consultar i actualitzar la configuració del $ovs-vswitchd$. Permet gestionar el *switch*.

ovs-appctl
: Una utilitat que envia comandes als dimonis del OVS que s'estan executant.

ovs-ofctl
: Permet administrar i monitoritzar remotament el *switch* a través del protocol OpenFlow.

Com hem dit anteriorment, l'Open vSwitch es basa en l'encaminament per flux. El procediment que seguiria el primer paquet d'un flux en concret seria el següent:

- El primer paquet d'un flux arriba al mòdul del *kernel space* del OVS (*datapath*) i veu que no té cap acció associada a aquest flux.
- Envia el paquet cap al mòdul del *user-space* ($ovs-vswitchd$ que és el *control path*) i aquest programa les accions del *datapath* per a aquest flux en concret i les guarda a la *Flow table* del *kernel*. Acostuma a associar-hi una acció, però també pot ser una llista. Les accions inclouen:
  - Reenviar cap a un/s port/s
  - Encapsular i reenviar cap al controlador
  - Descartar
- Retorna el paquet cap al *datapath* (mòdul del *kernel space*) per a ser enviat cap a on correspongui.
- Els paquets que arribin després pertanyents a aquest mateix flux, són gestionats directament pel *datapath* (mòdul del *kernel* del OVS).

Cal recordar que, l'encaminament a través del mòdul del *kernel* és molt més ràpid que no pas passant pel *user space*. Per sort, només haurem de passar pel $vswitchd$ quan es tracti del primer paquet d'un flux determinat. Podem apreciar el procediment descrit anteriorment en la Figura 2.4.

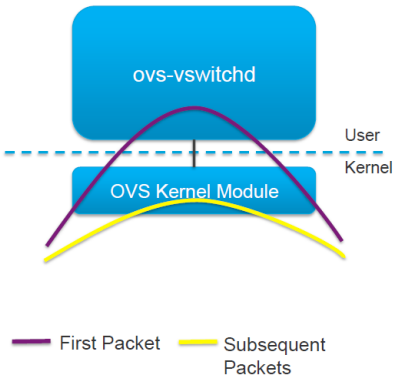

Figura 2.4: Disseny de l'encaminament del OVS

En la nostra arquitectura hem fet córrer diversos *switch* en un mateix *host*, permetent-nos realitzar les proves virtuals de l'arquitectura que comentarem en el Capítol 3. En la Figura 2.5 podem observar l'arquitectura. Tenim 4 *namespaces* diferents, un dels quals és el que pertany al controlador ($ns\_controller$). La resta, són instàncies independents del dimoni del OVS, per tal d'emular 3 *switch* diferents amb les seves pròpies interfícies i les seves

pròpies bases de dades del $ovsdb-server$. Tant el *switch* $s0$ com el *switch* $s2$ estaran connectats a la xarxa *backhaul* a través d'un BH_bridge -que no és un namespace, sinó un *bridge* virtual-, mentre que els 3 *switchs* estaran connectats entre sí a través d'enllaços ràdio virtualitzats.

Aquests enllaços ràdio es virtualitzen amb un mòdul del $mac80211$, que són els *drivers* que gestionaran les interfícies *wireless* del nostre escenari tant virtual com real.

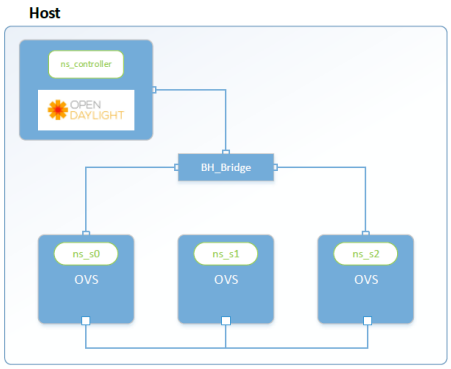

Figura 2.5: Diagrama bàsic del muntatge virtual

2.3.2. mac80211

Existeixen diverses implementacions de *drivers* (controladors) dels dispositius *wireless* degut a que cada companyia fabrica els seus. Les companyies intenten treure la màxima eficiència del seu *hardware* adaptant els controladors a la seva manera. Això comporta que siguin diferents per a cada un i significa que no hi ha una única manera de tractar els dispositius. En conseqüència, sorgeixen incompatibilitats entre els sistemes tant per *hardware* com per *software*.

El $mac80211$ [12] és una API o *framework* que adapta els controladors dels fabricants per tal de que hi hagi una manera estàndard d'accedir a ells. A la Figura 2.6 podem observar com interactua el $mac80211$ dins de les diferents capes del *kernel* de Linux.

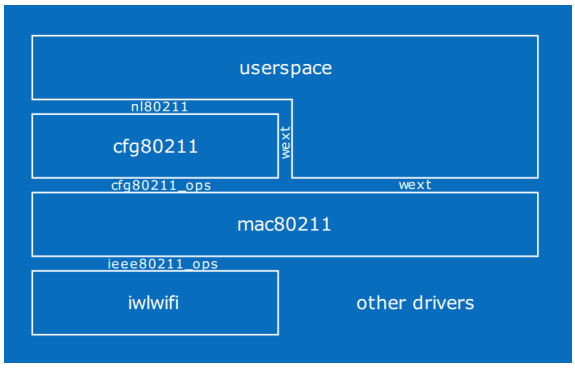

Figura 2.6: mac80211 en el mòdul del kernel de Linux

Al mòdul $cfg80211$ és on s'allotja la gestió de configuració dels dispositius *wireless*. Aquest mòdul i el $mac80211$ formen part del *kernel space* ja que són els encarregats d'obtenir la informació dels *drivers*. Des de *user space* podem accedir directament al mòdul

$mac80211$ o al $cfg80211$ a través de la API $nl80211$ per obtenir informació i configuració dels dispositius.

En el nostre projecte hem modificat el codi dels *drivers* $mac80211$ (els $linux-backports$), per a que pugui acceptar la metodologia d'etiquetes VLAN que s'ha implementat, explicada en profunditat a la Secció 2.4. També accedim a la API $nl80211$ per poder obtenir les estadístiques de les interfícies *wireless* per a enviar-les al controlador, que s'explicarà a la Secció 2.5.

A més, el *framework* incorpora diferents funcionalitats útils com per exemple la simulació d'interfícies *wireless* 802.11 amb el mòdul $mac80211\_hwsim$. Aquest mòdul l'utilitzem en l'entorn virtual de l'arquitectura per simular les interfícies *wireless* dels nodes, que s'explicarà al Capítol 3.

2.3.3. Interfícies

Cal dir però, que en el cas de la Figura 2.2 tenim interfícies *wireless* i la situació canvia. Tenint en compte que el *wireless* és un medi de difusió, múltiples *switch* de la xarxa poden ser accessibles físicament des d'un sol port OpenFlow *wireless*.

No obstant això, considerar connectivitat física punt a multi-punt en una xarxa OpenFlow és potencialment molest per als components existents de software de SDN, ja que moltes aplicacions –com ara el descobriment per LLDP [13]-, es basen en el fet de que els enllaços físics proporcionen connexions punt a punt. Per tant, en la nostra arquitectura vam decidir amagar les capacitats punt a multi-punt dels dispositius *wireless* de l'agent SDN mitjançant la introducció d'un MUX.

2.3.4. MUX

El MUX és una entitat funcional que està a càrrec de la multiplexació dels paquets de dades provinents de diferents interfícies físiques Int_{DP1} a través d'una única interfície virtual Int_{DP2}, i de la seva de-multiplexació en la direcció inversa.

Hem decidit utilitzar com a mecanisme de multiplexació, les VLANs del 802.1Q degut a la seva gran disponibilitat. Hem d'observar però, que la incorporació d'una etiqueta VLAN de 4 Bytes en els paquets de dades transmesos sobre l'aire, aporten una sobrecàrrega addicional que considerem acceptable, donada la funcionalitat que ens proporciona. A més, una etiqueta VLAN permet multiplexar fins a 4096 interfícies Int_{DP1} diferents, essent més que suficient per als nostres escenaris d'interès. Per tant, quan rebem un paquet des d'una interfície Int_{DP1}, el MUX posa una $VLAN_i$ al paquet i el reenvia a través de la interfície Int_{DP2}. D'altra banda, en rebre un paquet des de la interfície Int_{DP2}, el MUX treu la etiqueta VLAN del paquet i el reenvia cap a la interfície Int_{DP1} apropiada. En el següent apartat, explicarem el criteri de decisió de les VLAN entre els diferents *switch*.

Finalment, la interfície Int_{DP2} transporta els paquets etiquetats amb VLAN cap al dispositiu *wireless* D. Tal i com s'explicarà més endavant, descodifica el *switch* veí al qual se li ha d'enviar el paquet a través de la VLAN que porta incorporada el paquet.

2.4. Assignació de VLAN

Les etiquetes VLAN s'utilitzen en el MUX per multiplexar els paquets de dades que provenen de l'agent SDN, i demultiplexar els paquets provinents del dispositiu *wireless*. Tenint en compte això, es requereix un mecanisme per construir les etiquetes VLAN apropiades que s'utilitzen per representar cada interfície Int_{DP1}. En particular, el mecanisme dissenyat ha de complir amb els requisits següents:

- En rebre un paquet del MUX amb $VLAN_i$, el dispositiu *wireless* D ha de poder descodificar quin *switch* veí és el destí del paquet. És a dir, la informació d'encaminament de cada paquet està codificada a l'etiqueta VLAN.
- En rebre un paquet des del dispositiu *wireless* D amb $VLAN_i$, el MUX ha de ser capaç de descodificar la interfície Int_{DP1} per la qual el paquet ha de ser reenviat. Cal tenir present que el MUX rep una trama Ethernet, és a dir, que la capçalera *wireless* es extreta pel dispositiu *wireless* D, i d'aquesta manera, el MUX no és conscient de la direcció del *switch wireless* que ha transmès el paquet.

Existeixen problemes similars, per exemple, a MPLS, en el que s'utilitza un protocol de distribució d'etiquetes (LDP) per distribuir les etiquetes entre els diferents nodes d'una xarxa [14]. Un concepte similar s'utilitza en la nostra arquitectura. Però en lloc de definir un nou protocol de distribució d'etiquetes, ens hem basat en el protocol de *Peer Link Management* (PLM), disponible en qualsevol dispositiu *wireless* suportat pel *kernel* de Linux.

2.4.1. PLM (Peer Link Management)

El PLM és un protocol definit com a part de l'estàndard 802.11s [15] (extensions *mesh* del 802.11). Permet als dispositius *wireless* establir connexions lògiques amb altres dispositius veïns, configurats per operar en la mateixa xarxa *wireless*.

El PLM funciona mitjançant la monitorització de les transmissions de trames *Beacon* dels dispositius veïns i, implementa a "*two way handshake*" on els dispositius intercanvien les capacitats *wireless* i, un identificador local únic per l'enllaç veí, conegut com a *Local Link Identifier* (LLID). Després del *handshake* del PLM, un dispositiu *wireless* obté un identificador local únic (LLID) per l'enllaç amb el veí. Aquest valor de LLID és utilitzat pel dispositiu veí per referir-se al mateix enllaç, que s'emmagatzema localment com a *Peer Link Identifier* (PLID).

Un exemple simplificat del funcionament del protocol PLM es mostra a la Figura 2.7, en la que podem veure com cada dispositiu *wireless* D identifica possibles enllaços veïns amb un parell de valors LLID/PLID. Realment, per cada enllaç, cada dispositiu ha d'iniciar un "*two way handshake*", però només un únic *handshake* es mostra a la Figura 2.7 per simplicitat.

El nostre sistema es basa en els valors LLID/PLID intercanviats per construir les etiquetes VLAN utilitzades per multiplexar diverses interfícies Int_{DP1} a través d'un sol dispositiu *wireless*. Per això, el MUX ha de ser conscient dels valors de LLID/PLID recollits pel dispositiu *wireless*, essent possible això gràcies a l'ús d'eines de gestió per a interfícies *wireless* disponibles en el *kernel* de Linux [16].

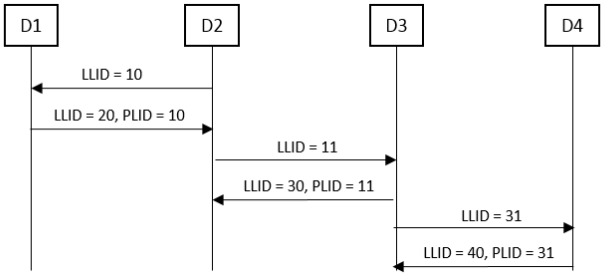

Figura 2.7: Simplificació de la configuració dels enllaços veïns del 802.11s

2.4.2. Exemple pràctic i Algorisme

La part de l'algorisme 1 que es mostra a continuació, conté la descripció dels procediments específics executats en el MUX i en el dispositiu *wireless* D i les variables utilitzades. Per l'algorisme complet veure Apèndix A.

Algorithm 1: Mux/Demux and wireless device setup algorithms

Variables:
$A \leftarrow$ SDN agent
$D \leftarrow$ wireless device
$MUX \leftarrow$ MUX function in Figure 2.2
$e_{MD} \leftarrow$ Ethernet device connecting the MUX to D
$l \leftarrow$ Single wireless link characterized by $LLID/PLID/neighbour\_address$
$L \leftarrow$ List of all known wireless links by device D
$parent\_port \leftarrow$ Parent OF port in A for inband set up
$child\_ports \leftarrow$ List of child OF ports in A for inband setup
$IP_{ctrlr} \leftarrow$ IP address of the SDN controller C
$IP_{sw} \leftarrow$ IP of this wireless switch A
Executed in the MUX
Add e_{MD} to MUX's datapath
$L \leftarrow$ Poll D for wireless links
for $l \in L$ **do**
    $(e_1, e_2) \leftarrow$ Create new veth pair to represent link l
    Add e_1 to A's datapath
    Add e_2 to MUX's datapath
    $rule_{mux} \leftarrow in = e_2 \Rightarrow out = e_{MD}$, push VLAN=$l.LLID$
    $rule_{demux} \leftarrow in = e_{MD}, VLAN = l.LLID \Rightarrow out = e_2$, pop VLAN
    Install $rule_{mux}$ and $rule_{demux}$ to MUX
Executed in the wireless device D
//Upon having to transmit a packet P
for $l \in L$ **do**
    if $P.VLAN = l.LLID$ **then**
        $P.VLAN = l.PLID$
        $P.dst\_addr = l.neighbour\_address$

En el MUX s'executen una sèrie d'accions descrites en l'Algorisme 1. Primerament el MUX descobreix quants veïns té el node *mesh* i ho emmagatzema a la llista L. Busca el cinquè byte de l'adreça MAC de la seva interfície *mesh* local. De manera que quan busca el cinquè byte dels seus veïns, pot assignar-li "nom" als enllaços que l'uneixen a ells, gràcies als identificadors que ha recollit de l'adreça MAC. Si mirem la Figura 2.8, veiem

que el $Link\_2\_1$ és l'enllaç virtual que connecta el dispositiu 1 amb el 2, i de la mateixa manera el $Link\_2\_3$ és el que connecta els dispositius 2 i 3. Abans però, és necessari un bucle per crear els *veth pairs* que aniran cap a l'agent SDN A (*switch* OVS, el de la part superior del MUX) i portaran els noms corresponents. Aquests *veth pairs* seran els enllaços virtuals dels nodes *wireless*. Un cop hem creat els *veth pairs* que aniran cap al OVS, creem el MUX que serà un *bridge* OVS i afegirem l'altre extrem dels *veth pairs*. Finalment, només queda instal·lar les següents regles de reenviament al MUX:

- Si el paquet prové d'una de les interfícies anomenades $Link\_x\_x$, s'ha de reenviar el paquet cap al dispositiu *wireless* D posant l'etiqueta VLAN corresponent al destinatari.
- Si el paquet prové de la interfície que connecta el MUX amb el dispositiu *wireless* D, aleshores ha de reenviar el paquet cap a la interfície $Link\_x\_x$ que correspongui (sense etiqueta), tenint en compte la informació codificada a l'etiqueta VLAN.

En el cas del dispositiu *wireless* D, es porten a terme unes accions gràcies a la modificació del *driver wireless* $mac80211$ (els $linux - backports$). La versió dels $backports$ utilitzada és la 3.12-1, que la podem trobar a [17]. Els hem modificat per poder implementar el mòdul *wireless* de SESAME. Les modificacions que s'hi han realitzat són les següents:

- Funcions afegides que comproven si la trama que s'ha d'encaminar té una etiqueta VLAN. Si és el cas, l'etiqueta VLAN conté un LLID que el *driver* busca en la seva llista d'adreces MAC destí i el que coincideix amb aquell LLID li executa dues accions:
  - Primer substitueix el LLID de l'etiqueta VLAN pel PLID (que és el LLID del punt *mesh* receptor).
  - Segon, envia la trama cap a l'adreça MAC que consta com a següent salt d'acord amb el LLID, de manera que evita l'encaminament HWMP[2] (*Hybrid Wireless Mesh Protocol*).
- Diverses modificacions per a fer complir l'encaminament basat en LLID/PLID. Cal recordar que les trames *broadcast*/*multicast* es transmetran com *unicast*. El direccionament correcte es transporta a la capçalera *mesh*.

L'operació de l'algorisme s'il·lustra mitjançant un exemple a la Figura 2.8.

Tal com podem veure a la Figura 2.8, el MUX utilitza el valor de LLID assignat pel dispositiu *wireless* per multiplexar i demultiplexar els paquets transmesos i rebuts. Cal dir que, el dispositiu *wireless* D està modificat de manera que només assigna LLIDs entre 0 i 4096, ja que és el valor màxim per a una VLAN. Quan es té un nou paquet per transmetre, el dispositiu *wireless* inspecciona l'etiqueta VLAN del paquet, utilitza el valor del LLID contingut per descobrir el dispositiu veí al qual se li ha de transmetre el paquet i, finalment, sobreescriu el valor de la etiqueta VLAN pel corresponent PLID d'aquell enllaç.

[2] *Hybrid Wireless Mesh Protocol* definit al IEEE 802.11s és un protocol d'encaminament bàsic per a una xarxa *mesh wireless*. Està basat en AODV i l'encaminament està basat en arbre. Es basa en el protocol PLM per descobrir punts *mesh* i seguir als nodes veïns.

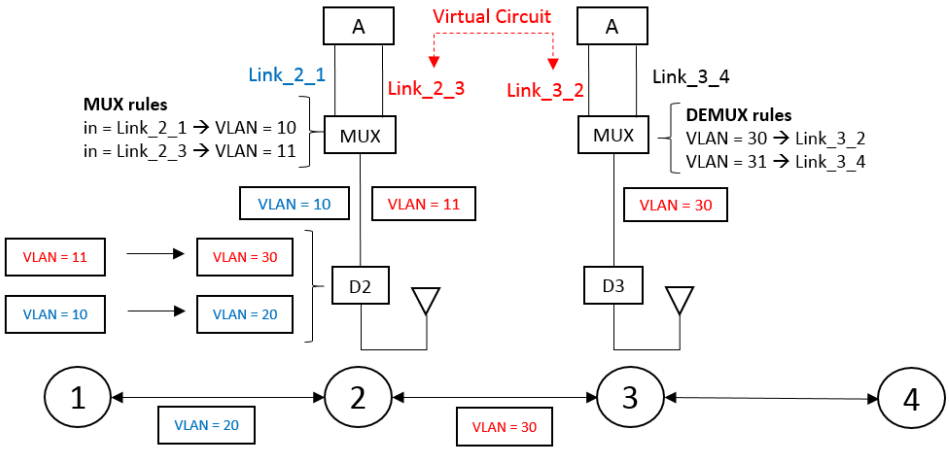

Figura 2.8: Exemple de distribució d'etiquetes i configuració del MUX

S'ha de tenir present que el dispositiu *wireless* D necessita canviar el valor de la VLAN pel PLID, ja que aquest és l'identificador assignat pel dispositiu a l'enllaç pel qual s'ha de transmetre el paquet. D'aquesta manera, es garanteix que aquest valor sigui únic pel dispositiu receptor.

Finalment, cal tenir en compte que tots els paquets rebuts per la interfície *wireless* són reenviats cap al MUX i cap a l'agent SDN, on es troba tota la intel·ligència de l'encaminament.

2.5. Interfície del pla de control i extensió del OpenFlow

Per tal de poder controlar de manera efectiva el medi *wireless*, el controlador SDN C de la Figura 2.2 ha de ser conscient de les condicions dels enllaços ràdio. Per aquesta raó, dues interfícies de la Figura 2.2, aborden aquest problema de portar la informació del medi cap al controlador.

En primer lloc, la interfície Int_{CP} permet al agent SDN A del *switch wireless* recuperar les mesures de les condicions del medi ràdio extretes des del dispositiu *wireless* D. En segon lloc, la interfície Ext_{SB} s'utilitza per transportar els paràmetres ràdio recollits des de l'agent SDN cap al controlador SDN.

Per tal de poder extreure aquestes estadístiques del medi, necessitem consultar-les a la interfície física del dispositiu. No podem fer-ho de manera directa, ja que l'Open vSwitch treballa en *user-space*, i necessitem accedir al *Kernel space* per extreure les dades. La manera de poder fer crides a mòduls del *kernel* des de *user-space* és mitjançant Netlink.

2.5.1. Netlink i nl80211

Netlink és una interfície del *Kernel* de Linux pertanyent a la família de *sockets*, que s'utilitza per a les comunicacions entre el *kernel* i els processos de *user-space*. Concretament, està dissenyat per a fer diverses transferències d'informació de xarxa. Eines de xarxa tals com $iproute2$ o *drivers* $mac80211$ utilitzen Netlink per comunicar-se amb el *kernel*.

En el nostre projecte utilitzarem la API del $nl80211$. Aquesta API és una interfície dissenyada per a poder accedir a la informació dels dispositius *wireless* tal i com hem vist en la secció 2.3.2. D'aquesta manera obrirem un canal de comunicació a través d'un *socket* de Linux per accedir als *drivers* de la interfície física com s'observa a la Figura 2.9.

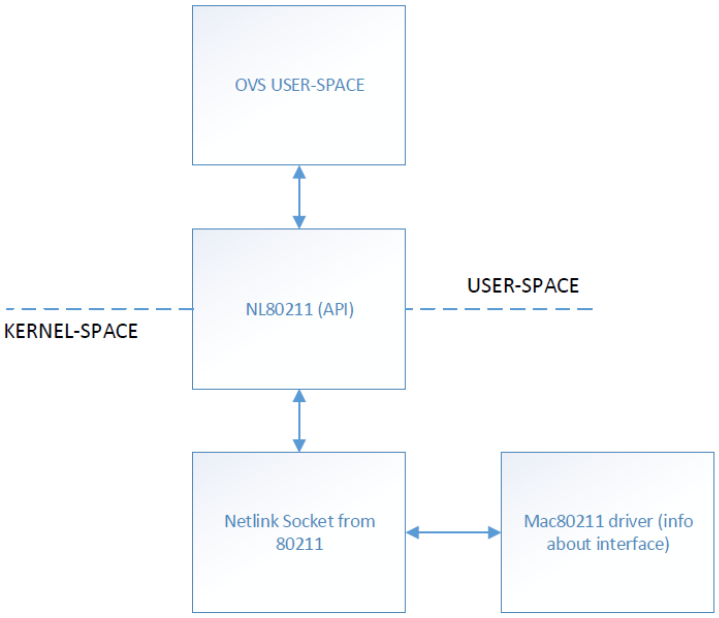

Figura 2.9: Diagrama de petició d'estadístiques

Així doncs, el nostre Open vSwitch estarà estès i modificat de manera que la funció encarregada de demanar les estadístiques de port habituals, també demani les del medi *wireless*. Ho farà cridant la nostra llibreria que utilitza Netlink per accedir als paràmetres *wireless*, processant els resultats i omplint l'estructura de dades corresponent que s'explica en la secció 2.5.2.

D'aquesta manera, l'agent SDN pot utilitzar en qualsevol moment la interfície Int_{CP} per sondejar al dispositiu *wireless* sobre les últimes condicions de l'enllaç ràdio. El dispositiu *wireless* informa de les condicions de la ràdio per a cada enllaç que ha estat descobert a través del *handshake* del PLM. La implementació d'aquesta interfície està descrita al capítol 3. Finalment, la interfície Ext_{SB} està implementada utilitzant OpenFlow 1.0 extés de manera que sigui capaç de transportar els paràmetres del medi ràdio.

2.5.2. Extensió del OpenFlow: OpenFlow-Wireless (OFW)

Hem utilitzat la versió 1.0 del OpenFlow ja que és la que suporta per defecte la versió 2.1.2 del Open vSwitch utilitzada en el projecte. La versió 1.3 del protocol OpenFlow havia de ser activada manualment dins del mòdul del $vswitchd$ del Open vSwitch, però per al nostre propòsit, la versió 1.0 cobreix de llarg les nostres necessitats.

Concretament, la nostra implementació de l'estructura ofp_port_stats definida al OpenFlow, és la que conté les estadístiques per port de cada dispositiu *wireless* i és la que s'omple mitjançant la crida a la nostra llibreria. De manera que, cada vegada que l'agent

SDN A rebi una sol·licitud de les estadístiques per port des del controlador, l'agent enviarà el paquet habitual de resposta d'estadístiques a més dels nostres camps d'interès, els paràmetres ràdio. Cal tenir en compte, que al haver afegit nous camps d'estadístiques, la mida del paquet ha augmentat, podent causar alguna petita penalització.

Els paràmetres afegits a l'estructura del OpenFlow 1.0 estan descrits a la Taula següent.

Paràmetre	**Explicació**
rx_bytes	Bytes rebuts en aquest port del dispositiu wireless
$rx\_packets$	Paquets rebuts en aquest port del dispositiu wireless
tx_bytes	Bytes transmesos en aquest port del dispositiu wireless
$tx\_packets$	Paquets transmesos en aquest port del dispositiu wireless
$tx\_retries$	Paquets reenviats en aquest port del dispositiu wireless
tx_failed	Paquets fallits en aquest port del dispositiu wireless
$signal$	Últim nivell de senyal en dBm rebut en aquest enllaç
avg_signal	Nivell mig de senyal en dBm rebut en aquest enllaç
$tx\_bitrate$	Última taxa de bit transmesa a nivell físic en aquest enllaç

Taula 2.1: Paràmetres ràdio per port del OpenFlow

Per poder analitzar els paquets modificats d'estadístiques del OpenFlow, s'ha hagut de modificar el *dissector* de l'analitzador de paquets Wireshark i així, poder examinar-los i evitar que els tracti com a paquets mal formats. A la Figura 2.10 podem veure una captura del tràfic OpenFlow on veiem que el controlador, amb ip 192.168.42.1, demana amb un $Stats\ Request$ informació al *switch* $s1$ amb adreça ip 192.168.42.201. Mentre que a la Figura 2.11 tenim un paquet $Stats\ Reply$ en resposta al $Stats\ Request$ anterior.

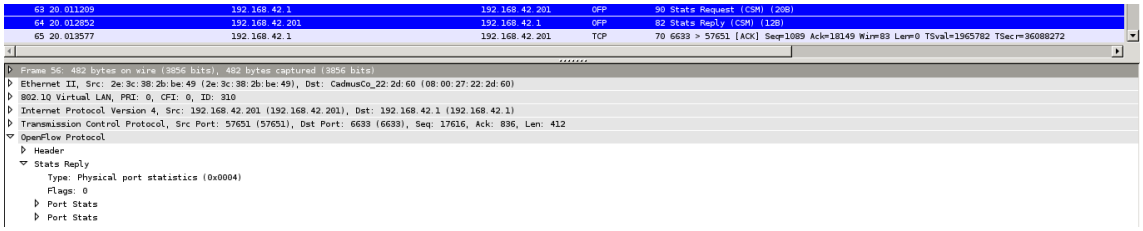

Figura 2.10: Traça del wireshark amb el paquet modificat

A la Figura 2.11, quan despleguem un dels $Port\ Stats$ del paquet podem observar les noves estadístiques *wireless* afegides a l'estructura original del paquet d'estadístiques del Openflow.

Les estadístiques que provenen de les interfícies *wireless* porten el prefix ofw (Openflow-Wireless) per distingir-les de les originals. Podem veure que hem doblat el nombre d'estadistiques, ja que originalment n'hi havia 12 i n'hem afegit 12 més. Per tant, la nostra arquitectura ha modificat la mida del paquet d'estadistqiues d'Openflow multiplicant-lo per 2.

L'estructura d'estadistiques és fixa, però la mida del paquet serà variable en funció del nombre de ports que tingui cada *switch*.

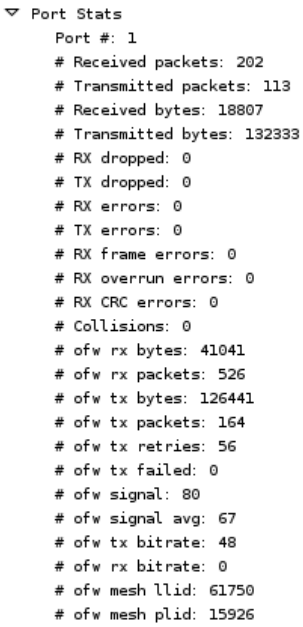

Figura 2.11: Estructura del nou paquet d'estadístiques

2.6. Connexió in-band al controlador

En els *Data Centers* on l'OpenFlow està actualment desplegat, és molt habitual utilitzar una xarxa separada per transportar la senyalització de la connexió entre els *switch* OpenFlow i el controlador. Sabent això, veiem que no és possible aplicar-ho en el nostre escenari de *Small cells*, on la senyalització de la connexió ha de ser transportada a través de les mateixes interfícies *wireless* que componen el *datapath* d'OpenFlow. Aquesta manera de connectar-nos a un controlador OpenFlow es coneix com una connexió $in-band$.

La connexió $in-band$ però, té un problema ja que hem de tenir un camí disponible per a un *switch* OpenFlow per tal de que pugui establir un primer contacte amb el controlador. En un desplegament normal, les taules dels *switch* estàn inicialment buides, i només poden ser omplertes després de l'establiment d'un canal OpenFlow amb el controlador.

Per resoldre aquest problema en la nostra implementació, l'agent de SDN A pre-omple les taules OpenFlow d'acord amb les regles descrites en l'Algorisme 2.

Algorithm 2: In-band setup algorithm

1 **Executed in the SDN agent A for inband setup**
2 // Rules dealing with ARP to/from controller
3 $arp_1 \leftarrow prot = arp, arp\_tpa = IP_{ctrlr} \Rightarrow out = parent\_port$
4 $arp_2 \leftarrow prio = 2, prot = arp, arp\_tpa = IP_{sw} \Rightarrow out = LOCAL$
5 $arp_3 \leftarrow prio = 1, prot = arp, arp\_spa = IP_{ctrlr} \Rightarrow out = child\_ports$
6 // Rules dealing to IP to/from controller
7 $ip_1 \leftarrow prot = ip, ip\_dst = IP_{ctrlr} \Rightarrow out = parent\_port$
8 $ip_2 \leftarrow prio = 2, prot = ip, ip\_dst = IP_{sw} \Rightarrow out = LOCAL$
9 $ip_3 \leftarrow prio = 1, prot = ip, ip\_src = IP_{ctrlr} \Rightarrow out = child\_ports$
10 Install arp_1, arp_2, arp_3, ip_1, ip_2 ip_3 in A

L'Algorisme emprat es basa en tenir un port OpenFlow en l'agent A designat com a "*parent*" per poder arribar al controlador, i un conjunt d'altres ports designats com a "*child*"

que seran utilitzats per transmetre la senyalització del controlador cap a altres dispositius *wireless* de la xarxa. Per tal de dur a terme la selecció *offline* de les interfícies "*parent*" i "*child*", l'Algorisme necessita que els dispositius *wireless* siguin introduïts de manera seqüencial començant per un que tingui connexió directa cablejada amb el controlador (veure Figura 1.4).

Concretament, les regles mostrades a l'Algorisme 2 s'expliquen a continuació, però només es fa èmfasi a les tres primeres, ja que les tres següents són iguals però tractant amb paquets del tipus IP.

- Amb la primera regla aconseguim que un paquet del tipus ARP que tingui com a pregunta la direcció IP del controlador, l'enviï cap al port designat com a *parent*, que és el camí cap al controlador.
- La segona regla, si arriba un paquet del tipus ARP preguntant per la IP del propi *switch*, l'encamina cap al port LOCAL (cap a sí mateix).
- A la tercera regla, si es rep un paquet ARP amb direcció IP origen la del controlador, l'envia cap als ports designats com a *child* per distribuir-lo per la xarxa fins que arribi al node que compleixi la segona regla i l'accepti.

Cal dir que aquest procés només es requereix en la fase d'alta d'un node. La trajectòria actual seguida per la connexió de senyalització, pot ésser optimitzada des del controlador un cop la connexió inicial ja ha estat establerta.

2.7. Controlador SDN

Seguint la nostra arquitectura, qualsevol controlador de SDN basat en OpenFlow que estigui disponible pot ésser utilitzat per controlar els *switch wireless* en la "*Small Cell backhaul network*".

Tenim diversos controladors disponibles escrits en llenguatges diferents, però en la nostra arquitectura utilitzarem l'OpenDaylight escrit en Java.

2.7.1. OpenDaylight

El projecte OpenDaylight [19] és un projecte col·laboratiu "open source" patrocinat per la Linux Foundation. L'objectiu del projecte és accelerar l'adopció del SDN i crear una base sòlida per a "*Network Functions Virtualization*" (NFV).

La primera versió del projecte OpenDaylight va ser publicada al febrer del 2014 sota el nom d'Hydrogen. Aquesta versió conté tres edicions diferents: base, virtualització i proveïdor de serveis. La que utilitzem en la nostra arquitectura és la "*Base Edition*".

L'edició Base d'Hydrogen és per a aquells que estan explorant el SDN i el protocol OpenFlow per a proves de concepte o iniciatives acadèmiques en entorns físics o virtuals. A la Figura 2.12 podem observar gràficament què és el que compon aquesta edició del OpenDaylight.

OpenDaylight conté –entre altres projectes-, una plataforma de controlador modular, extensible, escalable, multi-protocol i flexible. Aquest controlador està implementat estrictament en *software* i està contingut en la seva pròpia màquina virtual java (JVM). De manera que, pot ésser desplegat en qualsevol dispositiu o plataforma de sistema operatiu que suporti Java.

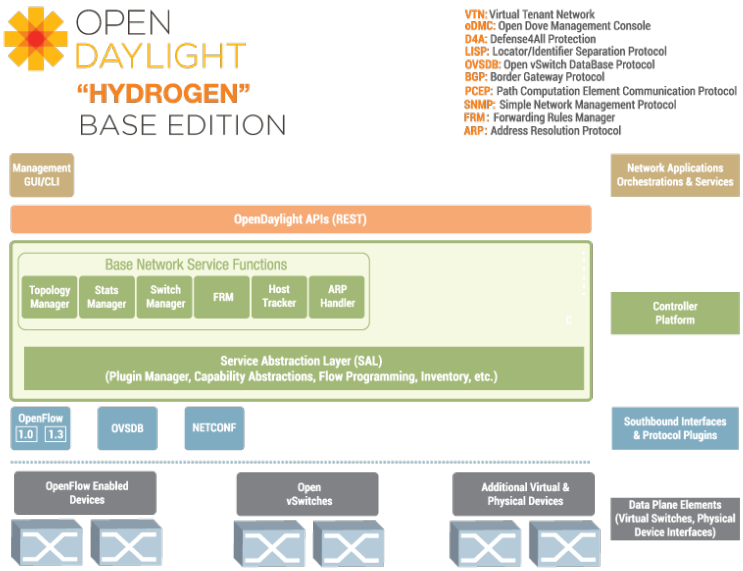

Figura 2.12: Edició Base de la versió Hydrogen del ODL

La plataforma del controlador conté una col·lecció de mòduls dinàmics per dur a terme les tasques de xarxa necessàries. Hi ha una sèrie de serveis de xarxa bàsics per a aquestes tasques, com ara entendre quins dispositius estan continguts a la xarxa i les capacitats de cadascun d'ells, la recopilació d'estadístiques, etc. A més, els serveis orientats a plataforma i altres extensions poden ser incorporats a la plataforma del controlador per millorar funcionalitats de SDN.

A la Figura 2.12 podem observar els mòduls i elements que composen aquesta versió base del Projecte OpenDaylight.

Els elements més rellevants del Projecte OpenDaylight per a la nostra arquitectura són:

Controller

Com bé hem dit anteriorment, conté tots els mòduls necessaris per a controlar una xarxa SDN com per exemple, els gestors de topologia, de *switch*, d'encaminament, d'estadístiques, etc.

El SAL proporciona les abstraccions necessàries per suportar múltiples protocols *southbound* mitjançant *plugins*. L'arquitectura *north-bound* extensible orientada a aplicació, proporciona un conjunt molt ric d'APIs *Northbound* a través de serveis web RESTful i serveis OSGi. L'algorisme d'encaminament que utilitza és el Djikstra.

La Figura 2.13 il·lustra l'arquitectura del controlador. La interfície *southbound* és capaç de suportar múltiples protocols (com a *plugins* separats), per exemple: OpenFlow 1.0, OpenFlow 1.3, BGP-LS, etc. Aquest mòduls s'enllacen de manera dinàmica amb el *Service Abstraction Layer* (SAL).

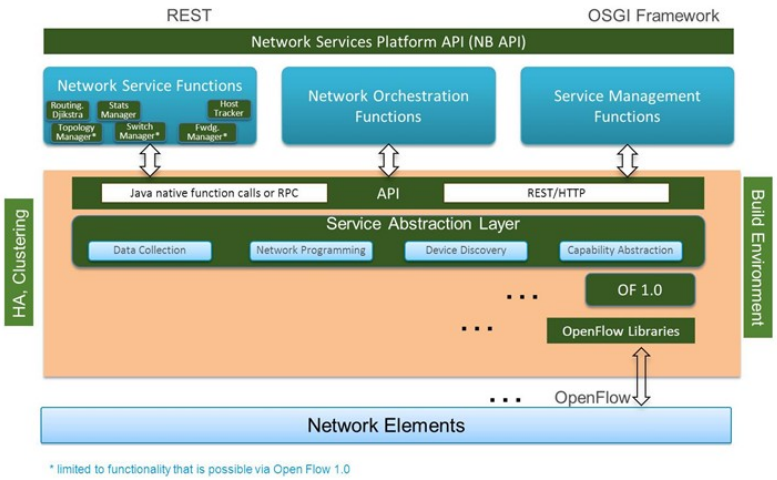

Figura 2.13: Arquitectura del controlador OpenDaylight

El SAL està al cor del disseny modular del controlador i permet suportar múltiples protocols al *southbound* i proporcionar serveis consistents pels mòduls i les aplicacions (on resideix la intel·ligència del negoci). El SAL determina com s'han de complir els serveis sol·licitats independentment del protocol subjacent emprat entre el controlador i els dispositius de la xarxa. Bàsicament, adapta les diferents tecnologies utilitzades en el *southbound*. Per tal de que el controlador pugui controlar els dispositius que estan sota el seu domini, necessita saber sobre aquests dispositius, com ara les seves capacitats, l'accessibilitat, etc. Aquesta informació es emmagatzemada i gestionada pel *Topology Manager*. Els altres components com el *ARP Handler*, *Host Tracker*, *Device Manager* i *Switch Manager* ajuden a generar la base de dades de la topologia per al *Topology Manager*.

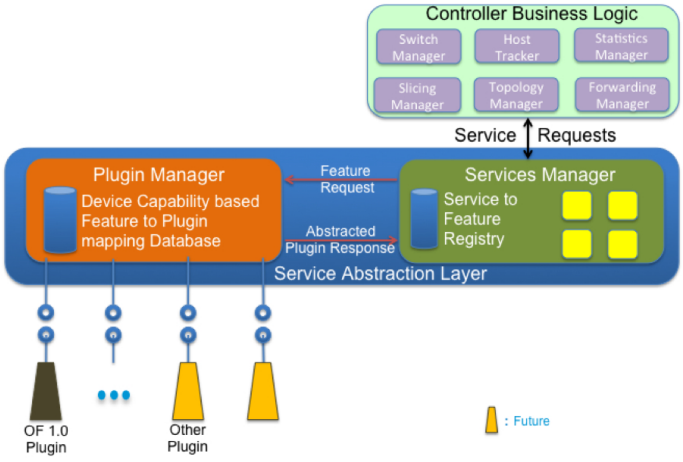

Figura 2.14: Diagrama del SAL

El SAL proporciona serveis bàsics com el descobriment de dispositius, que són utilitzats pels mòduls com el *Topology Manager* per construir la topologia i les capacitats del dispositiu. Els serveis es construeixen utilitzant les característiques exposades pels *plugins* (basats en la presència d'un *plugin* i les capacitats del dispositiu de xarxa). Basant-se en la petició de servei, el SAL la mapeja cap al *plugin* adequat i per tant, cap al protocol *Southbound* més apropiat per interactuar amb el dispositiu de xarxa donat. Cal recordar, que cada *plugin* és independent d'un altre.

Al cap de poc temps però, la comunitat va decidir migrar del AD-SAL (*Api Driven* SAL) a MD-SAL (*Model Driven* SAL), tal i com es pot veure a la Figura 2.15. Volien deixar enrere les APIs creades de manera estàtica (AD-SAL), per les APIs generades automàticament mitjançant models de dades del tipus YANG (MD-SAL), sempre mantenint la compatibilitat amb versions anteriors. D'aquesta manera, s'aconsegueix que sigui més fàcil desenvolupar components i aplicacions per al controlador.

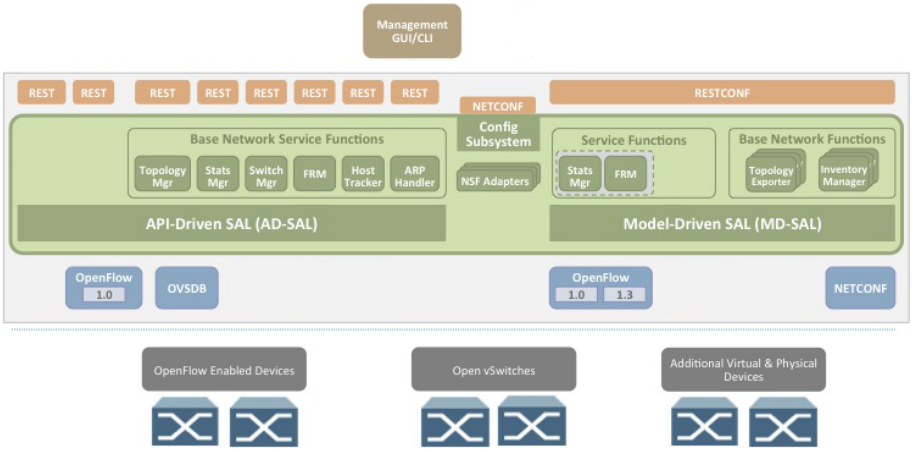

Figura 2.15: Arquitectura del controlador OpenDaylight (Hydrogen - Base Edition)

Les principals diferències entre el AD-SAL i el MD-SAL s'il·lustren en la taula 2.2.

El connector NETCONF és el *plugin Southbound* per a gestionar i configurar els dispositius de la xarxa. El controlador té un servidor intern de NETCONF que l'utilitza per a crides *Northbound*. El RESTCONF és el *plugin Northbound*. "*Config Subsystem*" és un *framework* basat en Netconf/Yang per a la configuració, rendiment i gestió de les fallades de la infraestructura del controlador i els *plugins* desplegats al mateix controlador.

El controlador exposa APIs *northbound* obertes que són utilitzades per les aplicacions. L'OpenDaylight suporta el *framework* OSGi[3] i REST bidireccional per l'API *northbound*.

El *framework* del OSGi s'utilitza en aplicacions que s'executaran en el mateix espai d'adreces que el controlador, mentre que l'API REST (basat en web) s'utilitza per aplicacions que no s'executen en aquest mateix espai d'adreces. La lògica de negoci i els algorismes resideixen en les aplicacions. Aquestes aplicacions utilitzen el controlador per recollir la intel·ligència de la xarxa, executar algorismes per realitzar anàlisis, i aleshores utilitza el controlador per orquestrar les noves regles a través de la xarxa.

El controlador disposa d'una interfície web d'usuari (GUI), implementada com a una aplicació que utilitza la *Northbound* REST API per interactuar amb els altres mòduls del controlador.

A part dels mòduls principals que componen el controlador, cal parar atenció en les interfícies Java que s'utilitzen per a interactuar entre sí, o bé, per a subscriure's a serveis determinats dels *Bundles*. A la Taula 2.3 podem veure les interfícies més importants que s'utilitzen per desenvolupar un *bundle* nou que doni servei a la xarxa.

[3]Open Service Gateway initiative (OSGi) descriu un sistema modular i una plataforma de serveis per al llenguatge Java que implementa un model de components complet i dinàmic. Les aplicacions o els components, que estan en forma de bundles (paquets) per al desplegament, poden ser instal·lats, iniciats, parats, actualitzats i desinstal·lats de manera remota, sense la necessitat d'un reinici.

AD-SAL	**MD-SAL**
API Driven SAL	Model Driven SAL
APIs SAL: les sol·licituds d'encaminament entre els consumidors i els proveïdors, i les adaptacions de les dades estan definides de manera estàtica en la compilació/construcció.	APIs SAL: les sol·licituds d'encaminament entre els consumidors i els proveïdors estan definides a partir de models, mentre que les adaptacions de les dades estan proporcionades per *plugins* d'adaptació "interns".
AD-SAL típicament té *Northbound* (NB) i *Southbound* (SB) APIs fins i tot per a funcions/serveis que s'assignen 1:1 entre els *plugins* SB i NB	MD-SAL permet als *plugins* NB i SB utilitzar la mateixa API generada a partir d'un model de dades. Un *plugin* esdevé l'API del proveïdor (servei), mentre que l'altre esdevé l'API del consumidor (servei).
En l'AD-SAL hi ha una API REST dedicada per a cada *plugin* NB/SB.	MD-SAL proporciona una API REST comuna per accedir a les dades i funcions definides en els models.
AD-SAL proporciona peticions d'encaminament (selecciona el *plugin* SB basantse en el tipus de servei) i opcionalment, proporciona adaptació del servei si una API NB és diferent de la seva corresponent API (protocol) SB.	MD-SAL proporciona peticions d'encaminament i la infraestructura per donar suport a l'adaptació de serveis, però no proporciona adaptació de serveis en sí; l'adaptació de serveis està proporcionada pels *plugins*.
Les peticions d'encaminament estan basades en el tipus de *plugin*: el SAL sap quina instància de node és servida per quin *plugin*, i quan un *plugin* NB sol·licita una operació en un node en concret, la petició és encaminada cap al *plugin* adequat i aquest encamina la petició cap al node corresponent.	Les peticions d'encaminament en el MD-SAL es realitzen tant en el tipus de protocol com en les instàncies del node, ja que les dades de la instància del node s'exporten des del *plugin* cap al SAL.
AD-SAL és "sense estat" (*stateless*)	MD-SAL pot emmagatzemar dades per als models definits pels *plugins*. Els *plugins* de proveïdor i consumidor poden intercanviar dades a través de l'emmagatzematge del MD-SAL.
Està limitat a dispositius compatibles amb fluxos i models de servei.	Model agnòstic. Pot suportar qualsevol model de dispositiu o servei i no està limitat a dispositius compatibles amb fluxos i models de servei.
Els serveis de AD-SAL acostumen a proporcionar versions tant asíncrones com síncrones del mateix mètode de la API.	A MD-SAL, les APIs de models de servei només proporcionen APIs asíncrones, però permeten al sol·licitant bloquejar la sol·licitud fins que no sigui processada i un objecte resultat estigui disponible. La mateixa API pot ser utilitzada tant per a un enfocament síncron i asíncron. Així l'MD-SAL, promou l'enfocament asíncron per al disseny d'aplicacions, però no exclou les aplicacions síncrones.

Taula 2.2: Comparativa AD-SAL vs. MD-SAL

Finalment, el controlador OpenDaylight està preparat per treballar tant amb la versió 1.0 del OpenFlow, com amb la 1.3. En la nostra arquitectura però, utilitzarem la versió 1.0 ja que l'Open vSwitch que estem utilitzant opera en aquesta versió d'OpenFlow.

Bundle	Interfícies importants	Descripció
arphandler	*IHostFinder*	Component responsable d'aprendre la localització dels *switchs* a través dels paquets ARP.
hosttracker	*IfIptoHost*	Seguiment de la localització del host relatiu a la xarxa SDN.
switchmanager	*ISwitchManager*	Component que conté tota la informació de l'inventari de *switchs*.
topologymanager	*ITopologyManager*	Component que conté tota la xarxa SDN en format gràfic.
usermanager	*IUserManager*	Component que s'encarrega de la gestió de l'usuari.
statisticsmanager	*IStatisticsManager*	Component que s'encarrega d'utilitzar el SAL *ReadService* per recollir les estadístiques de la xarxa SDN.
sal	*IReadService*	Interfície per obtenir de qualsevol node de la xarxa una visió de *hardware* de flux, ports i cues.
sal	*ITopologyService*	Mètodes de topologia oferts pel SAL a les aplicacions.
sal	*IFlowProgrammerService*	Interfície per instal·lar/modificar/eliminar fluxes en un node de la xarxa.
sal	*IDataPacketService*	Seveis de paquets SAL oferts a les aplicacions.
web	*IDaylightWeb*	Component per seguir parts del sistema d'interfícies depenent dels *bundles* instal·lats en el sistema.

Taula 2.3: Interfícies importants del controlador OpenDaylight

OpenFlow Plugin

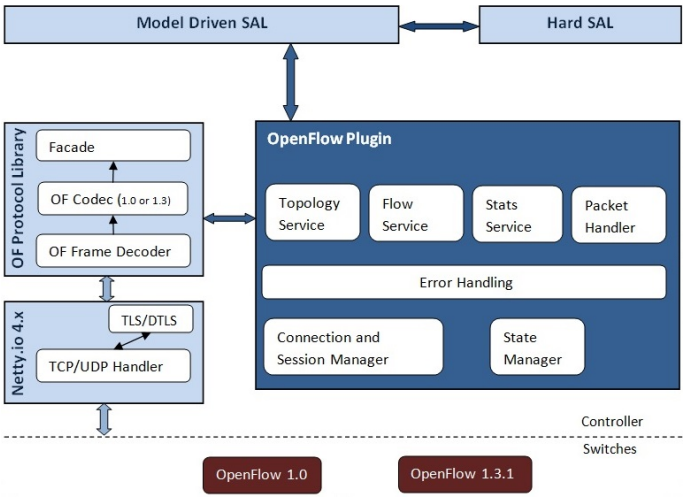

Figura 2.16: Arquitectura de l'OpenFlow Plugin

És un *plugin* del *southbound* que realitza la integració de la llibreria del protocol OpenFlow al *Model Driven Service Abstraction Layer* (MD-SAL) del controlador. És una interfície de

comunicacions definida per a permetre la interacció entre les capes de control i encaminament d'una arquitectura SDN. El *plugin* està basat en l'arquitectura de MD-SAL. Aquest *plugin* suporta tant la versió 1.0 com la 1.3 del protocol OpenFlow.

OpenFlow Protocol Library (OpenFlowJava)

És un component del OpenDaylight que intervé en la comunicació entre el controlador i els dispositius físics que suporten el protocol OpenFlow. Ofereix suport de múltiples versions del protocol OpenFlow (1.0 i 1.3) i possibles extensions del mateix. L'objectiu principal és proporcionar a l'usuari o, a les capes superiors del controlador, un canal de comunicació que pot ser utilitzat per gestionar els dispositius físics de la xarxa.

OVSDB

Implementa el protocol de gestió Open vSwitch Database, permetent la configuració *southbound* dels *switches*.

Open vSwitch Database Management Protocol, és un protocol de configuració OpenFlow designat per a gestionar les implementacions del Open vSwitch. En una implementació d'Open vSwitch, s'utilitza un servidor de base de dades i un dimoni de *switch*. El protocol OVSDB s'utilitza en clústers de control per subministrar informació al servidor de base de dades del *switch*.

En l'espai del SDN, el controlador i les aplicacions poden interactuar amb l'OVS a través de dos canals diferents: OpenFlow i OVSDB. Mentre el protocol OpenFlow aborda la funció de l'encaminament del OVS, l'OVSDB aborda el pla de gestió.

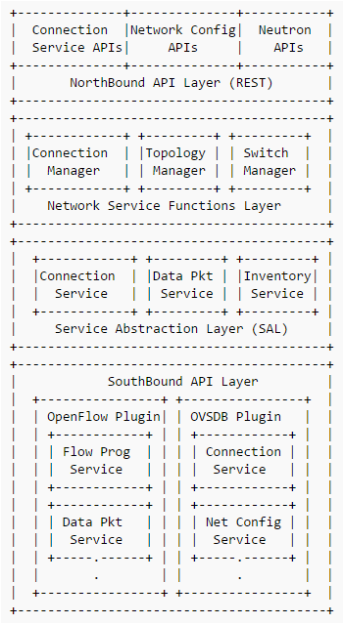

Figura 2.17: Arquitectura del controlador OpenDaylight amb els dos canals (OF i OVSDB)

YANG tools

És un projecte d'infraestructura amb l'objectiu de desenvolupar eines i llibreries que donen suport a NETCONF i YANG per a projectes i aplicacions Java com ara el Model Driven SAL

(MD-SAL) del controlador, que utilitza YANG com a llenguatge de modelat, i NETCONF /OFConfig *plugins*.

Està format per cinc subprojectes lògics que es descriuen a continuació:

- ***Concepts*** defineix uns conceptes base i unes classes "*helper*" que son agnòstiques al projecte i poden ser utilitzades fora de l'abast del projecte de YANG Tools.
- El projecte ***YANG*** conté *artifacts* relacionats amb el YANG, com ara la especificació que el relaciona amb el Java, traductor de YANG, model semàntic i *plugin* de Maven per processar fitxers del YANG.
- El ***Code Generator*** conté definicions per al generador de codi i classes d'utilitat per a generar els fitxers Java basats en els models YANG traduïts.
- Per últim, el ***Model*** és on es troben els mòduls basats en YANG encapsulats com a projectes Maven.

2.7.2. Processament d'un paquet

Després d'haver parlat dels elements que son més rellevants per a la nostra arquitectura, i d'haver comentat com es processa un paquet en Open vSwitch, ara veurem pas a pas com es processa un paquet en el controlador. Farem l'exemple del primer paquet d'un flux nou. Fixem-nos en la Figura 2.18.

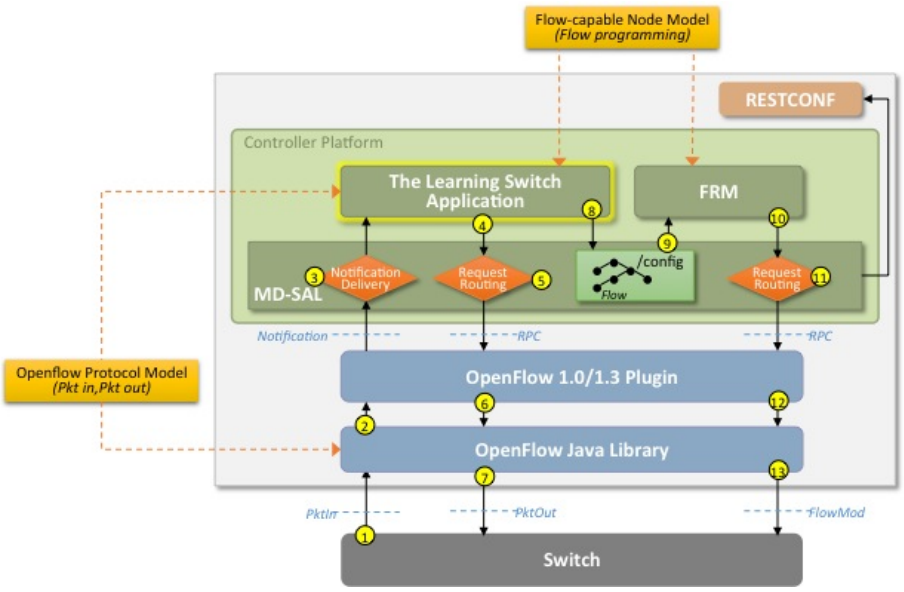

Figura 2.18: Processament del primer paquet d'un flux nou

Els passos descrits de l'1 al 3 mostren el processat d'un *Packet in*, els del 4 al 7 un *packet out* i finalment, els passos del 8 al 12 mostren la instal·lació del nou flux al *switch*.

1 S'envia un *packet in* des del *switch* al controlador.

2 La llibreria d'OpenFlow Java analitza el paquet i el tradueix en un format del MD-SAL governat pel model del protocol OpenFlow. El *plugin* del OpenFlow 1.0 crea una notificació MD-SAL i la publica al MD-SAL.

3 El MD-SAL encamina la notificació cap a tots els consumidors registrats, en aquest cas cap al *Learning Switch Application*.

4 Si la *Learning Switch Application* encara no compta amb l'assignació entre l'adreça MAC i el port, fa *flooding* amb el paquet. *Flooding* es fa com a un *pkt_out* cap al *switch*. L'aplicació emet un RPC MD-SAL (amb el paràmetre d'entrada que constitueix el camp de dades del *pkt_out*).

5 MD-SAL encamina el RPC cap al *plugin* del OpenFlow 1.0.

6 El *plugin* del OpenFlow processa el paquet, que finalment es convertit en un *Packet Out* en la llibreria OpenFlow Java.

7 El *Packet Out* és enviat al *switch*.

8 Quan l'aplicació vol programar un flux, crea un nou flux a la base de dades del MD-SAL. Aquest flux s'afegeix a la taula apropiada per relacionar-lo amb el *switch* al que es vol enviar.

9 MD-SAL genera una notificació de canvi que es rebut pel FRM (*Forwarding Rules Manager*). El FRM escolta les actualitzacions dels fluxos, és a dir, per tal de poder rebre les actualitzacions d'aquests, primer s'ha hagut de registrar al MD-SAL. Un cop registrat, cada cop que arribin canvis en els fluxos, el MD-SAL enviarà les notificacions als *bundles* que estiguin registrats, en el nostre cas, al FRM.

10 La FRM programa el flux realitzant una operació de procés remota (RPC) de fluxos del MD-SAL.

11 MD-SAL encamina la RPC cap al *plugin* del OpenFlow 1.0.

12 El plugin del OpenFlow 1.0 processa el paquet, que finalment és traduït per l'OpenFlow Java en una PDU de *FlowMod* del OpenFlow.

13 El paquet OpenFlow *FlowMod* és enviat cap al *switch*.

2.7.3. SesameForwarding OSGI Bundle

Per poder gestionar els nodes de la xarxa *wireless* controlada per SDN, s'ha implementat un nou mòdul al controlador anomenat *SesameForwarding*.

En el mòdul hem implementat un parell d'interfícies. En una d'elles ens subscrivim al servei que gestiona les estadístiques dels *switchs*. D'aquesta manera, quan arribi un paquet d'estadístiques de port, hi haurà una funció que serà cridada i ens passarà els nous paràmetres *wireless*. Així, el nostre algorisme s'aplicarà cada cop que arribi un paquet nou d'estadístiques i podrem detectar quan s'ha produït una anomalia en les condicions de l'enllaç ràdio.

La segona interfície és la que connecta el nostre mòdul amb el servei que s'encarrega de gestionar les taules d'encaminament, de tal manera que ens permetrà instal·lar la ruta calculada pel nostre algorisme al *switch* que resulti afectat.

A més, hem declarat una classe nova per emmagatzemar la llista de valors que ens arriben per poder comparar-los amb els següents. Així, podem veure l'evolució de les estadístiques i detectar anomalies a l'enllaç ràdio. Aquesta detecció la farem mitjançant el valor de *throughput* de les interfícies *wireless*, que hem hagut de calcular ja que el *driver wireless* no ens donava la taxa de velocitat instantània.

L'algorisme consisteix en agafar el valor de *throughput* anterior de la interfície *wireless* i comparar-lo amb el nou valor actual. Si la nova mostra ha decaigut un 70% respecte l'anterior, considerem que en aquella interfície hi ha una interferència que ha fet caure la taxa significativament. En aquest instant, l'algorisme mira en el node perjudicat una altra interfície *wireless* disponible i, quan la troba, crea una nova regla per a redirigir el tràfic cap a l'altra interfície. Quan el node rep la nova ruta, automàticament passa a enviar el tràfic per la nova interfície sense interrompre la transferència.

Una altra funcionalitat del nostre mòdul és que cada cop que arriba un paquet d'estadístiques de port, guarda tots els valors amb un *timestamp* d'aquell precís moment en un fitxer del tipus *CSV. D'aquesta manera, podem obtenir gràfiques de l'evolució de les estadístiques *wireless* de les interfícies en funció del temps. Aquesta informació ens serveix per poder fer l'estudi de les interferències i veure com afecta en els valors per decidir els llindars corresponents.

CAPÍTOL 3. AVALUACIÓ DEL RENDIMENT

En aquest últim capítol parlarem dels prototips i els experiments realitzats sobre la nostra arquitectura. Primerament, s'explicarà com s'ha realitzat el muntatge de l'entorn virtual i físic. Seguidament, es comentaran i s'avaluaran els resultats obtinguts amb els experiments, tot fent una comparativa amb l'extensió *mesh* del protocol 802.11.

3.1. Prototips i testbeds

Per implementar l'algorisme descrit anteriorment, necessitarem els elements comuns mostrats en la taula 3.1.

Component	**Software**	**Extensió personalitzada**
Dispositius *wireless*	Mac80211	Processament dels paquets que arriben en funció de l'algoritme
MUX	Open vSwitch	Configurat en funció de l'algoritme.
SDN Agent	Open vSwitch	Extreure a partir del driver $mac80211$ les estadístiques *wireless* i enviar-ho al controlador SDN
SDN Agent $\longleftrightarrow$ Dispositiu *wireless*	Nl80211	Cap extensió
Controlador SDN	OpenDayLight	OSGI Bundle dissenyat per controlar l'encaminament.

Taula 3.1: Elements comuns dels dos entorns

El primer element important de l'arquitectura són els dispositius *wireless* que permetran la connexió dels nodes mitjançant el protocol 802.11. Hem modificat els *divers* del $mac80211$ del *kernel* de Linux, per tal d'implementar la part de l'algoritme que s'encarrega de gestionar el tràfic etiquetat amb VLANs, per poder obtenir el destí del paquet en funció de l'etiqueta.

L'entitat MUX s'encarrega de multiplexar i desmultiplexar els paquets de múltiples connexions *wireless*, en un sol dispositiu *wireless*. Recordem que una mateixa interfície, pot estar connectada a més d'un node, obligant-nos a etiquetar el tràfic per tal d'identificar l'origen real dels paquets.

L'agent SDN és el client instal·lat en els nodes per tal de fer-los compatibles amb el protocol Openflow i així, que puguin parlar amb el controlador. En aquesta part s'ha programat una extensió que permetrà al *software* poder accedir als *drivers* de la interfície *wireless* i, obtenir les estadístiques de l'enllaç ràdio. Un cop extretes, les envia cap al controlador.

Per l'obtenció de les estadístiques dels dispositius *wireless* s'utilitza l'API del $nl80211$ (Netlink) que és la que permetrà al Open vSwitch connectar-se al *driver* $mac80211$.

Finalment, la part més important és la que engloba al controlador. És l'entitat que s'encar-

regarà de gestionar les estadístiques que s'han extret i enviat anteriorment. S'ha dissenyat un nou mòdul OSGI, per portar a terme la gestió de l'encaminament dels nodes *wireless* de manera eficient, en funció de les estadístiques que es reben dels enllaços ràdio. Cal recordar que en aquestes implementacions també hi ha els algorismes del MUX, el dispositiu *wireless* i les regles $in-band$ funcionant correctament (l'Algorisme al Apèndix A).

3.1.1. Muntatge virtual

El *testbed* virtual ens permetrà fer proves amb diverses topologies de xarxa en una única màquina. Aquest està implementat utilitzant funcions de virtualització del *kernel* de Linux com el $mac80211\_hwsim$ i els $network\ namespaces$. Amb aquestes dus eines hem sigut capaços de construir l'escenari virtual per poder posar a prova el rendiment de la nostra arquitectura. El muntatge ha estat creat sobre un portàtil Sony Vaio a 2.3 GHz amb 4 GB de RAM.

El $mac80211\_hwsim$ és un mòdul del *kernel* que simula interfícies ràdio 802.11. D'aquesta manera, hem pogut crear varies interfícies ràdio virtuals en una sola màquina per connectar-les entre sí i crear una xarxa *mesh*.

Per altra banda, amb la funcionalitat que ens proporcionen els $network\ namespaces$, hem pogut crear contenidors virtuals de configuració de xarxa diferents, amb les seves pròpies taules d'encaminament, regles de firewall i el més important, dispositius de xarxa. D'aquesta manera hem pogut distribuir les interfícies ràdio simulades amb el $mac80211\_hwsim$ en namespaces diferents, simulant com si hi haguessin diferents nodes separats amb les seves pròpies interfícies ràdio. El controlador també està iniciat en un $namespace$ diferent.

Finalment, hem interconnectat les interfícies ràdio dels diferents nodes mitjançant connexions virtuals, anomenades $veth\ pair$, per tal de crear les topologies desitjades. Cal dir que en aquest cas, totes les connexions són a través de $veth\ pair$ ja que és l'entorn virtual.

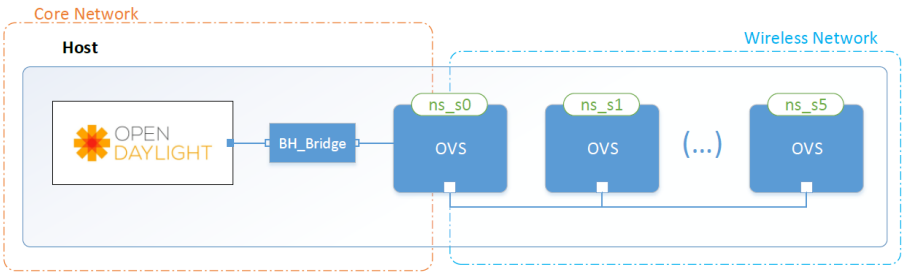

Figura 3.1: Muntatge virtual amb topologia lineal

Les topologies que s'han utilitzat son lineals, és a dir, un node darrera l'altre per poder avaluar el rendiment en funció del nombre de salts. En la Figura 3.1 tenim la topologia lineal on podem observar els elements que pertanyen a la xarxa *core* i els que pertanyen a la xarxa *wireless*. El node $s0$ és l'únic que pertany a les dues xarxes ja que connecta la xarxa *wireless* amb la core mitjançant una interfície cablejada cap al controlador. Els nodes que pertanyen a la xarxa *wireless*, com bé hem comentat en el paràgraf anterior, les seves interfícies ràdio estan connectades entre sí mitjançant $veth\ pair$.

Els experiments realitzats consisteixen en observar l'ample de banda, el retard i el nivell de senyal en topologies de 2 a 6 nodes. D'aquesta manera, es podrà observar la disminució de l'ample de banda i l'augment del retard i del nivell de senyal a mesura que augmenten els salts.

L'ample de banda el mesurarem amb l'eina $iperf$, llençant un tràfic de dades UDP amb un ample de banda i temps determinat, des del node més llunyà al node amb connexió directa al *backhaul*. Després de mesurar l'ample de banda, amb el mateix experiment i des del mateix node, es llençarà un $ping$ de 50 paquets que la mateixa comanda ens proporcionarà una mitja del retard d'aquesta transmissió. A la vegada que és fan aquests dos experiments, s'estarà emmagatzemant en una captura de $tshark$ tot el tràfic que s'ha generat. D'aquesta manera, podrem calcular tot el tràfic de senyalització generat per la nostra arquitectura en la topologia avaluada.

Cal recordar que l'entorn està simulat amb el mòdul $mac80211\_hwsim$ del *kernel* de Linux. Per tant, el màxim *throughput* en aquest cas no està limitat per la interfície *wireless*, sinó per la capacitat de processament del mòdul i de la resta de processament de la màquina física. En aquest cas no ens afecta en res, ja que amb aquests experiments volem veure la penalització que aporta la nostra arquitectura, en comparació amb el protocol *mesh* 802.11s. Per tant, estem parlant de gràfiques comparatives i no absolutes.

Cal tenir en compte que el protocol *mesh* 802.11s opera sobre una arquitectura de xarxa *wireless* clàssica, mentre que la nostra arquitectura ho fa sobre una xarxa *wireless* SDN. Com podem veure a la Figura 3.2, el node $s4$ en una arquitectura clàssica només disposa d'un camí per arribar a la xarxa core. En canvi, en una xarxa SDN, aquest mateix node està gestionat per un controlador que si les condicions de la xarxa ho requereixen, pot configurar una ruta alternativa cap a la xarxa *core*.

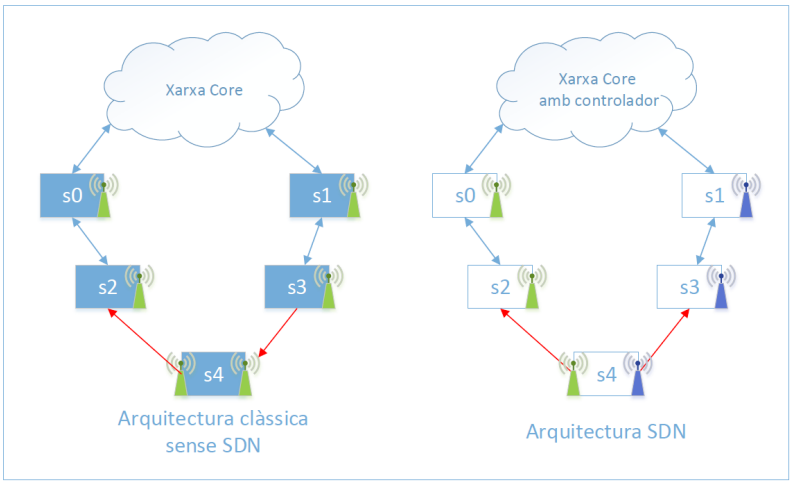

Figura 3.2: Xarxa clàssica vs. xarxa SDN

Al estar en un entorn virtual, s'ha pogut automatitzar la tasca de llençar els experiments mitjançant la crida d'un $script$. Aquest $script$ ens llença 20 vegades l'experiment sobre les diferents topologies: lineal de 2 nodes, lineal de 3 nodes... fins a 6 nodes. D'aquesta manera, s'ha pogut fer una mitja dels resultats en cada un dels escenaris i, fer que els valors siguin més acurats. Hem repetit els mateixos experiments en les mateixes condicions per el protocol 802.11s, per tal de poder fer una comparativa dels resultats.

3.1.2. Muntatge físic

El testbed físic consisteix en la implementació de totes les modificacions, abans testejades virtualment, en un entorn de *hardware* físic. Amb aquest muntatge físic es pretén demostrar els beneficis de la nostra arquitectura. En aquest entorn s'ha modificat el nostre OSGI Bundle del controlador de tal manera que implementi dues funcions.

La primera funció és guardar en un fitxer les estadístiques *wireless* de cada interfície de cada node. Aquestes estadístiques son guardades cada vegada que el controlador rep un missatge d'estadístiques de ports (cada 10 segons). Aquest fitxer és del tipus CSV per tal de poder fer un processat més ràpid de les dades i generar les gràfiques directament. Els valors que guardats en aquest fitxer CSV són els següents:

- *Timestamp* del valor.
- *Throughput* instantani.
- *Bitrate* corresponent al perfil de 80211.
- Paquets retransmesos.
- Paquets enviats.
- Nivell de senyal

A partir de les dades anteriors que rebem dels nodes de la xarxa a través del protocol OpenFlow, hem calculat el *throughput* instantani d'aquella interfície. D'aquesta manera, podem veure com es veu afectat el *throughput* en funció de les alteracions de l'enllaç ràdio.

La segona funcionalitat és un algoritme d'encaminament bàsic, que permet al nostre OSGI *Bundle* decidir quan ha d'haver un canvi de ruta cap al controlador, degut a alguna anomalia en l'enllaç que s'està utilitzant en aquell moment. Aquesta és una de les moltes funcionalitats que ens pot oferir una xarxa gestionada amb SDN.

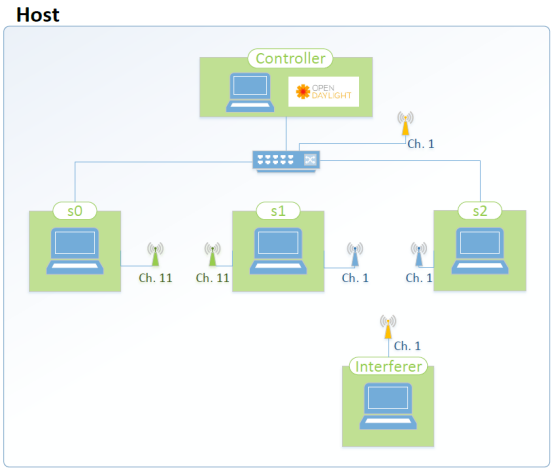

Figura 3.3: Muntatge del *testbed* físic

A la Figura 3.3 podem veure l'escenari on s'aplica l'experiment. Aquest és un escenari real on trobem una xarxa composta per 3 nodes *wireless*, dos dels quals són nodes *backhaul* que tenen connexió directa amb la xarxa *core* ($s0$ i $s2$), i un controlador que gestionarà la xarxa. El node $s1$ està connectat als dos nodes *backhaul* mitjançant dues interfícies ràdio diferents. Cada una d'aquestes interfícies treballa en un canal diferent (canal 1 i 11) per no crear interferències entre els dos enllaços.

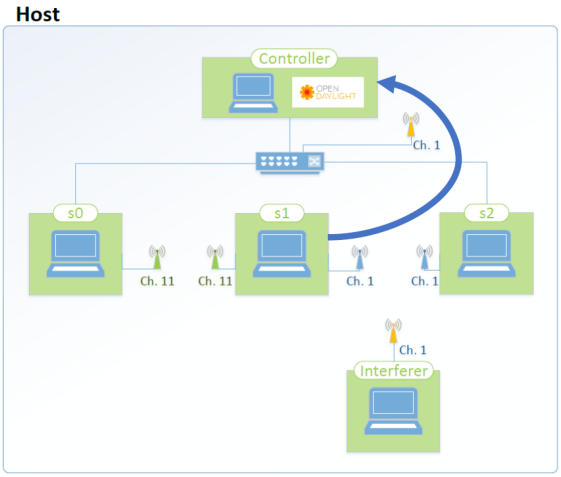

Figura 3.4: Transmissió de $s1$ a controlador

Primerament, s'arrenca tot l'escenari de tal manera que $s1$ pugui arribar a la xarxa *backhaul* a través de $s2$ mitjançant el canal *wireless* 1 (veure Figura 3.4). Un cop la connexió està establerta, s'envia una transferència UDP de $s1$ al controlador amb l'eina $iperf$, durant un temps i ample de banda determinats. Deixem el temps suficient per a que la transferència s'estabilitzi abans de realitzar el següent pas en l'experiment.

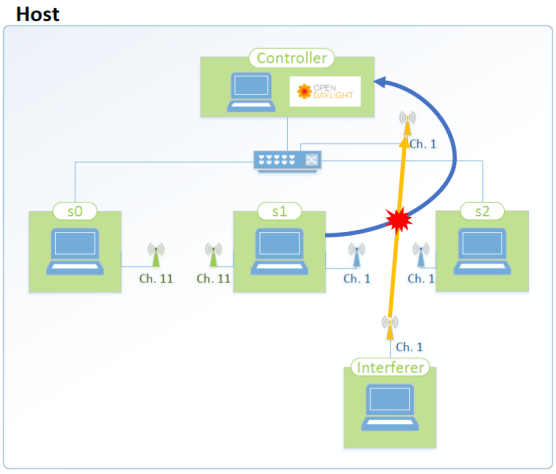

Figura 3.5: Transmissió del interferent al *bridge*

A continuació despleguem un dispositiu *wireless* al costat de l'escenari que enviarà un flux de dades interferent (Figura 3.5). Aquesta transferència s'envia cap al *bridge*[1] utilitzant

[1] El *bridge* que hem utilitzat per a realitzar l'escenari és un Router AP amb el *firmware* modificat (OpenWRT)

també el canal 1, de tal manera que provoca col·lisions a l'enllaç ràdio entre $s1$ i $s2$ i disminueix el *throughput* a causa d'aquesta interferència.

Quan al controlador li arriba el paquet d'estadístiques amb els valors actuals, és aleshores quan el controlador detecta una caiguda prou important del *throughput* en el node $s1$ respecte al *throughput* que havia guardat anteriorment. Automàticament, el controlador comprova les interfícies *wireless*, (ports per a la visió del Open vSwitch), que te disponibles $s1$ i troba una altra interfície *wireless* disponible. Seguidament, el controlador construeix la nova ruta per a instal·lar-la a $s1$. La nova ruta consisteix en fer passar el tràfic de la transferència per l'altra interfície *wireless* que ha trobat disponible el controlador (Figura 3.6). Al cap d'uns segons, s'observa com el *throughput* de la interfície wireless que connecta $s1$ amb $s0$ comença a augmentar, mentre que el de la interfície que connecta $s1$ amb $s2$ comença a disminuir.

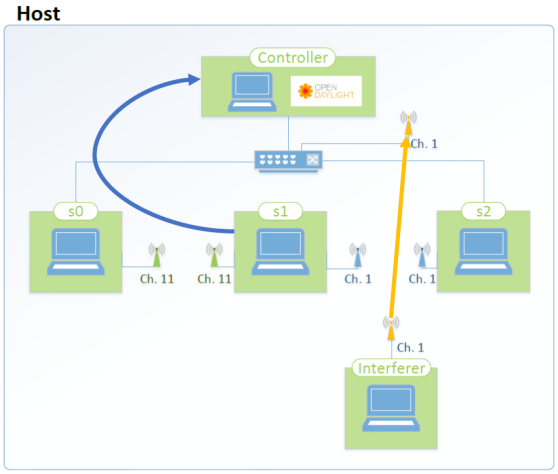

Figura 3.6: Canvi de ruta cap al controlador

3.2. Resultats obtinguts

3.2.1. Testbed virtual

El primer resultat que compararem serà el *throuhput* obtingut del 80211s i la nostra arquitectura en l'escenari virtual. Com podem veure a la Figura 3.7, tenim una gràfica de la taxa de velocitat màxima aconseguida amb els dos mètodes en funció del nombre de nodes que recorre aquesta transferència.

Veiem que el 80211s està per sobre de la nostra arquitectura. Per tant, tenim una penalització en els resultats de màxima transferència. Recordem que el límit del *throughput* no l'imposen les interfícies *wireless*, sinó el *hardware* de l'ordinador on fem les simulacions. La diferència del *throughput* és deguda al processament d'una còpia del paquet a l'entitat del MUX i, al tràfic de senyalització del produït pel SDN.

El segon resultat que hem calculat en la nostra arquitectura i, el protocol 80211s ha estat el retard. En la Figura 3.8, podem veure la diferència del retard de transmissió en funció del nombre de nodes. Observem que el retard de la nostra arquitectura és lleugerament

més alt. Aquest augment prové, com veurem després en el tercer resultat, del nivell de senyalització del protocol OpenFlow.

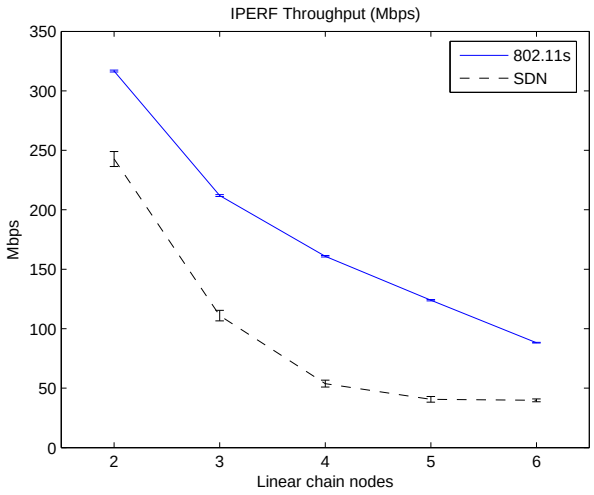

Figura 3.7: *Throughput* de SESAME vs 80211s

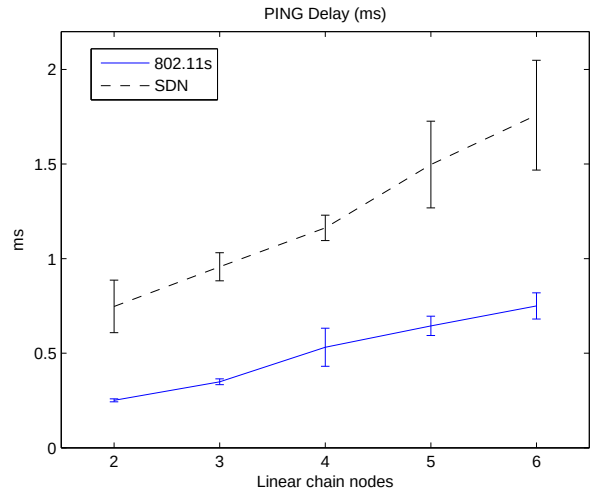

Figura 3.8: Retard de SESAME vs 80211s

Finalment, el tercer resultat que hem calculat ha estat el nivell de senyalització OTA (*Over-The-Air*) de l'arquitectura. En la Figura 3.9, veiem que el nivell de SESAME és més alt que el del protocol 80211s. Això és degut a que el protocol OpenFlow transporta més tràfic de senyalització que el protocol 802.11s, fent que ocupi més ample de banda del canal i un augment del retard.

Cal dir però, que el tràfic de senyalització circula pel mateix canal que els paquets de dades al ser un medi inalàmbric. En el cas de la nostra arquitectura, no només hi ha la senyalització convencional d'OpenFlow, sinó que també hi circulen les estadístiques dels enllaços ràdio. Per això ocasiona un augment del retard i una disminució del throughput respecte el protocol 802.11s.

Les estadístiques del OpenFlow s'envien cada 10 segons. Tot i que hi ha més missatges de senyalització, cal dir que aquests s'envien a la velocitat de les dades i no a una velocitat

de 1 Mbps, com passa amb la senyalització del protocol 80211. Això es degut a que el tràfic de senyalització SDN es un tràfic TCP normal per a la capa *wireless*.

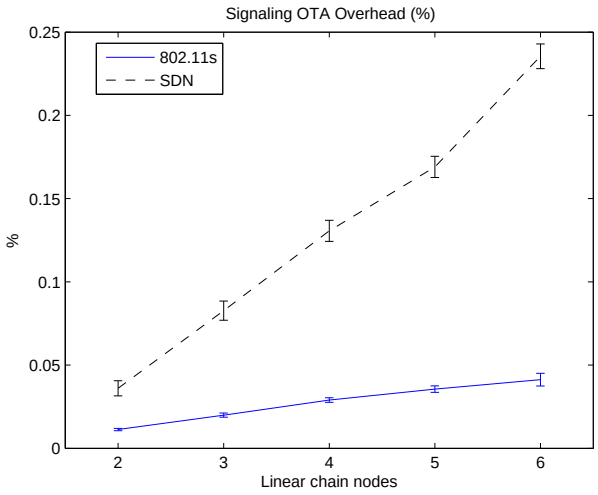

Figura 3.9: Senyalització de SDN vs 80211s

Una de les possibles solucions que podríem aplicar per a disminuir la senyalització i el retard seria augmentar el temps d'enviament de les estadístiques. No obstant, hem d'anar en compte ja que l'encaminament del tràfic és fa en funció de les estadístiques *wireless*. Per tant, no pot ser gaire alt degut a que el temps de reacció davant d'alguna anomalia en l'enllaç ràdio seria massa alt.

En un principi es va plantejar l'opció de fer els mateixos experiments variant el temps d'enviament d'estadístiques, i així poder dibuixar les gràfiques una a sobre de l'altra per poder veure com afectava el nivell de senyalització a la resta de paràmetres com el *throughput* i el retard. D'aquesta manera es podria haver deduït si la solució d'augmentar o reduïr el temps d'enviament d'estadístiques era factible o no. Això però, es deixa per a futures millores del projecte ja que aquests canvis no eren implementables a curt termini.

3.2.2. Testbed físic

Un cop hem analitzat els resultats virtuals per poder veure el rendiment que te la nostra arquitectura, hem passat a fer les proves funcionals. En aquest experiment, hem comprovat com el controlador SDN gestiona els nodes en funció de les seves estadístiques *wireless* i, com canvia la ruta del node $s1$ cap al controlador per un canal que està lliure.

Primer hem fet l'experiment sense que el controlador detecti la caiguda de *throughput* provocada per la interferència. En la Figura 3.10, podem veure com comença la transferència de tràfic UDP de $s1$ al controlador. Al cap de 3 minuts veiem que entra la interferència causant una disminució de la velocitat de la taxa de transferència de $s1$ al controlador. Finalment, quan la interferència s'acaba, el *throughput* es recupera a poc a poc. Com no hi ha cap canvi de ruta per part del controlador, ja que hem desactivat la part de canvi de ruta del nostre *bundle*, observem com el *throughput* de la interfície que connecta $s1$ amb $s2$, pateix una caiguda molt significativa durant els 3 minuts que dura la transferència de la interferència.

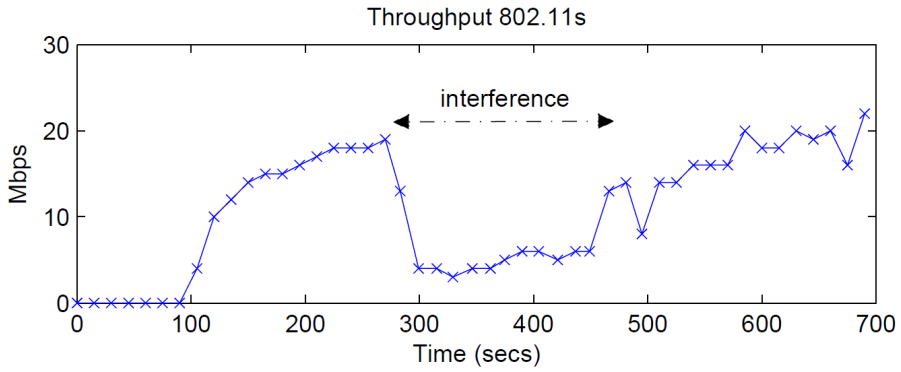

Figura 3.10: *Throughput* de $s1$ sense SesameForwarding

A continuació, hem fet el mateix experiment habilitant el nostre OSGI *bundle*. A la Figura 3.11 podem comprovar en quin moment l'algoritme del nostre mòdul detecta la interferència. En aquest instant, tal hi com hem explicat en la secció 3.1.2., el controlador construeix la nova regla d'encaminament i l'envia a $s1$ per instal·lar-la, fent el canvi de ruta. Amb aquest canvi de ruta, veiem que el tràfic passa a sortir per la interfície que connecta $s1$ amb $s0$ utilitzant el canal 11 que està lliure d'interferències. Per tant, podem observem que la transferència es podria considerar gairebé contínua.

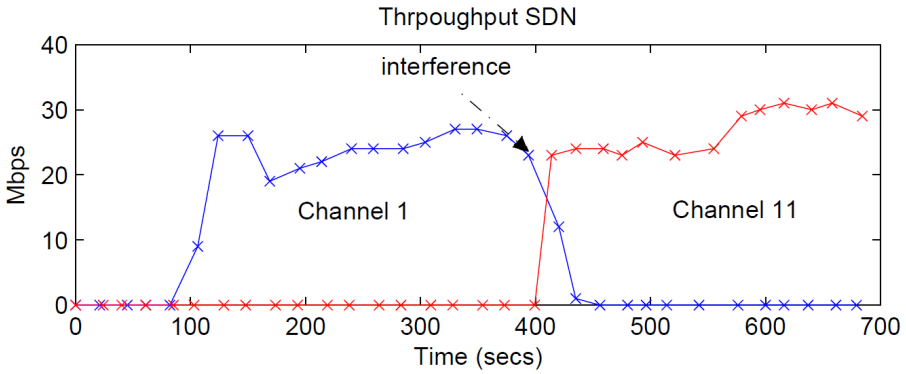

Figura 3.11: *Throughput* de $s1$ amb SesameForwarding

Per altra banda, en la Figura 3.12 podem veure l'evolució que ha patit la taxa de bit de les dues interfícies *wireless* del dispositiu $s1$ durant l'experiment. Veiem que la taxa de bit de l'enllaç $s1 \longleftrightarrow s2$ -que pertany al canal 1-, es manté més o menys constant fins arribar a 400 segons que és quan es produeix la interferència. Aleshores, pateix una forta caiguda degut a la interferència fins que es produeix el canvi de ruta. Tot i així, veiem que li costa recuperar-se ja que el canal 1 tenia una alta ocupació en l'entorn on s'han realitzat les proves físiques. En canvi, la interfície que connecta $s1 \longleftrightarrow s0$ pateix oscil·lacions però alhora també es manté constant durant més temps. Veiem que al voltant dels 400 segons té un pic de taxa de bit i després cau perque necessita adaptar-se a la transferència que transporta.

Finalment, a la figura 3.13 tenim la gràfica dels nivells de senyal de les dues interfícies de $s1$ abans esmentades. Podríem dir que no varia gaire el nivell d'una interfície respecte de l'altra, ja que cal tenir en compte que l'escala de la gràfica és relativament petita.

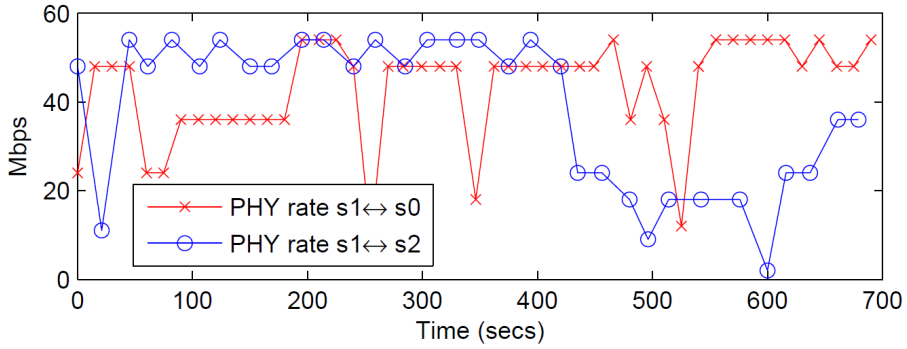

Figura 3.12: *Bitrate* dels enllaços $s1$-$s0$ i $s1$-$s2$

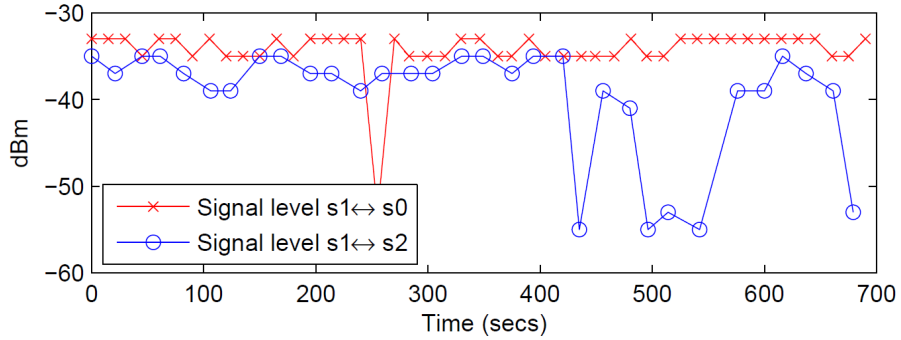

Figura 3.13: Nivell de senyal dels enllaços $s1$-$s0$ i $s1$-$s2$

3.3. Conclusions dels resultats

Un cop hem vist les funcionalitats que aporta la nostra arquitectura i els resultats obtinguts respecte les solucions actuals, intentarem valorar si les penalitzacions que tenim utilitzant la tecnologia SDN, son compensades pels avantatges que ens ofereix.

Els resultats obtinguts sobre l'eficiència de SESAME no han estat gaire favorables respecte el protocol 80211s. La diferència en el *throughput* és degut a que en una arquitectura SDN ja hi ha més intercanvi d'informació en quant a nivell de senyalització que el 80211s. Si afegim que hem extés un missatge del protocol OpenFlow per a que pugui transportar les estadístiques dels enllaços ràdio, veiem clarament que el *throughput*, el retard i el nivell de senyalització seran pitjors. Ja que ocupa l'ampla de banda i congestiona més el canal. Ara bé, la penalització que ens introdueix aquesta extensió del protocol OpenFlow, ens val la pena ja que ens proporciona beneficis que amb el 80211s no tenim.

No obstant, com hem dit anteriorment, el *throughput* dels resultats no estava limitat pels dispositius *wireless* sinó pel *hardware* de l'ordinador on es simula, ja que el *mac80211 hwsim* envia els paquets simulats a velocitat màxima. Considerem que el camí més llarg que pot recórrer un dispositiu en una xarxa en la ciutat és de 5 o 6 salts. Per tant, com que el resultat de la taxa de transferència d'aquesta distància és aproximadament uns 50 Mbps, considerem que és suficientment alt com per a suportar un perfil de taxa de bit del 802.11 a 54 Mbps. Per tant, la nostra arquitectura podria ser implementada en una xarxa de infraestructura del tipus 802.11g.

Per altra banda, en les proves de l'entorn físic hem comprovat les funcionalitats que ens

ofereix una xarxa SDN. Una d'elles és l'encaminament dinàmic del tràfic en funció de les estadístiques proporcionades pels dispositius *wireless*. Amb el *testbed* físic hem pogut comprovar que, la commutació de camí s'efectua en un temps relativament curt un cop es detecta alguna condició que interfereix en el camí d'una transferència. El protocol 802.11s no té una gestió de múltiples camins de sortida de la xarxa *wireless*, per tant, estaríem davant d'un gran avantatge que resoldria el balanceig de càrrega de la densificació de xarxa a la ciutat. Cal dir que la funcionalitat que hem implementat és una d'entre moltes que ens pot oferir la tecnologia SDN. No hem d'oblidar que, tots els beneficis que ens aporta tenen un objectiu comú, millorar la qualitat de servei a banda de guanyar altres beneficis.

El nostre mòdul del controlador es capaç de recollir l'estat i les estadístiques dels dispositius *wireless*. Una de les altres possibilitats que en un futur es poden implementar és emmagatzemar en una base dades del controlador tota aquesta informació. D'aquesta manera, podrà ser utilitzada per poder fer estudis de tràfic, qualitat de servei, inventari dels nodes... Podent així, optimitzar els recursos energètics apagant nodes que no són necessaris en determinats moments del dia i, fent-los despertar quan sigui necessària la seva presència degut a les condicions de la xarxa.

Al ser SDN una arquitectura de gestió centralitzada, ens permet tenir un control més estricte dels dispositius ja que disposem d'una visió global de la xarxa i a més, tenim emmagatzemades les estadístiques de les condicions dels enllaços ràdio, permetent-nos optimitzar el funcionament dels nodes. D'aquesta manera, aconseguim un impacte ambiental favorable, ja que contribuïm a l'estalvi d'energia de la ciutat.

Com podem veure, la nostra arquitectura ens ofereix més funcionalitats que els protocols actuals i alhora seguir en la línia d'optimitzar els recursos per aconseguir una ciutat intel·ligent.

CONCLUSIONS

En aquest projecte hem proposat una possible arquitectura SDN per a ser implementada en una xarxa *wireless backhaul*. Hem dissenyat, implementat i verificat la nostra arquitectura en una situació real avaluant el seu rendiment.

Hem analitzat la problemàtica actual: l'excés de dispositius en la xarxa i l'evolució prevista per aquesta en els pròxims anys. S'ha arribat a la conclusió de que ens hem de plantejar una nova gestió de la xarxa que ens permeti controlar aquest excés de dispositius.

En base a la problemàtica analitzada, hem proposat una arquitectura que pretén fer convergir les xarxes mòbils i la xarxa fixa, amb la intenció de gestionar tots els dispositius de manera eficient. Hem utilitzat la tecnologia de *Small Cells* que ens permet la densificació de la xarxa mitjançant la reducció de la mida de la cel·la i l'augment de la taxa de transmissió per usuari. Això comporta un control més estricte de la xarxa degut a l'augment del nombre de cel·les i a la quantitat d'usuaris connectats a elles. La tecnologia SDN ens permet prendre el control de tots els dispositius, gràcies a l'externalització de gairebé totes les funcionalitats a nivell de *software*, donant-nos flexibilitat a l'hora d'integrar les funcions.

La nostra arquitectura està construïda a través de diferents mòduls. El primer consta de nodes *wireless* dotats d'un agent SDN, que ens permetrà el control i la gestió tant del tràfic com dels dispositius. També enviarà les estadístiques dels enllaços ràdio al controlador. El segon mòdul és un controlador SDN a la xarxa *core* que analitzarà les estadístiques dels nodes i, en funció d'aquestes, decidirà els camins de sortida dels dispositius.

Un cop realitzada la implementació, hem validat l'arquitectura emulant i simulant situacions i entorns reals. En un escenari virtual, hem avaluat els resultats del rendiment en comparació amb el protocol 802.11s. Hem vist que el rendiment ha estat lleugerament més baix en quant a *throughput*, retard i nivell de senyalització. No obstant, el rendiment que hem obtingut és suficient per a que l'arquitectura pugui ser implementada amb el protocol 802.11g (a 54 Mbps).

Finalment, en un escenari real amb dispositius físics hem muntat una topologia bàsica per fer una prova de funcionalitat. Hem dissenyat un mòdul en el controlador que ens permet fer un balanceig de càrrega, en funció de les condicions del enllaços ràdio dels nodes. Aquesta funcionalitat és una de les moltes millores per guanyar qualitat de servei que ens permet una xarxa SDN.

S'ha realitzat una publicació en col·laboració amb Daniel Camps Mur i Sebastià Sallent Ribes sobre SESAME al IEEE ICC'15 (*International Conference on Communications*), com una possible solució a la gestió de les xarxes *wireless backhaul* mitjançant SDN, anomenada "*SDN Wireless Backhauling for Small Cells*".

BIBLIOGRAFIA

[1] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013–2018, disponible a: *http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html* 3

[2] Qualcomm, *Rising to Meet the 1000x Mobile Data Challenge*, disponible a: *https://www.qualcomm.com/media/documents/files/rising-to-meet-the-1000x-mobile-data-challenge.pdf* 6

[3] Ericsson Zero Site: Outdoor small cell site solution, disponible a: *http://www.ericsson.com/ourportfolio/products/zero-site* 6

[4] Bernardos, C.J.; De La Oliva, A; Serrano, P.; Banchs, A; Contreras, L.M.; Hao Jin; Zuñiga, J.C., *An architecture for software defined wireless networking,* Wireless Communications, IEEE , vol.21, no.3, pp.52,61, Juny 2014 12

[5] Lalith, S.; Schulz-Zander, J.; Merz, R.; Feldmann, A.; Vazao, T., *Towards Programmable Enterprise WLANs With Odin,* in Proc. Workshop on Hot Topics in Software Defined Networking (HotSDN '12), 2012. 12

[6] Schulz-Zander, J.; Sarrar, N.; Schmid, S., (August 2014). *Towards a scalable and near-sighted control plane architecture for WiFi SDNs.* In Proceedings of the third workshop on Hot topics in software defined networking (pp. 217-218). ACM. 12

[7] Dely, P.; Kassler, A.; Bayer, N., *OpenFlow for Wireless Mesh Networks,* IEEE International Workshop on Wireless Mesh and Ad Hoc Networks (WiMAN 2011), Hawaii, USA, August 2011. 12

[8] HP, Bridging the data center of today and tomorrow with SDN, disponible a: *http://h17007.www1.hp.com/docs/networking/datacenter/4AA5-1865ENW-Discover-FAQ.PDF* 13

[9] Google, A software Defined WAN Architecture, disponible a: http://www.opennetsummit.org/archives/apr12/vahdat-wed-sdnstack.pdf 13

[10] ONF Wireless and Mobile Working Group, disponible a: *https://www.opennetworking.org/working-groups/wireless-mobile* 18

[11] The OpenFlow Switch Specification. Disponible a: *https://www.opennetworking.org* 18

[12] Documentació del mac80211. Disponible a: *http://wireless.kernel.org/en/developers/Documentation/mac80211?action=AttachFile&do=get&target=mac80211.pdf* 21

[13] Sherwood, R.; Gibb, G.; Yap, K.K.; Appenzeller, G.; Casado, M.; McKeown, N.; Parulkar, G. (2009). *Flowvisor: A network virtualization layer*. OpenFlow Switch Consortium, Tech. Rep. 22

[14] De Ghein, Luc. *MPLS Fundamentals*, Cisco Press, 2006. 23

[15] Hiertz, G. R.; Denteneer, D.; Max, S.; Taori, R.; Cardona, J.; Berlemann, L.; Walke, B. (2010). *IEEE 802.11 s: the WLAN mesh standard*. Wireless Communications, IEEE, 17(1), 104-111. 23

[16] Vipin, M.; Srikanth, S. (January 2010). *Analysis of open source drivers for IEEE 802.11 WLANs*. In Wireless Communication and Sensor Computing, 2010. ICWCSC 2010. International Conference on (pp. 1-5). IEEE. 23

[17] La versió dels backports-3.12-1 de Linux disponible a: *https://www.kernel.org/pub/linux/kernel/projects/backports/stable/v3.12.1/* 25

[18] The Openvswitch project. Disponible a: *http://openvswitch.org/*

[19] The OpenDayLight Project, disponible a: *http://www.opendaylight.org/* 30

[20] Wiki del OpenDaylight Controller, disponible a: *https://wiki.opendaylight.org/*

APÈNDIXS

APÈNDIX A. ALGORISME DE L'ARQUITECTURA

Algorithm 3: Mux/Demux and Inband setup algorithms

Variables:
$A \leftarrow$ SDN agent
$D \leftarrow$ wireless device
$MUX \leftarrow$ MUX function in Figure 2.2
$e_{MD} \leftarrow$ Ethernet device connecting the MUX to D
$l \leftarrow$ Single wireless link characterized by $LLID/PLID/neighbour\_address$
$L \leftarrow$ List of all known wireless links by device D
$parent\_port \leftarrow$ Parent OF port in A for inband set up
$child\_ports \leftarrow$ List of child OF ports in A for inband setup
$IP_{ctrlr} \leftarrow$ IP address of the SDN controller C
$IP_{sw} \leftarrow$ IP of this wireless switch A

Executed in the MUX
Add e_{MD} to MUX's datapath
$L \leftarrow$ Poll D for wireless links
for $l \in L$ **do**
    $(e_1, e_2) \leftarrow$ Create new veth pair to represent link l
    Add e_1 to A's datapath
    Add e_2 to MUX's datapath
    $rule_{mux} \leftarrow in = e_2 \Rightarrow out = e_{MD}$, push VLAN=$l.LLID$
    $rule_{demux} \leftarrow in = e_{MD}, VLAN = l.LLID \Rightarrow out = e_2$, pop VLAN
    Install $rule_{mux}$ and $rule_{demux}$ to MUX

Executed in the wireless device D
//Upon having to transmit a packet P
for $l \in L$ **do**
    if $P.VLAN = l.LLID$ **then**
        $P.VLAN = l.PLID$
        $P.dst\_addr = l.neighbour\_address$

Executed in the SDN agent A **for inband setup**
// Rules dealing with ARP to/from controller
$arp_1 \leftarrow prot = arp, arp\_tpa = IP_{ctrlr} \Rightarrow out = parent\_port$
$arp_2 \leftarrow prio = 2, prot = arp, arp\_tpa = IP_{sw} \Rightarrow out = LOCAL$
$arp_3 \leftarrow prio = 1, prot = arp, arp\_spa = IP_{ctrlr} \Rightarrow out = child\_ports$
// Rules dealing to IP to/from controller
$ip_1 \leftarrow prot = ip, ip\_dst = IP_{ctrlr} \Rightarrow out = parent\_port$
$ip_2 \leftarrow prio = 2, prot = ip, ip\_dst = IP_{sw} \Rightarrow out = LOCAL$
$ip_3 \leftarrow prio = 1, prot = ip, ip\_src = IP_{ctrlr} \Rightarrow out = child\_ports$
Install arp_1, arp_2, arp_3, ip_1, ip_2 ip_3 in A