DISSENY DEL CORE NETWORK D' UN SISTEMA AUTÒNOM CONTROLAT DE MANERA CENTRALITZADA

Director: German Santos Boada

27 DE ENERO DE 2022

PAU COMA NINOT

Contenido

1. Context

1.1 Introducció

Actualment, vivim en una societat totalment connectada. Internet és una part fonamental de les nostres vides, i és estrany el dia en que no el fem servir, ja sigui cercant informació o comunicant-nos per xarxes socials.

Com a enginyers informàtics, és crucial que coneguem com funciona internet, els seus sistemes i protocols, per així poder contribuir en la millora de les xarxes per fer Internet més eficient. Actualment ens trobem en un estancament en l' evolució dels protocols de xarxa, i les xarxes SDN i el protocol OpenFlow poden suposar un canvi brutal en el disseny de xarxes, revolucionant així Internet.

Actualment, la configuració de les xarxes es realitza integrant el software dins de cada component que composa la xarxa, i OpenFlow proposa un paradigma totalment diferent, en el qual la configuració dels diferents components es faci a través d' un node centralitzat, que contingui tota la informació i que es comuniqui amb cada component.

En aquest treball de final de grau estudiarem com funcionen les xarxes centralitzades per software, així com el protocol OpenFlow. Veurem com s' integren una en l' altre, i comprovarem si aquests nous protocols poden donar-nos un servei eficient adequat per les xarxes actuals d' Internet.

El projecte de modalitat A es situa en el marc de la Facultat d' Informàtica de Barcelona, en l' especialització de Tecnologies de la Informació del grau d' Enginyeria Informàtica

1.2 Motivació del projecte

La motivació del projecte ve donada per d' afany de coneixement que sorgeix amb l' estudi de les xarxes convencionals. Com a alumne que està cursant 4t d' Enginyeria Informàtica he fet varies assignatures on hem estudiat les xarxes convencionals, els seus protocols i els seus models.

Així doncs, el pas següent és buscar nous protocols i sous paradigmes que ens empenyin a millorar les xarxes actuals.

Crec que és un tema força interessant, que em donarà una visió diferent de les xarxes estudiades fins ara, i em permetrà seguir aprenent sobre aquest tema.

1.3 Problema

La xarxa d' un ISP actual ha de mantenir una certa eficiència, ja que hi ha enrutadors que mantenen taules de forwarding de milers d' entrades, i ha de mantenir també una certa qualitat de servei.

Avui en dia, la qualitat de servei es manté a través del protocol MPLS, que és el protocol més utilitzat per aquesta funció. El problema és que cada vegada hi ha taules de forwarding més grans, i això ens obliga a buscar nous paradigmes que ens donin una eficiència més elevada.

1.4 Objectius

Abans de començar amb l' explicació del projecte, enumeraré una sèrie d' objectius que intentarem cobrir en aquest projecte:

- Conèixer i entendre les mancances de les xarxes convencionals i les causes de l' aparició d' OpenFlow.
- Obtenir un coneixement general sobre el protocol OpenFlow.
- Familiaritzar-me amb l' eina MININET i ser capaç de programar una topologia simple.
- Implementar un escenari OpenFlow en el qual es vegi el funcionament del protocol i de les xarxes SDN.
- Comprovar que una xarxa SDN pot complir amb els requisits de QoS dels serveis especificats a la xarxa NEBA.

1.5 Metodologia

Aquest treball es desenvoluparà amb un programa de simulació de xarxes, que és Mininet.

Mininet ens permetrà definir xarxes SDN fent servir el protocol OpenFlow.

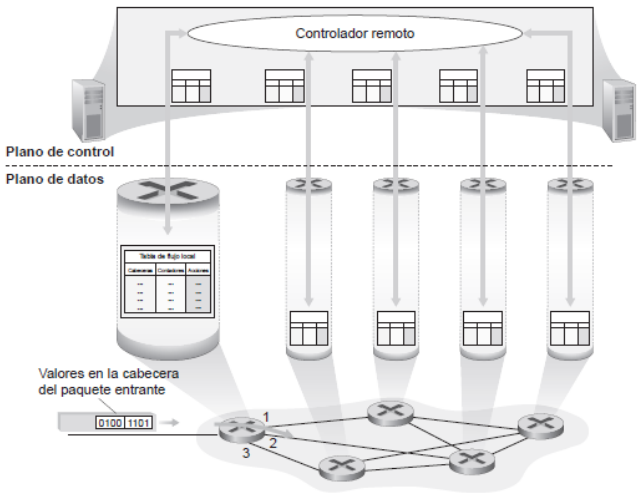
2. SDN

Les xarxes definides per software són un conjunt de tècniques l' objectiu del qual és facilitar la implementació de serveis de xarxa de manera dinàmica i escalable.

Utilitzen un enfocament pensat per optimitzar els recursos disponibles utilitzant software per prioritzar i ordenar el trànsit de la xarxa, considerant mètriques com la velocitat, latència i consum.

Podem identificar quatre característiques fonamentals d' una xarxa SDN:

- **Reenviament basat en flux**: En una xarxa SDN el reenviament de paquets pot estar basat en valors de diferents camps de les capçaleres de les capes de transport, de xarxa o d' enllaç. Això xoca frontalment amb el model tradicional, en el qual el reenviament de paquets es basa exclusivament en la direcció destí del paquet. En SDN les regles de reenviament de paquets estan especificades en la taula de flux d' un switch, això ens diu que es responsabilitat del pla de control SDN calcular, gestionar o instal·lar les entrades de les taules de flux en tots els commutadors de xarxa.
- **Separació del pla de control i el pla de dades**: El pla de dades està composat per commutadors de xarxa, dispositius relativament ràpids però eficaços que executen les regles de les taules de flux. El pla de control està compost per servidors i software que determinen les taules de flux dels commutadors.



- **Funcions de control de xarxa:** La "S" de SDN vol dir "Software", així que no resulta sorprenent que el pla de control SDN estigui implementat en software. A diferència del model tradicional, aquest software s' executa en servidors que són remots i diferents dels commutadors de xarxa. EL pla de control consta de dos components: el controlador SDN i un conjunt d' aplicacions de control de xarxa. El controlador SDN s' encarrega de mantenir l' estat de la xarxa en tot

moment, mentre que les aplicacions de xarxa s' executen en el pla de control i s' encarreguen de monitoritzar, programar i controlar els dispositius de la xarxa. Normalment veiem el controlador com un únic servidor central, però això només és cert en la teoria. En la pràctica el controlador consta de varis servidors diferenciats que proporcionen prestacions coordinades i escalables.

- **Una xarxa programable:** La xarxa es pot programar mitjançant les aplicacions que s' executen en el pla de control. Aquestes aplicacions utilitzen les API proporcionades pel controlador SDN.

2.1 Història

Encara que l' enorme interès envers les xarxes SDN sigui molt recent, la realitat és que el paradigma SDN i la separació entre pla de dades i pla de control daten de l' any 2004. En aquest any, diverses veus van argumentar a favor de la separació de plans de control i de dades de la xarxa.

Podem dividir la història de SDN en tres etapes: xarxes actives (1995-2000), separació del pla de control i el pla de dades (2001-2007 aproximadament) i aparició d' OpenFlow (2007-2010).

2.1.1 Xarxes actives

A principis dels anys 90 va sorgir internet, i això va produir un avenç tecnològic que va portar als investigadors a dissenyar i provar noves aplicacions, entre elles protocols de xarxa.

Alguns investigadors van perseguir una idea d' obertura del control de xarxes. Com que les xarxes convencionals no són programables, van sorgir les xarxes actives, orientades al control de xarxa, conceptualitzant una API que exposes els recursos en nodes de xarxa individuals.

L' impuls tecnològic de les xarxes actives va permetre reduir el cost computacional, avançar en llenguatges de programació i en la tecnologia de màquines virtuals.

Tot i que les xarxes actives no van tenir gaire èxit, van oferir contribucions relacionades amb SDN, com per exemple funcions programables de xarxa, virtualització de xarxa, etc.

2.1.2 Separació del pla de control i el pla de dades

A principis dels anys 2000, la necessitat de xarxes més robustes, predictibles i manejables, i l' augment del tràfic va portar als investigadors a buscar enfocaments més moderns per a certes funcions dins la gestió de xarxes.

Els commutadors i enrutadors convencionals tenien un problema, i és que tenien una estreta integració als plans de control i dades, i per resoldre aquest problema, la idea de separar els dos plans va començar a sorgir.

Degut al creixement d' Internet, els ISPS van començar a separar el pla de dades del pla de control amb l' objectiu de poder gestionar les seves xarxes creixents i poder aportar als clients protocols que les fessin més segures, com VPN.

Això va dur a terme diferents protocols nous, com per exemple RCP o PCE, ambdós conceptes clau en dissenys futurs d' SDN.

2.1.3 Aparició d' OpenFlow

A mitjans dels anys 2000, els investigadors i les empreses van guanyar interès en investigar sobre control lògic centralitzat. Un dels fruits d' aquestes investigacions va ser el projecte 4D, que establia quatre capes diferenciades. Nombrosos grups d' investigació van començar recerques de sistemes basats en aquest enfocament. Principalment, va sorgir el projecte Ethane, que posteriorment es convertiria en la base de la API d' OpenFlow.

Gràcies a l' adopció d' OpenFlow a les empreses, que van obrir les seves API per permetre als programadors controlar certs comportaments de reenviament, la versió inicial del protocol es va establir en els commutadors a través d' una simple actualització de firmware, sense haver de canviar el hardware.

Encara que OpenFlow faci servir els principis de molts dels seus predecessors, també innova en molts aspectes, per exemple la generalització de dispositius de xarxa, la visió d' un sistema operatiu de xarxa i tècniques de gestió distribuïda de l' estat de dispositius de xarxa entre altres.

2.2 Per què SDN?

Les xarxes tradicionals no són capaces de fer front als requeriments de telecomunicacions actuals, i les empreses i investigadors estan buscant noves maneres d' aprofitar al màxim les seves xarxes. Tot i així, cal trobar nous protocols i nous paradigmes per fer front a aquest problema. Les limitacions de les xarxes actuals inclouen:

- **Complexitat:** Les xarxes cada cop són més grans i mes complexes, i davant d' aquesta realitat, les empreses han millorat els protocols de xarxa per ser més eficients, per poder acomodar les xarxes a les necessitats dels usuaris en general.
- **Polítiques incoherents:** Les xarxes tradicionals donen massa feina als administradors de xarxa, que es veuen obligats a configurar milers de mecanismes i dispositius cada cop que un nou dispositiu entra a la xarxa.
- **Poca escalabilitat:** És inevitable que la xarxa vagi creixent cada vegada més, i amb el creixement també augmenta la complexitat d' aquesta amb la suma de cents de milers de dispositius nous que han de ser configurats i gestionats.

El creixement de les xarxes durant els últims anys estan portant la xarxa existent al límit, i necessitem noves xarxes que puguin assumir les necessitats següents:

- **Heterogeneïtat en els patrons de trànsit:** a diferència del paradigma client-servidor, en les que gran part de la comunicació es dóna entre client i servidor, les aplicacions modernes creen trànsit màquina a màquina mitjançant l' accés a bases de dades i servidors. A més, el poder connectar-se des de qualsevol punt usant les xarxes inalàmbriques està canviant els patrons de trànsit.
- **Augment de carga de treball dels administradors de la xarxa:** Nous dispositius com ara telèfons intel·ligents i tabletes estan començant a accedir a la xarxa, i això dóna feina extra als administradors de la xarxa a dl' hora de protegir les dades dels usuaris i mantenir la seguretat de les xarxes.
- **Augment de serveis basats en el núvol:** Aquest augment, degut a la gran acollida d' aquests serveis per part dels gran públic i les empreses, fa que tinguem una necessitat d' augmentar l' agilitat d' accés a aplicacions, infraestructures i altres recursos de telecomunicacions. Això requereix una escalabilitat dinàmica de la capacitat de computació, emmagatzematge i recursos de xarxa.
- **Big Data:** El processament del big data requereix processament massiu per part de m olts servidors.

SDN ens dóna resposta a totes aquestes necessitats, atorgant una manera flexible de controlar-les i el potencial de revolucionar tot el món de les xarxes.

2.2.1 Beneficis

A més d' oferir xarxes programables centralitzades que poden atendre les necessitats de les empreses, SDN proveeix els beneficis següents:

- **Redueix el CAPEX:** Quan parlem de CAPEX ens referim als béns que adquireix una empresa que tenen una vida útil que va més enllà de l' any natural. Es refereix a equipaments, propietats o edificis industrials. En informàtica, parlem de CAPEX quan ens referim a ordinadors, servidors, commutadors, etc. Així doncs, una SDN redueix el CAPEX ja que ens permet reutilitzar el hardware existent, i redueix la necessitat de invertir en hardware nou.
- **Redueix l' OPEX:** Quan parlem d' OPEX ens referim als costs permanents per al funcionament d' un producte, negoci o sistema. Per exemple, la compra d' una impressora suposa un cost en CAPEX, mentre que el paper, la tinta i l' energia necessaris per fer-la funcionar entraria en l' OPEX. També entraria en l' OPEX el cost dels treballadors que gestionen una xarxa, per això podem dir que SDN redueix en OPEX, ja que fa mes senzilla la configuració i gestió de les xarxes i permet reduir el temps de gestió per part dels administradors i redueix la possibilitat d' error humà.
- **Agilitat i flexibilitat:** SDN permet que les empreses puguin desplegar aplicacions, serveis i infraestructures ràpidament.

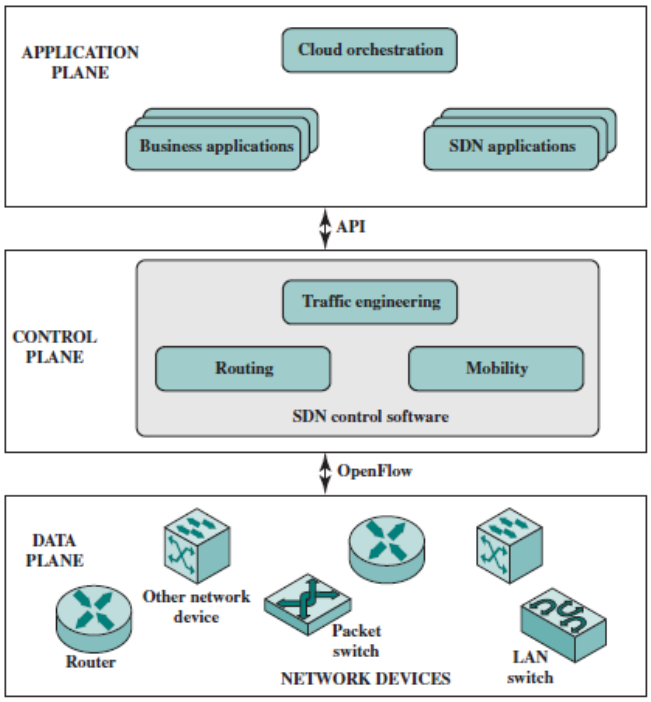
- **Permet innovació:** Permet crear nous models de negoci i nous tipus d' aplicació per part de les empreses.

2.3 Arquitectura

L' arquitectura de SDN consisteix en una xarxa composada per Switchos en la qual hi ha un controlador centralitzat encarregat de dur a terme totes les funcions complexes de la xarxa, com són enrutament, "naming" i seguretat. Això composa el pla de control de SDN, que consisteix en un o més servidors SDN.

El controlador SDN defineix els fluxos de dades que es duen a terme al pla de dades. Cada flux de dades ha de sol·licitar el vistiplau del controlador SDN, el qual verifica que la comunicació és possible segons les polítiques de la xarxa. Si el controlador permet un flux, aquest afegeix una entrada a cada Switch per el flux corresponent.

La comunicació entre el controlador i els Switchos usa un protocol estandarditzat i una API. El protocol més utilitzat és OpenFlow, del qual parlarem més endavant.



La arquitectura SDN és bastant flexible: pot operar amb diferents tipus de Switchos i a diferents capes de protocol.

En una estructura SDN, un Switch du a terme les següents funcions:

1. El switch encapsula i envia el primer paquet del flux al controlador SDN, i espera una resposta del controlador que diu si el flux ha de ser afegit a la taula de flux del switch.
2. El switch fa "forwarding" del paquets entrants pel port corresponent basant-se en la seva taula de flux.

3. El switch pot descartar paquets d' un flux determinat, temporal o permanentment, tal com indiqui el controlador.

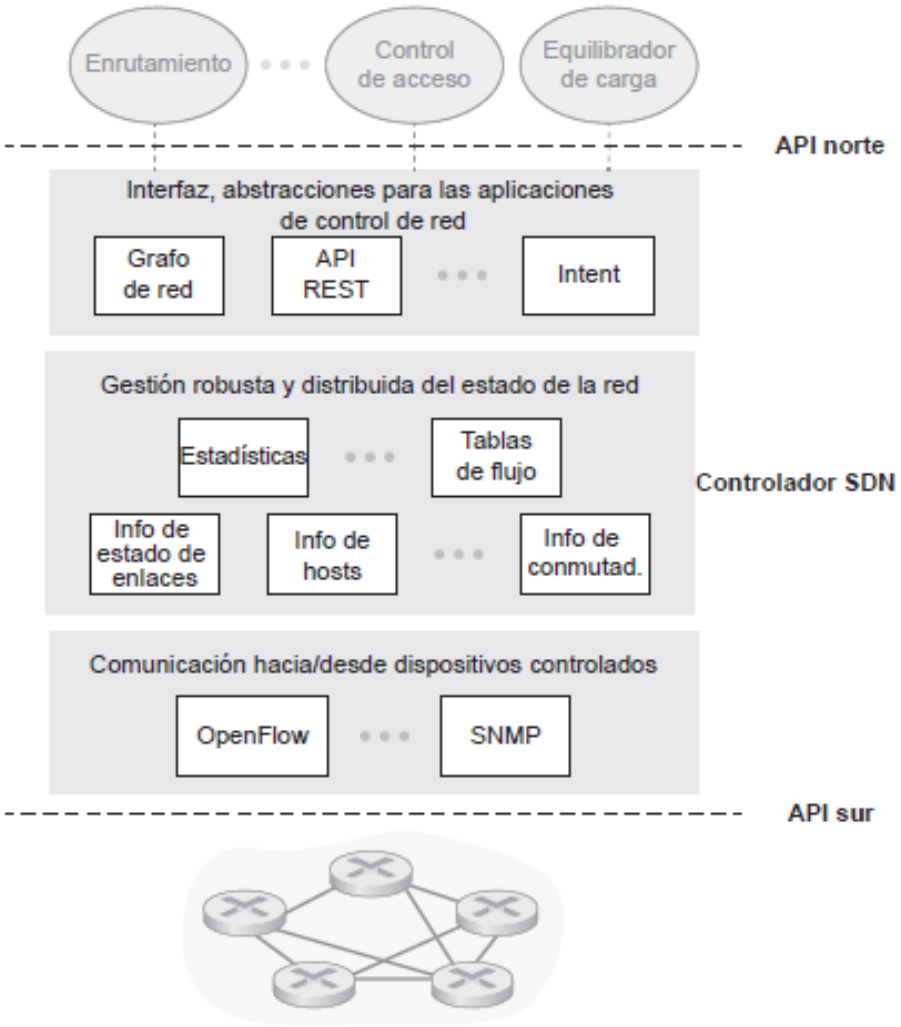
2.3.1 Pla de control

En aquesta secció veurem en què consisteix el pla de control SDN. Veurem la lògica de la xarxa que controla el reenviament de paquets entre els dispositius de la xarxa SDN, així com la configuració i la gestió dels dispositius i serveis.

Com hem vist a l' apartat "SDN", el pla de control es divideix en dues parts: el controlador SDN i les aplicacions de control de xarxa.

Començarem veient en què consisteix un controlador SDN. La funcionalitat d' un controlador SDN pot dividir-se en tres capes.

- **Capa de comunicacions:** Està clar que si un controlador ha de controlar el funcionament d' un commutador, necessitem un protocol que ens permeti enllaçar els diferents commutadors amb el controlador SDN. A més a més, un dispositiu de xarxa ha de ser capaç de comunicar al controlador els diferents esdeveniments que és capaç de detectar. Aquests esdeveniments són els que proporcionaran al controlador una visió actualitzada del estat de la xarxa. Aquest protocol constitueix la capa inferior de l' arquitectura del controlador. El protocol més extens per realitzar aquest tipus de comunicació és el protocol OpenFlow, el qual veurem més endavant.
- **Capa de gestió de l' estat de la xarxa:** Per poder gestionar la xarxa SDN, és necessari que el controlador disposi d' informació actualitzada sobre l' estat dels hosts, enllaços, commutadors, etc. La taula de flux d' un commutador conté comptadors que poden ser utilitzats per les aplicacions del control de xarxa. Un controlador pot mantenir una còpia de les taules de flux de cada commutador, per fer ús dels comptadors de manera més eficaç.
- **Capa interfície amb les aplicacions de xarxa:** El controlador SDN interactua amb les aplicacions de xarxa a través de la seva API nord. Aquesta API permet a les aplicacions llegir i escriure l' estat de la xarxa i les taules de flux dins la capa de gestió de l' estat. Les aplicacions poden registrar-se per rebre avisos quan es produeixin canvis d' estat per poder prendre accions, canviant les taules de flux o amb altres accions.



2.3.2 Pla de dades

Anteriorment, hem comentat que les decisions d' un enrutador tradicional es basen únicament en la direcció de destí dels paquets. Podem caracteritzar l' enviament basat en el destí com un procés en dos passos. El primer seria buscar la direcció IP destí (correspondència) i el segon seria enviar el paquet al port de sortida especificat, a través del entramat de commutació (acció).

Quan parlem de SDN, considerem un paradigma "correspondència-acció" bastant més general, en el qual la correspondència pot buscar-se a múltiples camps de la capçalera associats a diferents protocols en diferents capes. L' acció consisteix en reenviar el paquet per un o més ports de sortida.

Cada commutador de xarxa consta d' una taula "correspondència-acció" que és calculada, instal·lada i actualitzada per un controlador remot.

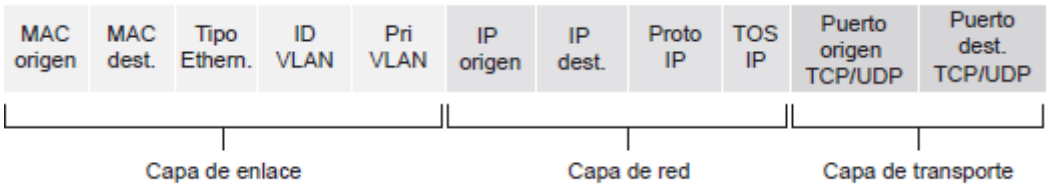
A partir d' ara, per realitzar l' explicació suposarem que el controlador utilitza el protocol OpenFlow, que es el protocol més extens per treballar en SDN.

Cada entrada de la taula de reenviament "correspondència-acció" que es coneix amb el nom de "taula de flux" en OpenFlow, inclou:

- **Un conjunt de valors de camps de capçalera**: Els farà servir per buscar una correspondència en un paquet entrant. Un paquet que no correspongui amb cap entrada de la taula de flux pot ser eliminat o enviat al controlador remot per un processament addicional.
- **Un conjunt de comptadors:** Aquests comptadors s' actualitzen a mesura que es troben correspondències de paquets amb entrades de la taula de flux. Podrien incloure el nombre de paquets pels quals s' ha trobat correspondència amb les entrades de la taula, i el temps transcorregut des de que l' entrada de la taula es va actualitzar per primera vegada.
- **Un conjunt d' accions que s' han de prendre:** les accions que s' han de prendre quan un paquet es correspon amb una entrada de la taula de flux. Aquestes accions podrien consistir en enviar el paquet per un port de sortida corresponent, eliminar el paquet, enviar-lo a múltiples ports o reescriure certs camps de la capçalera.

2.3.2.1 Correspondència

Aquests són els 11 camps de la capçalera dels paquets amb els quals es pot buscar una correspondència en OpenFlow 1.0:



Podem veure que la correspondència en OpenFlow 1.0 es pot fer en tres capes de capçalera de protocol, violant així el principi de separació de capes. Així doncs, un dispositiu compatible amb OpenFlow pot comportar-se tant com un enrutador de paquets (nivell 3) com un commutador de paquets (nivell 2).

El conjunt d' 11 valors que permeten fer buscar correspondència en OpenFlow 1.0 ha augmentat fins a 41 en les especificacions d' OpenFlow més recents.

Les entrades de la taula de flux també admeten caràcters comodí. Per exemple, una direcció IP de 128.120.*.* en una taula de flux correspondrà amb tot el camp de direcció rellevant de qualsevol datagrama que tingui com a primers 16 bits la direcció de valor 128.120. Cada entrada de la taula de flux té associada també un nombre de prioritat. Si un paquet entrant coincideix amb múltiples entrades de la taula de flux, es seleccionarà l' entrada de major prioritat.

2.3.2.2 Acció

Les accions més importants que es poden dur a terme són les següents:

- **Reenviament:** un paquet entrant pot ser enviat per un sol port de sortida, per tots els ports de sortida menys el port d' entrada, o per un conjunt de ports de sortida. El paquet també pot ser enviat al controlador remot corresponent. Si això succeeix, el controlador pot canviar les taules de flux dels commutadors i retornar el paquet al dispositiu perquè el tracti amb la nova informació.
- **Eliminació:** Una entrada de la taula de flux sense cap acció ens indica que el paquet corresponent a aquella entrada ha de ser eliminat.
- **Modificació de camps:** Els valors de deu camps de la capçalera del paquet poden ser reescrits abans de reenviar el paquet al port de sortida.

3. OpenFlow

Necessitem dues coses perquè una xarxa SDN sigui funcional.

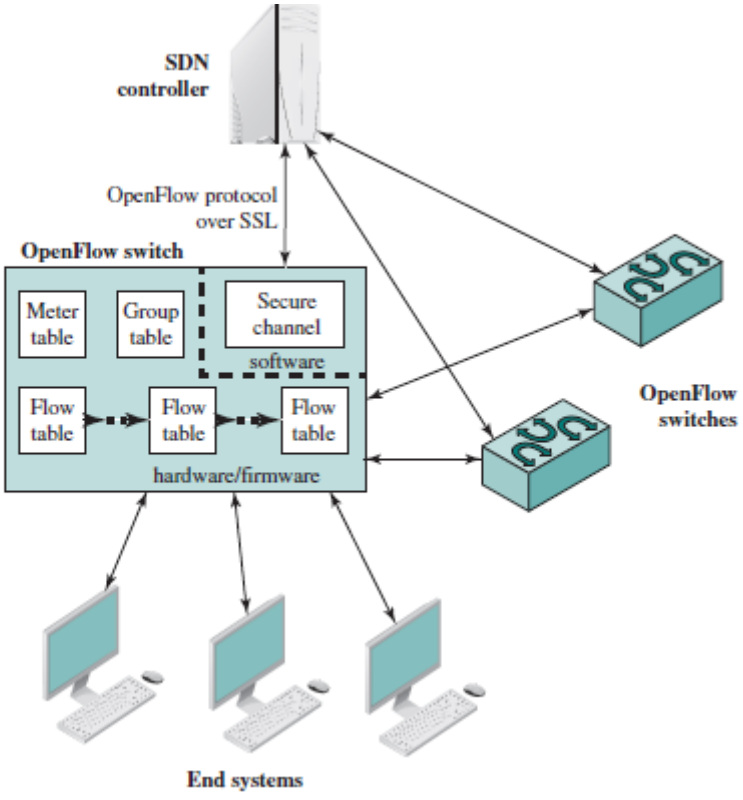
La primera és una estandardització dels commutadors, enrutadors i altres dispositius de xarxa. Necessitem que tinguin la mateixa estructura lògica, cosa que no és fàcil perquè cada venedor programa els dispositius de manera diferent.

La segona és un protocol efectiu que sigui capaç de comunicar-se amb els diferents dispositius que composen la nostra xarxa.

Aquests dos problemes són adreçats per OpenFlow, que actua tan com a protocol de comunicació entre controlador i dispositius, com d' estandardització de l' estructura lògica dels commutadors.

3.1 Arquitectura del commutador

Un controlador SDN es comunica amb els commutadors utilitzant el protocol OpenFlow a través de Secure Socket Layer (SSL).



Cada commutador es connecta amb altre commutadors, amb el controlador i amb ordinadors, que fan d' origen i destí dels fluxos.

El protocol OpenFlow especifica tres tipus de taula en l' estructura lògica del commutador. La primera és la ja vista taula de flux, que relaciona els paquets entrants

al commutador amb un flux concret i determina les accions que es duen a terme per a cada flux. Pot haver múltiples taula de flux que actuïn a mode de tub.

Una taula de flux pot enviar un flux a la taula de grup, que pot activar una varietat d' accions que afectin a un o més fluxos.

Una taula de metre pot activar una varietat d' accions relacionades amb el rendiment en un flux.

3.1.2 Taules de Flux

La taula de flux és l' estructura bàsica que composa un commutador OpenFlow. Quan un paquet entra al commutador, passa per una o més taules de flux. Cada una d' aquestes taules de flux es composa d' entrades amb sis components:

- **Camps de coincidència:** S' usen per seleccionar els paquets que coincideixen amb els valors dels camps.
- **Prioritat:** Indica la prioritat de les entrades de les taules.
- **Comptadors:** L' especificador d' OpenFlow defineix una sèrie de comptadors. Per exemple el nombre de bytes i paquets rebuts per port, per taula de flux o per entrada de la taula; el nombre de paquets eliminats o la duració d' un flux.
- **Instruccions:** Les accions que es duen a terme quan hi ha una coincidència amb una entrada de la taula.
- **Time-out:** Màxim temps d' inactivitat abans que un flux expiri.
- **Cookie:** És un conjunt de dades triades pel controlador, que es fan servir per filtrar estadístiques dels fluxos, etc. No es fa servir per processar paquets.

Per últim, una taula de flux pot contenir una entrada final que filtri tots els paquets que no tinguin cap coincidència amb les entrades anteriors. Aquesta última entrada té prioritat 0.

3.1.2.1 Accions a dur a terme

Al apartat "Acció" de la part de "SDN" hem comentat per sobre les accions que es poden dur a terme quan hi ha una coincidència amb una entrada de la taula de flux. Ara veurem amb més detall el set d' instruccions que es duen a terme quan hi ha una coincidència. Abans de parlar dels tipus d' instruccions, cal definir els termes "acció" i "set d' accions". Una acció descriu el reenviament de paquets, la modificació de paquets i el processament de les taules de grup. Mentre que el "set d' accions" és una llista d' accions que estan associades amb cada paquet que s' acumulen mentre el paquet està sent processat per cada taula i que s' executen quan el paquet surt del tub de processament.

L' especificació d' OpenFlow inclou les següents accions:

- **Sortida:** Reenvia el paquet al port especificat.

- **Set-Queue:** Col·loca un identificador de cua a un paquet. Quan un paquet és reenviat a un port usats l' acció de "sortida", l' identificador de cua indica quina cua associada al port es fa servir per programar i reenviar el paquet. El comportament del reenviament el dicta la configuració de la cua i es fa servir per proveir Qos bàsic.
- **Grup:** Es processa el paquet al grup especificat.
- **Push Tag/Pop Tag:** El fa "push" o "pop" d' una etiqueta VLAN o MPLS.
- **Set-Field:** Es fa servir per modificar els valors de determinats camps de les capçaleres del paquet.
- **Canviar-TTL:** Es fa servir per canviar els camps: IPv4 TTL, IPv6 Hop Limit o MPLS TTL.

Els "set d'accions" que es poden dur a terme són els següents:

- **Direccionar paquet a través del tub:** Es fa servir per enviar un paquet a una taula més avançada al tub.
- **Realitzar acció al paquet:** Es fa servir per realitzar una acció al paquet quan coincideix amb una entrada de la taula.
- **Actualitzar el "set d' accions":** Fusiona les accions especificades amb el "set d' accions" pel paquet corresponent, o neteja totes les accions del "set d' accions".
- **Actualitzar metadades:** Un valor de metadades pot ser associat amb un paquet. Pot ser usada per portar informació d' una taula a una altra.

3.1.2.2 Tub de taules de flux

Un commutador consta de una o més taules de flux. Si n' hi ha més d' una, s' organitzen en forma de tub o "pipeline", amb les taules etiquetades del 0 en endavant.

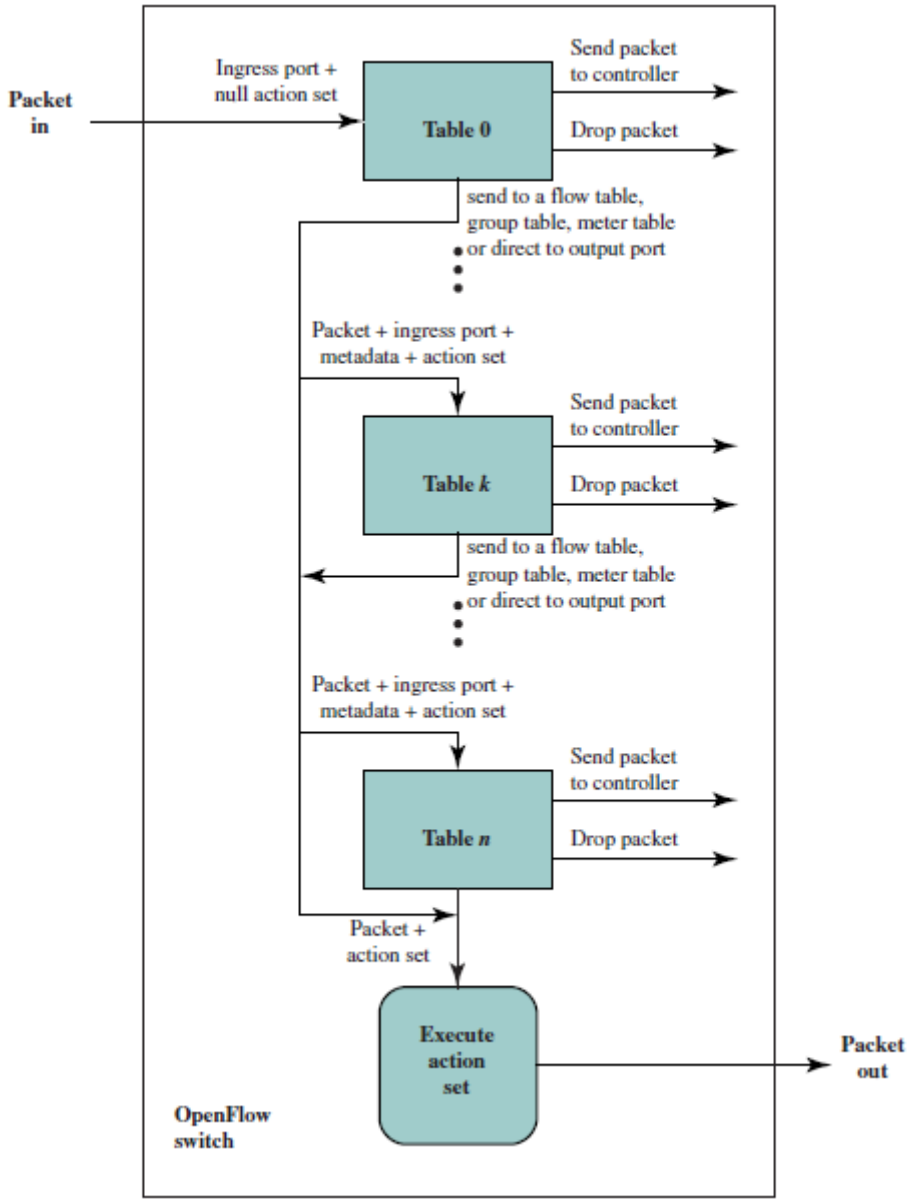
Quan un paquet entra a una taula per ser processat, l' entrada consisteix en el paquet en si, la identitat del port d' entrada, les metadades associades i el "set d' accions" associat.

El processament es duu a terme de la següent manera:

1. Busca l' entrada coincident amb la prioritat més alta. Si no coincideix amb cap entrada i no hi ha entrada final, llavors el paquet és eliminat. Si només hi ha una coincidència amb l' entrada final, llavors aquesta entrada especifica una d' entre aquestes tres accions: la primera és enviar el paquet al controlador. Això permetrà al controlador un nou flux per aquest paquet i similars, o decidirà si eliminar el paquet. La segona acció que pot dur a terme és direccionar el paquet a una altra taula de flux més avançada al tub i la tercera acció que es pot fer és eliminar el paquet.
2. Si hi ha una coincidència en alguna de les entrades que no sigui l' entrada final, llavors agafem la coincidència amb la prioritat més alta. Tot seguit es duen a

terme les següents accions: actualitzar tots els comptadors associats amb aquesta entrada, executar totes les instruccions associades amb aquesta entrada i direccionar el paquet a la taula de flux següent o al port de sortida.

Quan un paquet és enviat al port de sortida, llavors el "set d' accions" acumulades són executades i el paquet és enviat a la cua de sortida.



3.2 Missatges d' OpenFlow

Entre els missatges més importants que flueixen des del controlador fins el commutador tenim els següents:

- **Configuració:** Aquest missatge permet al controlador consultar i configurar els paràmetres de configuració del commutador.

- **Modificar estat:** El controlador envia aquest missatge per afegir, esborrar o modificar les entrades de la taula de flux del commutador, i també per configurar els ports del commutador.
- **Llegir estat:** El controlador fa servir aquest missatge per recopilar dades i estadístiques dels comptadors dels ports i taules de flux del controlador.
- **Enviar paquets:** El controlador fa servir aquest missatge per enviar un paquet a través d' un port de sortida especificat del commutador. El missatge conté el paquet a enviar.

La comunicació entre controlador i commutador també es pot donar en direcció contrària. Entre els missatges més importants que es donen de commutador a controlador hi ha els següents:

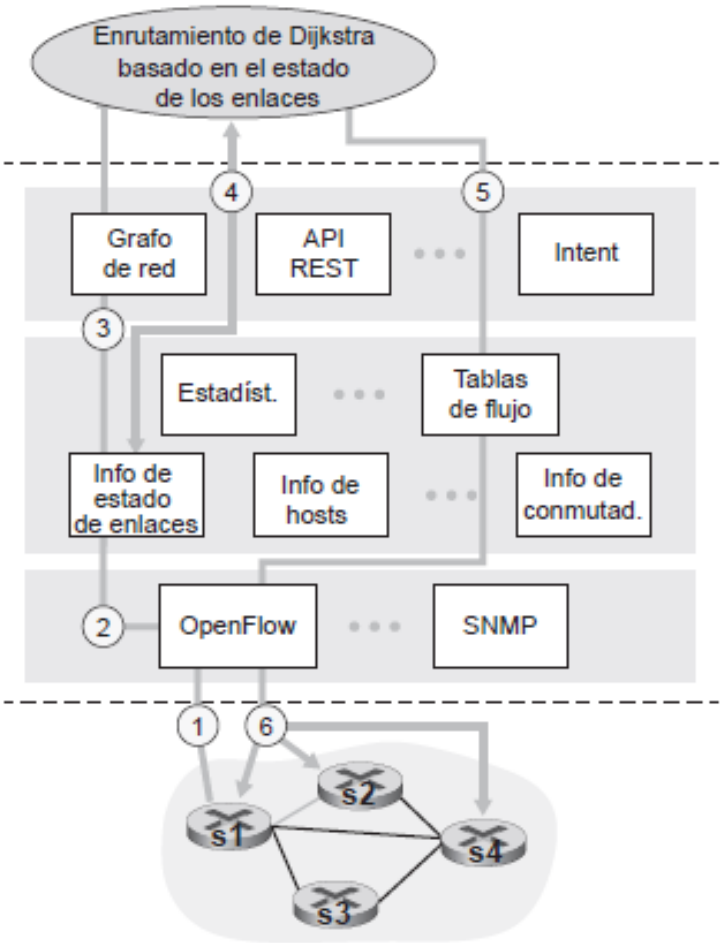
- **Flux eliminat:** Aquest missatge indica al controlador que s' ha eliminat una entrada de la taula de flux, ja sigui perquè s ha rebut un "modificar estat" o perquè ha finalitzat un temporitzador.
- **Estat de port:** El commutador envia aquest missatge per notificar un canvi en l' estat d' un port.
- **Paquet entrant:** Com hem vist anteriorment, si arriba un paquet a un commutador i aquest no correspon amb cap entrada de la taula de flux, aquest s' envia al controlador. El missatge "paquet entrant" s' utilitza per enviar aquests paquets al controlador.

3.3 Interacció entre pla de dades i pla de control

Un cop hem vist el pla de dades i el pla de control de SDN, i hem estudiat com es comporta el protocol OpenFlow, veurem un exemple que il·lustra perfectament la interacció entre el pla de dades i el pla de control.

Considerem l' ús de l' algorisme de Dijkstra per determinar el camí més curt entre dos punts. Aquest algorisme s' executa com una aplicació separada, fóra dels commutadors de paquets.

En aquest exemple, tenim quatre commutadors nombrats de s1 fins a s4, i suposem que l' enllaç entre s1 i s2 es desactiva. Suposem també que està implementat un enrutament basat en la ruta més curta, pel qual les regles de reenviament de fluxos entrants i sortints en s2, s1 i s4 es veuran afectades, i suposem per últim que la operació de s3 no es veu afectada. Fem servir OpenFlow com a protocol de comunicació entre commutadors i controlador.



1. El commutador s1, al experimentar una fallada entre ell mateix i s2, notifica el canvi d' estat del enllaç al controlador SDN, fent ús del missatge "Estat de port" d' OpenFlow.
2. El controlador SDN rep el missatge OpenFlow que indica el canvi en l' estat de l' enllaç i envia una notificació al gestor de l' estat dels enllaços, que actualitza una base de dades d' estats d' enllaços.
3. L' aplicació de control de xarxa que implementa l' enrutament de Dijkstra basat en l' estat dels enllaços, rep la notificació del canvi en l' estat de l' enllaç.
4. L' aplicació d' enrutament basada en l' estat dels enllaços interactua amb el gestor de l' estat dels enllaços per obtenir informació actualitzada de l' estat dels enllaços. Després, calcula les noves rutes de cost mínim.
5. L' aplicació d' enrutament interactua amb el gestor de taules de flux, que determina quines taules de flux s' han d' actualitzar.
6. El gestor de taules de flux utilitza llavors OpenFlow per actualitzar les entrades de les taules de flux dels commutadors afectats.

4. QoS

La qualitat de servei (Quality Of Service) és un conjunt de tecnologies que treballen conjuntament en una xarxa per oferir les millors qualitats en una capacitat de xarxa limitada. Les tecnologies QoS aconsegueixen això proveint capacitats diferenciades als diferents fluxos en el trànsit de la xarxa.

Els paràmetres més importants de qualitat de servei existents són: la taxa de transferència efectiva (throughput), la latència (delay) i la fluctuació de retard (delay jitter).

La qualitat de servei és particularment important en el trànsit en temps real, com per exemple Voice over IP (VoIP), videoconferències, o vídeo sota demanda, que tenen una tolerància mlt baixa als canvis en latència o la fluctuació de retard.

4.1 Necessitat de QoS

La QoS neix de la necessitat de donar un tracte especial a un cert tipus d' informació que viatja a través d' una xarxa.

Com hem mencionat anteriorment, avui en dia s' ofereixen serveis com telefonia IP, videoconferències, vídeo sota demanda, streaming, etc. i cada un d' aquests serveis necessita un tracte especial depenent de les seves característiques. La QoS és l' encarregada de determinar el tracte adequat que se li ha de donar a cada flux d' informació mitjançant certs mecanismes i tècniques d' analitzar el tipus de trànsit que viatja a través de la xarxa.

Una de les grans dificultats que ofereix la prestació d' aquests nous serveis sobre la xarxa de dades és el maneig de la congestió. Existeixen serveis que requereixen un gran ample de banda, i encara que el medi físic tingui la capacitat de proveir-lo, els alts nivell de congestió el degraden tant que és difícil poder oferir aquests serveis.

4.2 Paràmetres de QoS

4.2.1 Pèrdua de paquets

La pèrdua de paquets (packet loss) es dóna quan un o més paquets de dades viatjant en una xarxa no arriben al seu destí. La pèrdua de paquets es pot produir per errors en la transmissió de dades, típicament en xarxes inalàmbriques o per congestió a la xarxa.

La pèrdua de paquets acceptable en una xarxa depèn del tipus de dades que estan sent enviades. Per exemple, en serveis VoIP, un o dos paquets perduts no afectaran a la comunicació, però pèrdues d' entre el 5 i el 10% pot afectar significativament a la comunicació.

En canvi per streaming d' àudio o vídeo, menys d' un 1% es considera bo, i entre un 1 i un 2,5% es considera acceptable.

4.2.2 Latència

La latència o retard és el temps de trànsit que necessiten els paquets en recórrer la distància, des de l' origen al destí. És un problema de les xarxes de telecomunicacions, sovint amb enllaços lents o congestionats.

La latència en trànsit de temps real com per exemple streaming de música o vídeo hauria de ser inferior als 150 ms. L' oïda humana és capaç de detectar latències d' uns 200 ms. Si es supera aquest límit, pot ser que el servei deixi de ser adequat.

No existeix una solució que es pugui implementar de manera senzilla. Moltes vegades depèn dels equips pels qual passen els paquets. Si el problema de la latència està a la nostra pròpia xarxa, podem augmentar l' ample de banda o velocitat d' enllaç, o prioritzar aquests paquets dins de la nostra xarxa.

4.2.3 Taxa de transferència efectiva

La taxa de transferència efectiva (throughput) és la velocitat real de transport de dades a través d' una xarxa. Normalment es mesura en megabits per segon i sempre serà inferior al ample de banda.

La taxa de transferència efectiva pot ser afectada per varis factors, incloent les limitacions del medi físic de transmissió, la velocitat de processament dels dispositius de xarxa, etc.

L' ample de banda de les comunicacions és limitat i acostuma a ser compartit per nombroses aplicacions.

Si tenim problemes d' ample de banda, podem: augmentar l' ample de banda de les xarxes per les quals circulen les nostres comunicacions. També podem reduir el consum que facin altres aplicacions de l' ample de banda.

4.2.4 Fluctuació de retard

La fluctuació de retard (delay jitter) és un efecte de les xarxes no orientades a connexió i basades en commutació de paquets. Com que la informació es divideix en paquets, cada paquet pot seguir un camí diferent per arribar al seu destí. Així doncs, la fluctuació es defineix com la variació en el temps en l' arribada de paquets, causada per pèrdua de sincronització, congestió de xarxa o per els diferents camins que els paquets han pres per arribar al destí.

Les comunicacions en temps real són especialment sensibles a aquest efecte. L' espera que l' augment de mecanismes de QoS com prioritat a les cues, reserva d' ample de banda o enllaços de major capacitat puguin reduir els problemes de fluctuació.

La fluctuació entre el punt inicial i final hauria de ser menor a 100 ms. Si el valor és menor a 100 ms la fluctuació pot ser compensada de manera adequada.

La solució més adoptada ara mateix és la utilització del buffer de fluctuació. Consisteix en assignar una petita cua per anar rebent els paquets i servint-los amb un petit

retard. Si algun paquet no està al buffer quan es necessiti, aquest es descarta. Un augment de buffer implica menys pèrdua de paquets però més retard. Una disminució implica menys retard però m´ñes pèrdua de paquets.

4.3 Xarxa NEBA

NEBA vol dir “Nou Ethernet de Banda Ampla” i és un servei majorista de banda ampla d’ accés indirecte que proporciona un accés amb entrega restringida a una localitat que podrà contractar-se tant sobre la xarxa d’ accés de coure, com les tecnologies ADSL2+, VDSL2, com sobre la xarxa de fibra FTTH.

Dit amb altres paraules, es tracta d’ un servei que permet que els altres operadors ofereixin un servei de connexió a internet als seus clients en zones on no compten amb la seva pròpia infraestructura de xarxa. NEBA és una obligació imposada a Telefònica per compartir les seves xarxes.

4.3.1 Tipus de servei

NEBA ofereix 3 qualitats de servei diferents, que són “Real Time”, ORO i “Best Effort”.

4.3.1.1 Best Effort (BE)

El mecanisme “best effort” designa un tipus de servei de xarxa que no pot garantir que les dades arribin al seu destí, ni oferir al usuari una determinada qualitat de servei en les seves comunicacions.

En una xarxa “best effort” tots els usuaris reben el millor servei possible en aquell moment, el que significa que obtindran diferents amples de banda i temps de resposta en funció del volum de trànsit a la xarxa.

4.3.1.2 ORO

Aquest tipus de trànsit és orientat al trànsit d’ empreses. És prioritari respecte al “best effort” i tindrà associats SLA de valors de pèrdua de paquets per a un percentatge del temps i retard.

El tipus de trànsit ORO serà menys prioritari que RT, però més prioritari que Best Effort.

4.3.1.3 Real Time (RT)

El servei Real Time de NEBA està orientat a VoIP. És un trànsit prioritari respecte a la qualitat ORO. Tindrà associats SLA de valors de pèrdua de paquets per un percentatge del temps i retard inferiors als de la qualitat ORO i un paràmetre addicional de variació de fluctuació.

4.3.2 Paràmetres de NEBA

A partir d’ ara ens centrarem en la xarxa NEBA de Movistar, que és la que hi ha al nostre país.

Movistar estableix una sèrie de paràmetres per a cada tipus de servei, que són els següents:

- **Pèrdua de paquets:** Es defineix la pèrdua de paquets com el rati expressat en percentatge sobre un període de temps definit, del nombre de trames no entregades respecte del nombre de trames que haurien d' haver sigut entregades. Els valors associats a NEBA respecte la pèrdua de paquets són:

	Valor máximo de pérdida de tramas GPON
QoS BE	0,6%
QoS ORO	0,02%
QoS RT	0,01%

- **Retard:** Es defineix el retard com el temps necessari per transmetre una trama des de la interfície usuari-xarxa d' entrada fins a la interfície usuari-xarxa de sortida. Els valors de la xarxa NEBA respecte el retard són:

	Retardo medio unidireccional GPON
QoS BE	-
QoS ORO	50 ms
QoS RT	30 ms

- **Fluctuació de retard (Jitter):** La fluctuació de retard es defineix com la diferència entre el percentil 95 i el valor mig del retard unidireccional de les trames entregades de forma satisfactòria. Els valors de la xarxa NEBA respecte la fluctuació de retard són els següents:

	Variación de retardo (percentil 95%) GPON
QoS BE	-
QoS ORO	40 ms
QoS RT	8 ms

- **Taxa de transferència efectiva (Throughput):** Els perfils de taxa de taxa de transferència efectiva que ofereix NEBA de Movistar són els següents:

Modalidad	BE		ORO		RT	
	DOWN	UP	DOWN	UP	DOWN	UP
f1	300M	300M	50M	50M	2M	2M
f2	600M	600M	50M	50M	2M	2M

5. Mininet

Mininet és un programa que ens permet emular xarxes. Executa una sèrie de hosts, commutadors, enrutadors i links, i ens permet tenir una virtualització de una xarxa completa en un sol sistema.

Un host Mininet s' executa com una màquina real, podem connectar-nos-hi per ssh, i córrer programari típic d' una màquina Linux.

5.1 Per què utilitzar Mininet?

- **És ràpid:** Engegar una xarxa simple és qüestió de segons.
- **Podem crear topologies personalitzades:** podem crear xarxes que tingui des de dos commutadors fins a topologies que simulin la xarxa d' una universitat.
- **Podem córrer programes reals:** qualsevol programa que funcioni en Linux està disponible que fer-se servir dins d' un host Mininet.
- **Podem personalitzar el reenviament de paquets:** els commutadors de Mininet són personalitzables usant el protocol OpenFlow.
- **Podem córrer Mininet en un portàtil, en un servidor o en una màquina virtual.** Cal remarcar que Mininet està inclòs a partir de Ubuntu 12.10.
- **Podem compartir o replicar resultats** en diferents màquines: qualsevol que tingui un ordinador pot córrer el teu codi exactament igual que ho has fet tu.
- **És fàcil de fer servir**. Podem crear i córrer experiments en Mininet escrivint simples (o complexos) scripts en Python.
- **Mininet és un programa de codi obert**, així que podem mirar el seu codi, modificar-lo, trobar i solucionar bugs, etc. També podem modificar la seva documentació per eliminar errors i afegir informació addicional.
- **Mininet és un programa en desenvolupament actiu.** Això vol dir que si es troben bugs o alguna cosa no funciona com hauria de funcionar, podem contactar amb els creadors de Mininet perquè ho solucionin.

5.2 Limitacions de Mininet

Mininet és un programa molt útil, però també té una sèrie de limitacions que cal tenir en compte a l' hora de simular xarxes. Aquestes són algunes de les seves limitacions:

- **Els recursos de la màquina on correm Mininet seran repartits** entre els hosts i els commutadors simulats, això vol dir que cal utilitzar una màquina amb bons recursos de CPU i RAM.
- **Mininet fa servir un únic kernel de Linux** per tots els hosts virtuals. Això vol dir que no podem córrer programari que depengui de BSD, Windows o MAC.
- **Mininet no escriurà el controlador OpenFlow per tu.** Això vol dir que si necessitem enrutament personalitzat o commutació personalitzada, caldrà que trobem o escrivim nosaltres mateixos el controlador SDN.
- Per defecte, **la nostra xarxa de Mininet està isolada** de la nostra LAN o de Internet. Això normalment és bo, però pot ser que en algun moment vulguem

que això no passi. Podem fer servir NAT per connectar la xarxa Mininet a la LAN del nostre ordinador.

- Per defecte tots **els hosts Mininet comparteixen el sistema de fitxers** del host, així com l' espai del PID. Això significa que hem d' anar en compte si estem corrent daemons que requereixin configuració en /etc, i hem d' anar en compte de no matar els processos incorrectes per error.

5.3 Instal·lació de Mininet

Hi ha tres maneres d' instal·lar Mininet al nostre ordinador. La primera manera és instal·lar una màquina virtual amb Mininet instal·lat. Aquest mètode és útil per usuaris amb ordinadors de gran capacitat, però no és el nostre cas. Cal especificar que estic fent servir un ordinador portàtil Lenovo Z50-70 amb Ubuntu 20.04 instal·lat, i si hem d' utilitzar només una part dels recursos de l' ordinador, pot ser que la màquina virtual ens vagi una mica lenta.

La segona opció és descarregar el paquet de Mininet i instal·lar-lo mitjançant les comandes que veurem a continuació, i la tercera opció és instal·lar-lo fent servir un gestor de paquets. Vaig provar la tercera opció i no va acabar de funcionar, així que jo recomano fer servir la segona opció, que és la que em va funcionar perfectament.

Per començar, cal que ens descarreguem el codi font, amb la següent comanda:

- git clone git://github.com/mininet/mininet

Tot seguit, comprovarem que estem instal·lant la versió indicada, la 2.3.0. Ho farem amb les següents comandes:

- cd mininet
- git tag
- git checkout -b mininet-2.3.0 2.3.0
- cd ..

Un cop tenim el codi font descarregat, l' instal·lem amb la següent comanda:

- mininet/util/install.sh

Un cop tenim Mininet instal·lat, cal que instal·lem els paquets de OpenVS Switch. Ho fem amb les comandes:

- sudo apt-get install openvswitch-switch
- sudo apt-get install openvswitch-common

Per últim, comprovem que tenim la instal·lació feta correctament provant Mininet amb la següent comanda:

- sudo mn –switch ovsbr –test pingall

5.4 Selecció d' eines

A continuació faré una breu descripció de les eines triades per la realització de la part pràctica d' aquest treball de final de grau. Òbviament, hi ha alternatives que poden ser igual de vàlides a les eines triades.

5.4.1 Python

He triat Python com a eina de desenvolupament per aquest treball. L' he triat perquè Python té una sèrie d' avantatges que són molt útils:

- **És un llenguatge d' alt nivell**, el qual fa la vida més fàcil als programadors.
- **És fàcil de llegir i d' aprendre**, el qual el fa fàcil de llegit per persones que no estan familiaritzades amb aquest llenguatge.
- **Integra el protocol OpenFlow**. Mininet integra Python en el seu funcionament, i podem fer-lo servir per programar els controladors SDN.

5.4.3 Wireshark

Wireshark és una eina de captura de paquets que farem servir per analitzar el trànsit de la xarxa que simularem. He triat Wireshark perquè és un software que compta amb una interfície d' usuari molt simple i fàcil d' utilitzar. A més, ja l' he fet servir una mica en altres assignatures per tant m' és força familiar.

5.4.4 Iperf

IPERF és una eina que s' utilitza per fer proves en xarxes informàtiques. Ens permet crear fluxos de dades TCP i UDP i mesurar el rendiment de la xarxa.

IPERF ens permetrà inundar la xarxa de paquets per mesurar la qualitat de servei que ens dona la xarxa SDN.

5.4.5 TCPDump

TCPDump és una eina de línia de comandes que té la funció principal d' analitzar el trànsit que circula per una xarxa. Farem servir aquesta eina en col·laboració amb Wireshark per analitzar el trànsit de xarxa.

Per utilitzar-lo, només cal descarregar el paquet i cridar-lo per línia de comandes.

5.5 Topologies

Una topologia es defineix com un mapa físic o lògic d' una xarxa per intercanviar dades. És la forma en la qual està dissenyada la xarxa, ja sigui en el pla físic o lògic.

Amb Mininet podem recrear tot tipus de topologies. Podem fer servir topologies predefinides al mateix Mininet o podem crear una topologia personalitzada utilitzant Python.

5.5.1 Topologies Bàsiques

La topologia més bàsica que podem invocar amb Mininet és la topologia MINIMAL, que consta de dos hosts, un commutador OpenFlow i un controlador bàsic. Per invocar-la, només cal que fem:

- sudo mn

També podem invocar la topologia SINGLE, que consta d' un sol commutador, i X hosts connectats a ell. Per invocar-la ho fem amb la comanda:

- sudo mn –topo single,X

on X és el nombre de hosts que volem que estiguin connectats al commutador.

Per últim, veurem la topologia LINEAR, que consisteix en X commutadors connectats entre ells, cadascun amb un host connectat.

Per invocar-la, farem:

- sudo mn –topo linear, X

on X és el nombre de commutadors que tenim a la topologia.

5.5.2 Topologies Personalitzades

La part interessant de Mininet i les seves topologies, és que podem crear-ne de personalitzades fent servir la API de Python. Podem fer ús d' aquest llenguatge per programar topologies personalitzades i després executar-les amb Mininet utilitzant la comanda:

- sudo mn –custom /ubicació-de-la-topo –topo mytopo

Així doncs, ara veurem un exemple de topologia personalitzada escrita en Python:

```python
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class SingleSwitchTopo(Topo):
    "Single switch connected to n hosts."
    def build(self, n=2):
        switch = self.addSwitch('s1')
        # Python's range(N) generates 0..N-1
        for h in range(n):
            host = self.addHost('h%s' % (h + 1))
            self.addLink(host, switch)

def simpleTest():
    "Create and test a simple network"
    topo = SingleSwitchTopo(n=4)
    net = Mininet(topo)
    net.start()
    print( "Dumping host connections" )
    dumpNodeConnections(net.hosts)
    print( "Testing network connectivity" )
    net.pingAll()
    net.stop()

if __name__ == '__main__':
    # Tell mininet to print useful information
    setLogLevel('info')
    simpleTest()
```

Aquestes són algunes de les classes, mètodes, funcions i variables que inclou aquest codi:

- **Topo:** la classe base per topologies en Mininet.
- **Build ():** és el mètode que volem sobreescriure en la nostra classe. Aquest mètode crea una plantilla que és usada posteriorment per Mininet per crear la topologia.
- **AddSwitch ():** afegeix un commutador a la topologia i en retorna el nom.
- **AddHost ():** afegeix un host a la topologia i en retorna el nom.
- **AddLink ():** afegeix una connexió bidireccional a la topologia.
- **Mininet:** la classe principal per crear i manejar una xarxa.
- **Start ():** inicia la xarxa.

5.6 Tipus de controladors

Hi ha varis tipus de controladors SDN que podríem fer servir en aquest projecte. Existeixen el controlador NOX, el POX, el Beacon, el Floodlight, el Ryu, etc. Tots ells són bons controladors per fer servir amb Mininet.

A continuació veurem una sèrie de característiques que ha de tenir un controlador per ser-nos útil:

- **Suport OpenFlow**: a l' hora d' escollir un controladors necessitem conèixer les característiques de les versions d' OpenFlow que el controlador suporta. Una raó pel qual això és necessari és que algunes funcions importants com el suport IPv6 no és part de OpenFlow 1.0, i s' inclou a la versió 1.2.
- **Virtualització de xarxa**: un controlador SDN ha de suportar virtualització de xarxa degut als beneficis que això comporta. Aquesta característica permet al programador crear dinàmicament la xarxa virtual basada en polítiques, per satisfer una amplia gamma de requisits, sense afectar als fluxos existents.
- **Funcionalitats de xarxa**: per aconseguir major flexibilitat en termes de com els fluxos són enrutats, és important que el controlador pugui prendre decisions d' enrutament basant-se en múltiples camps de la capçalera d' OpenFlow.
- **Escalabilitat:** una consideració fonamental amb respecte a l' escalabilitat d' una xarxa SDN és el nombre de commutadors que un controlador SDN pot suportar.
- **Rendiment**: una de les principals funcions d' un controlador SDN és establir fluxos. Per això, dos dels indicadors clau de rendiment associats amb un controlador SDN són el temps de formació d' un flux i el nombre de fluxos per segon que pot establir el controlador.
- **Programació de xarxa**: una de les característiques fonamentals de les SDN és la existència d' interfícies per la programació dels controladors, el qual possibilita que aquests ofereixin varies funcionalitats. Exemples de programació que s' han de buscar en un controlador SDN són la capacitat de redirigir el trànsit, i la possibilitat d' aplicar filtres sofisticats als paquets.
- **Confiabilitat**: una de les tècniques que un controlador SDN pot utilitzar per augmentar la fiabilitat de la xarxa és la capacitat de descobrir múltiples camins des de l' origen fins el destí, el qual es pot realitzar si contínuament es controla la topologia de xarxa.
- **Monitorització centralitzada i virtualització**: un controlador SDN ha de ser capaç de utilitzar les dades ofertes per protocol OpenFlow per identificar els problemes de la xarxa i automàticament, canviar la ruta que pren un flux determinat. El controlador també ha de permetre veure els fluxos, tant de la perspectiva de la xarxa física com la virtual, i obtenir informació detallada sobre aquests.
- **Processament**: Al avaluar un controlador hem de saber si suporta processos múltiples o no, doncs això pot repercutir en la escalabilitat dels nuclis de la

CPU. No tindria sentit que un controlador mono-procés s' executi sobre un hardware amb múltiples CPUs.

5.7 Controlador POX

POX és un programari de xarxes escrit en Python.

EL programari POX va començar com un controlador OpenFlow, però actualment també pot funcionar com un commutador OpenFlow, i pot ser útil per escriure software de xarxes en general.

POX pot córrer sobre Linux, MAC o Windows, sempre i quant es tingui Python 3 instal·lat.

Hem triat el controlador POX perquè compleix tots els criteris que busquem en un controlador. Té suport OpenFlow, permet virtualització de xarxa, és força escalable, a més està escrit en Python, que és el llenguatge de programació que hem triat per aquest projecte. També ens permet programar-lo perquè s' adapti a les nostres necessitats, és confiable, ens permet fer servir programes externs com Wireshark...

Com a punts negatius tenim que és un dels controladors més nous que hi ha, i a més està en fase de desenvolupament. Això vol dir que, per exemple, la documentació que hi ha sobre POX potser és inferior a la que hi pot haver en altres controladors.

5.7.1 Instal·lació de POX

Instal·lar POX és bastant senzill. Només hem de descarregar la carpeta que el conté, i ja podem fer-lo servir.

Per fer-ho, primer entrarem a la carpeta on volem instal·lar POX i després executarem la comanda següent:

- git clone http://github.com/noxrepo/pox

Ara podem fer:

- cd pox

per accedir als components de POX.

Ara ja tenim POX instal·lat, però encara no el podem començar a utilitzar. Hi ha un bug conegut, i hem de resoldre'l abans de poder utilitzar POX.

Per solucionar el bug, hem d' entrar a la carpeta de POX i buscar l' arxiu "dns.py". L' obrim amb un editor de text. El primer que hem de fer és eliminar totes les crides a la funció "ord", ja que és una funció que ja no serveix.

Per últim, cal afegir la funció "b()" a la línia 390, tal com s' indica a la pàgina web:

https://stackoverflow.com/questions/64312971/typeerror-ord-expected-string-of-length-1-but-int-found-error-in-pox-control

5.7.2 Components de POX

Un component en POX és un script que podem invocar des de línia de comandes al iniciar POX, i que afegeix funcionalitats extres al controlador POX.

5.7.2.1 Py

Aquest component inicia un intèrpret Python que pot ser útil per debugar i per experimentació interactiva. Durant la fase de beta, aquest era el component per defecte quan iniciàvem el controlador POX. Els altre components poden afegir funcions o valors a aquest intèrpret.

5.7.2.2 Forwarding.hub

El que fa aquest component és transformar cada commutador en un hub de 10 ports.

5.7.2.3 Forwarding.l2_learning

Aquest component fa que els commutadors OpenFlow actuïn com a commutadors de nivell 2.

5.7.2.4 Forwarding.l2_pairs

Com el component anterior, aquest fa que els commutadors OpenFlow actuïn com a commutadors nivell 2, però amb la diferència que aquest component ho fa de la manera més simple possible. Ho fa basant-se purament en les adreces MAC.

5.7.2.5 Forwarding.l3_learning

Aquest component no és un enrutador, però definitivament tampoc és un commutador de nivell 2, sinó que és una barreja dels dos.

6. Simulació de xarxa amb Mininet

6.1 Telecat

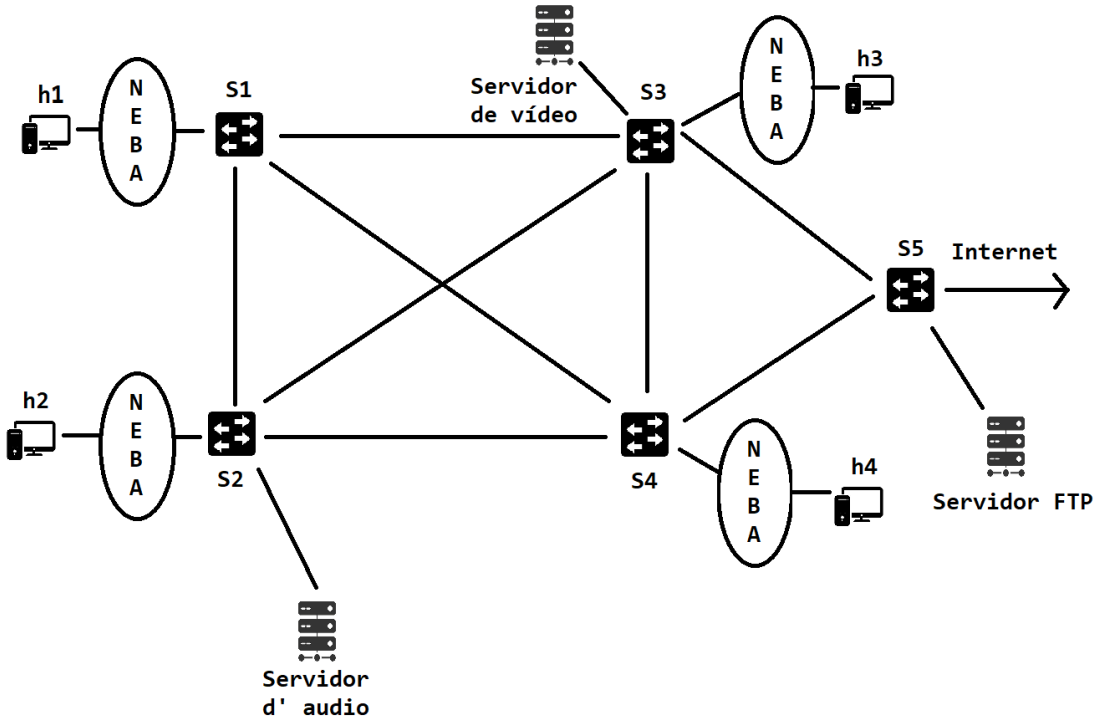
Per fer la simulació de la xarxa OpenFlow, imaginarem que som un operador d' Internet que vol donar servei a tot Catalunya, utilitzant SDN.

El nostre operador de telecomunicacions es dirà Telecat, i donarà servei d' Internet amb fibra òptica a les quatre províncies que formen Catalunya.

Per això, cal crear el core network de la nostra xarxa. Ens serà suficient amb un commutador OpenFlow a cada província. Els quatre enrutadors estaran connectats entre si, i també estaran connectats a Internet, per descomptat.

6.2 La nostra topologia

Un cop decidit l' abast de la nostra xarxa, cal fer la topologia lògica de la xarxa.



Podem veure que tenim quatre commutadors que formen el nostre core network, interconnectats entre ells, de manera que tenen redundància. Si un enllaç falla, tenim dos enllaços extres a cada commutador que poden suplir l' enllaç caigut. Tenim un cinquè commutador que ens servirà com a enllaç a Internet.

També tenim 4 hosts a mode d' exemple, que estan connectats mitjançant la xarxa NEBA.

També tenim 3 servidors, que ens serviran per determinar els paràmetres de connexió amb els hosts. Tenim un servidor de vídeo, un d' àudio i un servidor FTP.

Tot seguit, passarem el core network a Mininet. Per fer-ho, programarem la xarxa en Python.

```python
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def build( self ):
        "Create custom topo."

        # Add hosts
        firstHost = self.addHost ( 'h1' )
        secondHost = self.addHost ( 'h2' )
        thirdHost = self.addHost ( 'h3')
        fourthHost = self.addHost ( 'h4' )

        #Add servers
        audioServer = self.addHost ('h5' )
        videoServer = self.addHost ( 'h6' )
        ftpServer = self.addHost ( 'h7' )

        #Add switchs
        firstSwitch = self.addSwitch ( 's1' )
        secondSwitch = self.addSwitch ( 's2' )
        thirdSwitch = self.addSwitch ( 's3' )
        fourthSwitch = self.addSwitch ( 's4' )
        fifthSwitch = self.addSwitch ( 's5' )

        # Add links to hosts
        self.addLink (firstHost , firstSwitch)
        self.addLink (secondHost , secondSwitch)
        self.addLink (thirdHost , thirdSwitch)
        self.addLink (fourthHost , fourthSwitch)

        #Add links to servers
        self.addLink (audioServer , secondSwitch)
        self.addLink (videoServer , thirdSwitch)
        self.addLink (ftpServer , fifthSwitch)
```

```python
        #Add links to switchs
        self.addLink (firstSwitch , thirdSwitch)
        self.addLink (firstSwitch , fourthSwitch)
        self.addLink (firstSwitch , secondSwitch)
        self.addLink (secondSwitch , thirdSwitch)
        self.addLink (secondSwitch , fourthSwitch)
        self.addLink (thirdSwitch , fourthSwitch)
        self.addLink (thirdSwitch , fifthSwitch)
        self.addLink (fourthSwitch , fifthSwitch)



topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Primer de tot afegim els quatre hosts. Després afegim els servidors i per últim afegim els cinc commutadors.

Després afegim els links entre hosts i commutadors. Tot seguit, afegim els links entre servidors i commutadors i finalment afegim els links entre commutadors.

Amb això, ja tenim la topologia llesta per fer-se servir amb Mininet.

6.3 Desenvolupament de la xarxa i proves

6.3.1 Les comandes

Per executar la simulació de la xarxa, necessitarem dues comandes a executar des de la línia de comandes d' Ubuntu.

La primera és per executar el controlador POX, i un cop s' estigui executant, emprarem la segona comanda per executar la xarxa i connectar-la al controlador POX. En aquesta topologia tots cinc commutadors estaran connectats al controlador POX.

La primera comanda és:

- sudo ~/pox/pox.py forwarding.l2_learning openflow.spanning_tree --no-flood --hold-down log.level --DEBUG samples.pretty_log openflow.discovery host_tracker info.packet_dump

Els components que es fan servir en aquesta comanda són:

- **Forwarding.l2_learning**: Com hem vist abans, aquest component fa que els commutadors OpenFlow actuïn com a commutadors de nivell dos. Aprèn adreces MAC, i relaciona tots els camps de la capçalera del paquet, així que pot establir diferents fluxos per cada parell d' adreces MAC.
- **Openflow.discovery:** Aquest component fa servir missatges LLDP enviats i rebuts dels commutadors OpenFlow per descobrir la topologia de xarxa. També detecta quan els links cauen o es tornen a activar. A més, la informació pot ser utilitzada per altres components.
- **OpenFlow.spanning_tree –no-flood –hold-down:** El component Spanning Tree es requereix en topologies que contenen bucles. Treballa conjuntament amb l' OpenFlow Discovery per contruir una topologia i construeix un spanning tree desactivant els links que formen bucles entre sí. Les opcions –no-flood i –hold-down s' usen per assegurar que cap commutador fa flooding abans de la creació del spanning tree.
- **Host_tracker:** Aquest component manté informació sobre els hosts de la xarxa. Examina els paquets rebuts per POX i aprèn les adreces MAC i IP dels hosts de la xarxa.
- **Info.packet-dump:** El paquet Packet Dump ens mostra per consola informació sobre paquets de dades rebuts per POX de part dels commutadors. Això ens ajudarà a veure com els commutadors interactuen amb el controlador POX sense fer servir TCP Dump.
- **Log.level –DEBUG:** El component Log Level permet al usuari de POX especificar el nivell de detall que vol veure al log d' informació produït per POX. El nivell amb més detall és DEBUG, i serà el que farem servir.

- **Samples.pretty_log:** El component Pretty Log formateja els missatges en un log personalitzat que el fa més llegible que el log per defecte.

La segona comanda és la següent. Aquesta servirà per executar Mininet i connectar-se al controlador POX:

> ➢ sudo mn –custom /home/pau/mininet/custom/5switches.py –topo mytopo –controller=remote,ip=127.0.0.1,port6633

Aquesta comanda comença amb l' ordre "mn", que serveix per engegar Mininet. Va seguit d' unes quantes opcions, que són:

- **--custom:** serveix per indicar que farem servir una topologia personalitzada, i va seguit de l' adreça de l' arxiu que conté aquesta topologia.
- **--topo:** serveix per indicar que farem servir una topologia, en el nostre cas fem servir "mytopo" perquè és una topologia personalitzada.
- **--controller**: serveix per indicar que farem servir un controlador SDN, i hi especifiquem l' adreça i el port. En el nostre cas, com que ho fem tot al mateix ordinador, passem l' adreça de loopback i el port 6633, que és el port d' OpenFlow.

6.3.1 Execució

6.3.1.1 Engegar la xarxa

A continuació, realitzarem l' execució de les comandes especificades a l' apartat anterior.

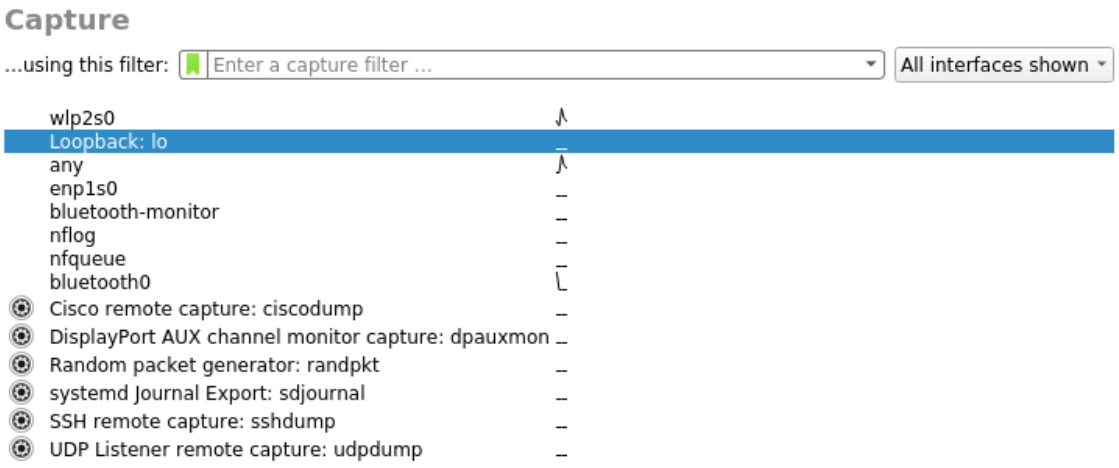
Començarem executant la primera comanda, per iniciar el controlador POX.

Aquest és el resultat:

```
pau@pau-Lenovo-Z50-70:~/mininet/POX/pox$ sudo ./pox.py forwarding.l2_learning op
enflow.spanning_tree --no-flood --hold-down log.level --DEBUG samples.pretty_log
 openflow.discovery host_tracker info.packet_dump
[sudo] password for pau:
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
[openflow.spanning_tree ] Spanning tree component ready
[host_tracker           ] host_tracker ready
[info.packet_dump       ] Packet dumper running
[core                   ] POX 0.7.0 (gar) going up...
[core                   ] Running on CPython (3.8.10/Sep 28 2021 16:10:42)
[core                   ] Platform is Linux-5.11.0-40-generic-x86_64-with-glibc2
.29
[version                ] Support for Python 3 is experimental.
[core                   ] POX 0.7.0 (gar) is up.
[openflow.of_01         ] Listening on 0.0.0.0:6633
```

A la imatge podem veure com s' inicien els diferents components, i per últim, podem veure que el controlador POX està engegat.

A continuació, iniciarem Wireshark, per poder veure els intercanvis de paquets un cop iniciem Mininet.



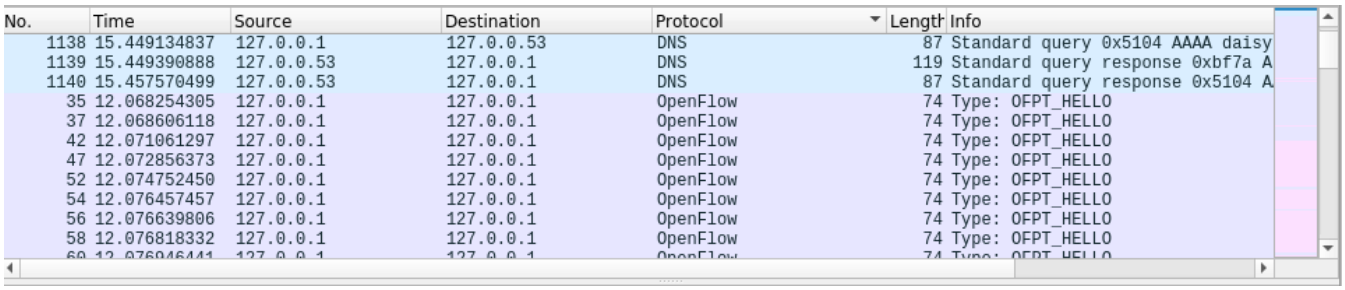
Seleccionarem l' interfície "lo", que vol dir loopback. Veiem que de moment no està rebent res. Un cop dins, escrivim la paraula clau "openflow", per veure els paquets OpenFlow.

Un cop està corrent Wireshark, executarem la segona comanda per engegar Mininet.

```
pau@pau-Lenovo-Z50-70:~$ sudo mn --custom ~/mininet/custom/5switches.py --topo m
ytopo --controller=remote,ip=127.0.0.1,port=6633
[sudo] password for pau:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (h5, s2) (h6, s3) (h7, s5) (s1, s2) (s1, s3)
 (s1, s4) (s2, s3) (s2, s4) (s3, s4) (s3, s5) (s4, s5)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7
*** Starting controller
c0
*** Starting 5 switches
s1 s2 s3 s4 s5 ...
*** Starting CLI:
mininet>
```

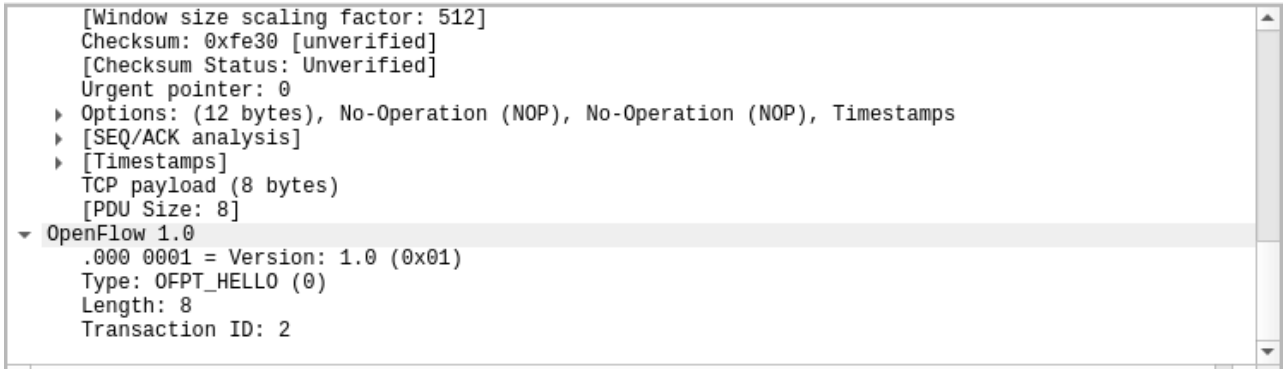
Quan engeguem Mininet ens surt una cosa semblant a aquesta. Podem veure tots els hosts que hi ha a la topologia, els commutadors, els links i veiem que s' inicia un controlador "c0" que és el nostre controlador POX.

Per l' altra banda, tenim que a Wireshark comencen a aparèixer paquets relacionats amb OpenFlow:



No.	Time	Source	Destination	Protocol	Length	Info
1138	15.449134837	127.0.0.1	127.0.0.53	DNS	87	Standard query 0x5104 AAAA daisy
1139	15.449390888	127.0.0.53	127.0.0.1	DNS	119	Standard query response 0xbf7a A
1140	15.457570499	127.0.0.53	127.0.0.1	DNS	87	Standard query response 0x5104 A
35	12.068254305	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_HELLO
37	12.068606118	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_HELLO
42	12.071061297	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_HELLO
47	12.072856373	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_HELLO
52	12.074752450	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_HELLO
54	12.076457457	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_HELLO
56	12.076639806	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_HELLO
58	12.076818332	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_HELLO

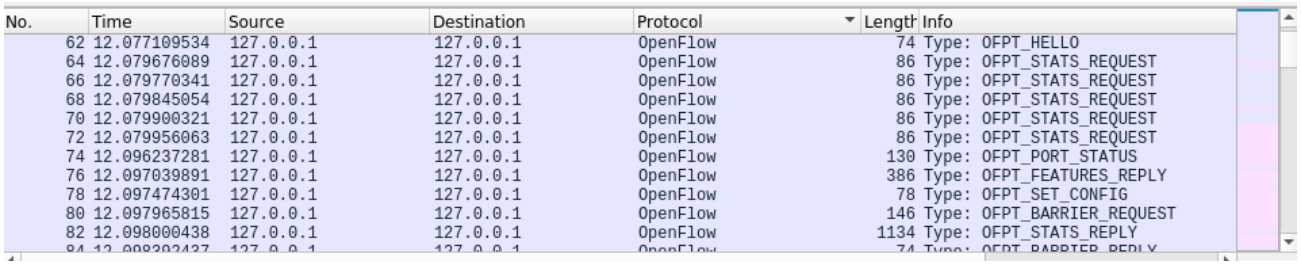
Veiem que apareixen molts paquets de "Hello" del protocol OpenFlow. Aquests paquets s' estableixen entre els commutadors i el controlador per mantenir activa la connexió.



```
    [Window size scaling factor: 512]
    Checksum: 0xfe30 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [SEQ/ACK analysis]
  ▸ [Timestamps]
    TCP payload (8 bytes)
    [PDU Size: 8]
▾ OpenFlow 1.0
    .000 0001 = Version: 1.0 (0x01)
    Type: OFPT_HELLO (0)
    Length: 8
    Transaction ID: 2
```

Aquest és un dels paquets "hello".

També podem veure que un cop establerta la connexió entre els commutadors i el controlador mitjançant els paquets “hello”, altres tipus de paquets comencen a arribar al controlador:



No.	Time	Source	Destination	Protocol	Length	Info
62	12.077109534	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_HELLO
64	12.079676089	127.0.0.1	127.0.0.1	OpenFlow	86	Type: OFPT_STATS_REQUEST
66	12.079770341	127.0.0.1	127.0.0.1	OpenFlow	86	Type: OFPT_STATS_REQUEST
68	12.079845054	127.0.0.1	127.0.0.1	OpenFlow	86	Type: OFPT_STATS_REQUEST
70	12.079900321	127.0.0.1	127.0.0.1	OpenFlow	86	Type: OFPT_STATS_REQUEST
72	12.079956063	127.0.0.1	127.0.0.1	OpenFlow	86	Type: OFPT_STATS_REQUEST
74	12.096237281	127.0.0.1	127.0.0.1	OpenFlow	130	Type: OFPT_PORT_STATUS
76	12.097039891	127.0.0.1	127.0.0.1	OpenFlow	386	Type: OFPT_FEATURES_REPLY
78	12.097474301	127.0.0.1	127.0.0.1	OpenFlow	78	Type: OFPT_SET_CONFIG
80	12.097965815	127.0.0.1	127.0.0.1	OpenFlow	146	Type: OFPT_BARRIER_REQUEST
82	12.098000438	127.0.0.1	127.0.0.1	OpenFlow	1134	Type: OFPT_STATS_REPLY

Fixem-nos en que com que estem fent una simulació amb un sol equip, tots els paquets OpenFlow van de l’ adreça 127.0.0.1 a l’ adreça 127.0.0.1.

A la terminal on hem obert el controlador POX també podem veure els logs del programa, on podem trobar paquets OpenFlow:

```
[openflow.spanning_tree ] Disabling flooding for 4 ports
[openflow.of_01         ] [00-00-00-00-00-02 6] connected
[openflow.discovery     ] Installing flow for 00-00-00-00-00-02
[forwarding.l2_learning ] Connection [00-00-00-00-00-02 6]
[openflow.spanning_tree ] Disabling flooding for 6 ports
[openflow.discovery     ] link detected: 00-00-00-00-00-03.5 -> 00-00-00-00-00-0
4.4
[openflow.discovery     ] link detected: 00-00-00-00-00-05.3 -> 00-00-00-00-00-0
4.5
[openflow.discovery     ] link detected: 00-00-00-00-00-03.3 -> 00-00-00-00-00-0
1.2
[openflow.discovery     ] link detected: 00-00-00-00-00-05.2 -> 00-00-00-00-00-0
3.6
[openflow.discovery     ] link detected: 00-00-00-00-00-02.4 -> 00-00-00-00-00-0
3.4
[openflow.discovery     ] link detected: 00-00-00-00-00-03.6 -> 00-00-00-00-00-0
5.2
[host_tracker           ] Learned 2 2 76:74:54:69:cb:3d
[dump:00-00-00-00-00-02 ] [ethernet][ipv6][icmpv6][NDNeighborSolicitation]
[openflow.discovery     ] link detected: 00-00-00-00-00-03.4 -> 00-00-00-00-00-0
2.4
[openflow.discovery     ] link detected: 00-00-00-00-00-02.5 -> 00-00-00-00-00-0
4.3
[openflow.discovery     ] link detected: 00-00-00-00-00-02.3 -> 00-00-00-00-00-0
```

Podem veure com al iniciar-se Mininet tots els components comencen a activar-se. Veiem que l’ spanning tree comença a funcionar, i desactiva el flooding per a tots els commutadors. També podem veure com es fa el discovery sobre diferents links, etc.

Al Mininet, podem executar la comanda “net” per veure tots els links establerts a la nostra topologia de xarxa:

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
h4 h4-eth0:s4-eth1
h5 h5-eth0:s2-eth2
h6 h6-eth0:s3-eth2
h7 h7-eth0:s5-eth1
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s3-eth3 s1-eth3:s4-eth2 s1-eth4:s2-eth3
s2 lo:  s2-eth1:h2-eth0 s2-eth2:h5-eth0 s2-eth3:s1-eth4 s2-eth4:s3-eth4 s2-eth5:
s4-eth3
s3 lo:  s3-eth1:h3-eth0 s3-eth2:h6-eth0 s3-eth3:s1-eth2 s3-eth4:s2-eth4 s3-eth5:
s4-eth4 s3-eth6:s5-eth2
s4 lo:  s4-eth1:h4-eth0 s4-eth2:s1-eth3 s4-eth3:s2-eth5 s4-eth4:s3-eth5 s4-eth5:
s5-eth3
s5 lo:  s5-eth1:h7-eth0 s5-eth2:s3-eth6 s5-eth3:s4-eth5
c0
mininet>
```

Veiem cada component de xarxa, i tot seguit, les interfícies que té connectades a altres components.

Per exemple, veiem que el host 1 té una interfície eth0, connectada a la interfície eth1 del commutador 1.

Ara provarem de fer un ping entre tots els components de la xarxa per comprovar que hi ha connectivitat. Farem servir la comanda “pingall”, que executa un ping des de cada host a tota la resta de hosts:

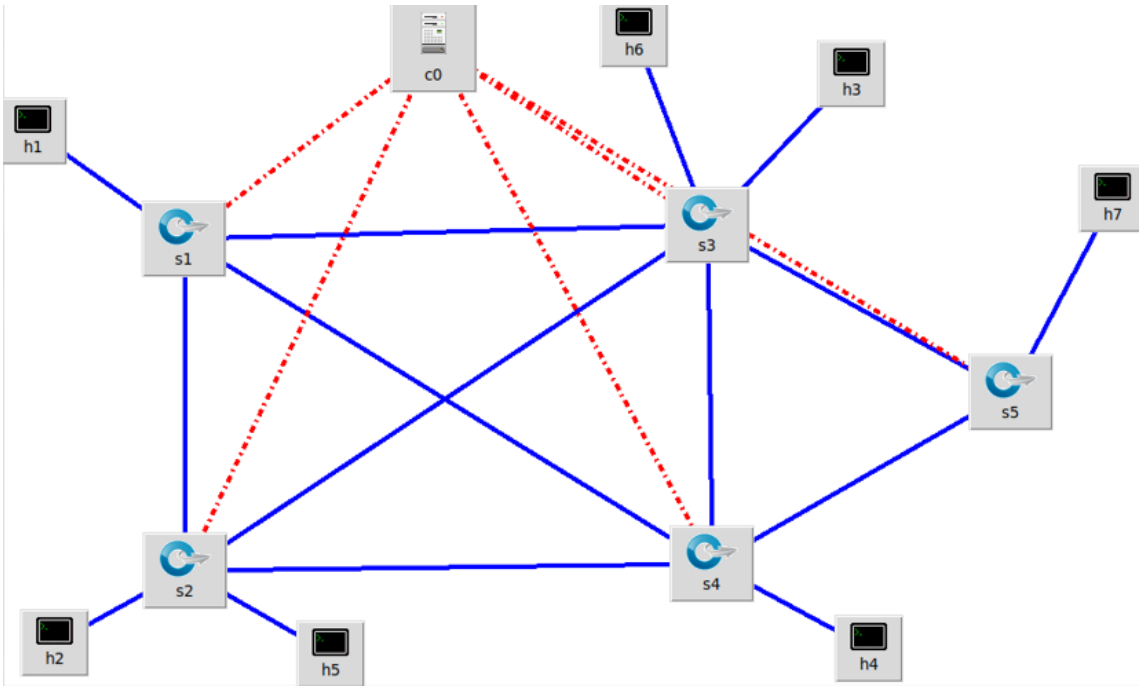
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7
h2 -> h1 h3 h4 h5 h6 h7
h3 -> h1 h2 h4 h5 h6 h7
h4 -> h1 h2 h3 h5 h6 h7
h5 -> h1 h2 h3 h4 h6 h7
h6 -> h1 h2 h3 h4 h5 h7
h7 -> h1 h2 h3 h4 h5 h6
*** Results: 0% dropped (42/42 received)
```

Podem comprovar que s’ envien 42 paquets i hi ha una taxa de caiguda del 0%, això vol dir que hi ha connectivitat entre tots els hosts de la xarxa.

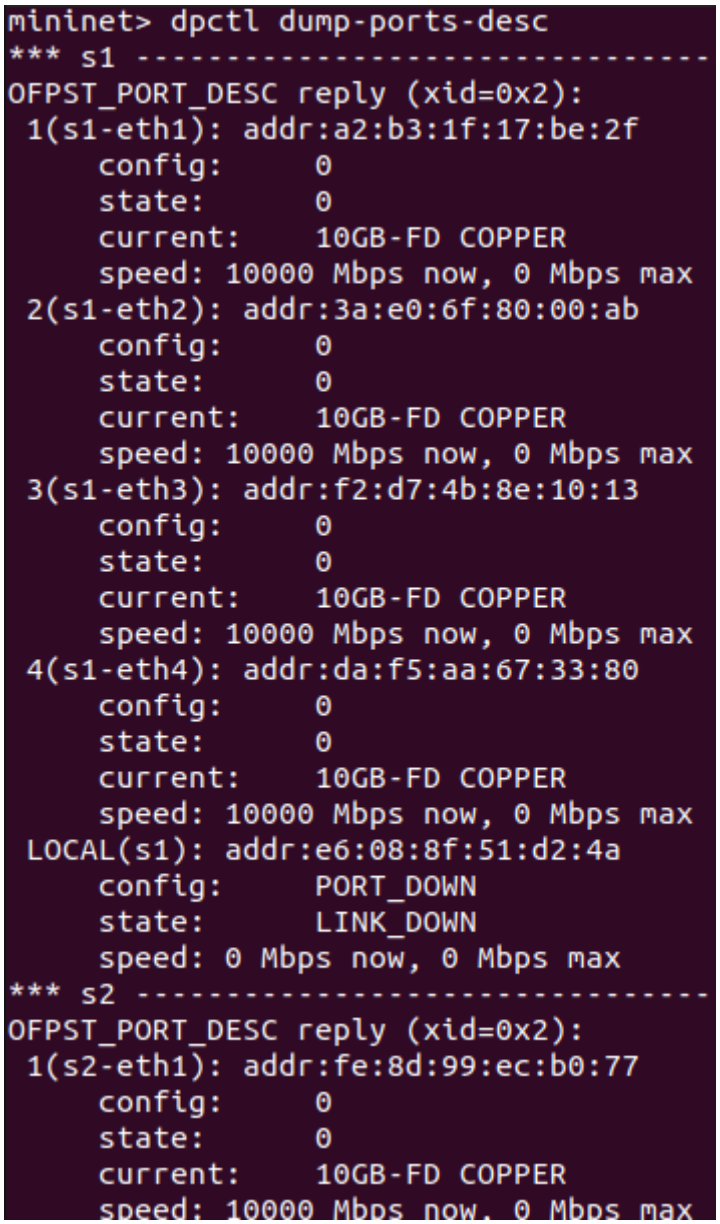
6.3.1.2 Funcionament de Spanning Tree

Per provar el funcionament de spanning tree farem servir Miniedit.

L’ eina Miniedit és una eina gràfica que ens permet crear topologies per al Mininet. La farem servir per dibuixar la nostra topologia, i també perquè ens permet desconnectar els links que vulguem fent clic dret sobre cada enllaç.

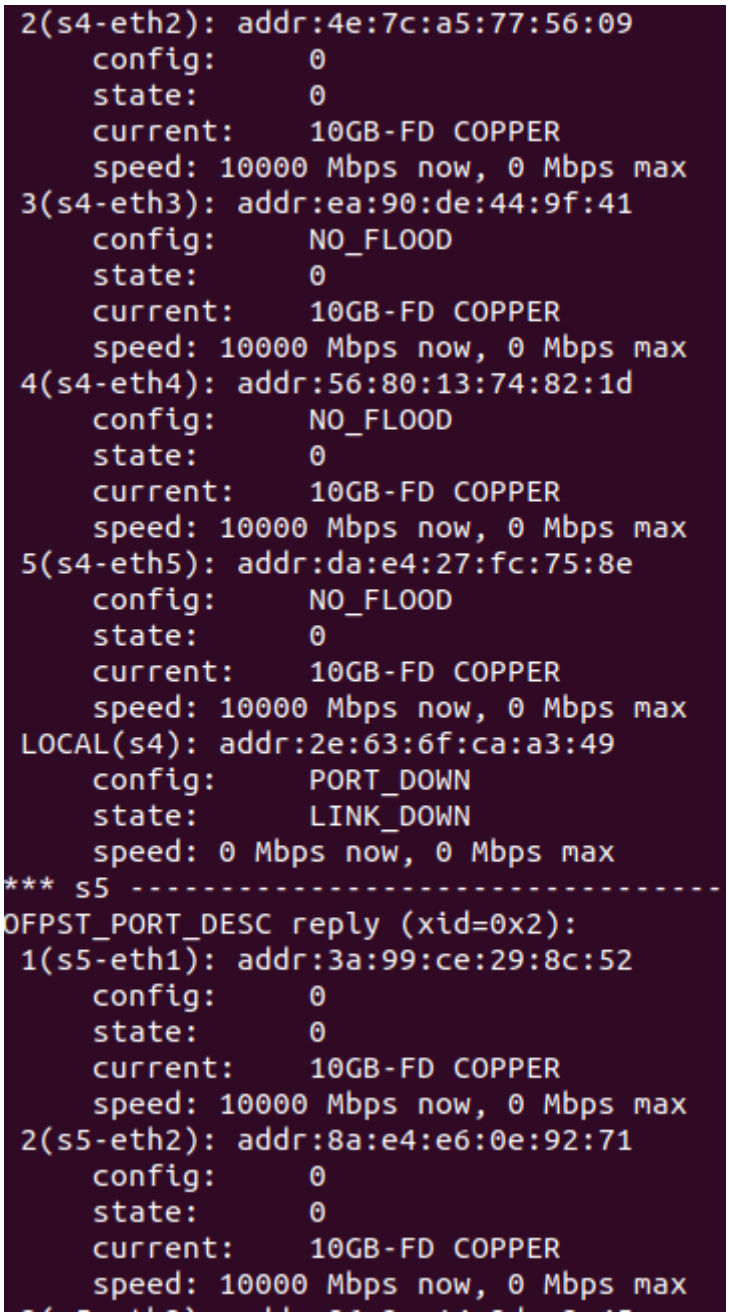


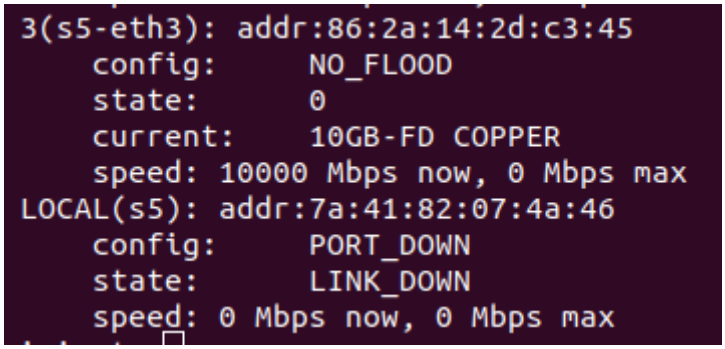
Tot seguit, provarem la comanda “dump ports description” (dpctl dump-ports-desc). Aquesta comanda ens permet veure informació detallada de cada port:


mininet> dpctl dump-ports-desc
*** s1 ----------
OFPST_PORT_DESC reply (xid=0x2):
1(s1-eth1): addr:a2:b3:1f:17:be:2f
config: 0
state: 0
current: 10GB-FD COPPER
speed: 10000 Mbps now, 0 Mbps max
2(s1-eth2): addr:3a:e0:6f:80:00:ab
config: 0
state: 0
current: 10GB-FD COPPER
speed: 10000 Mbps now, 0 Mbps max
3(s1-eth3): addr:f2:d7:4b:8e:10:13
config: 0
state: 0
current: 10GB-FD COPPER
speed: 10000 Mbps now, 0 Mbps max
4(s1-eth4): addr:da:f5:aa:67:33:80
config: 0
state: 0
current: 10GB-FD COPPER
speed: 10000 Mbps now, 0 Mbps max
LOCAL(s1): addr:e6:08:8f:51:d2:4a
config: PORT_DOWN
state: LINK_DOWN
speed: 0 Mbps now, 0 Mbps max
*** s2 ----------
OFPST_PORT_DESC reply (xid=0x2):
1(s2-eth1): addr:fe:8d:99:ec:b0:77
config: 0
state: 0
current: 10GB-FD COPPER
speed: 10000 Mbps now, 0 Mbps max

```
 2(s2-eth2): addr:32:33:fa:a9:08:31
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s2-eth3): addr:46:01:50:d4:af:28
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 4(s2-eth4): addr:ea:99:14:16:c4:3f
     config:     NO_FLOOD
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 5(s2-eth5): addr:9a:58:14:81:11:81
     config:     NO_FLOOD
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s2): addr:fa:4e:b0:95:f4:40
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
*** s3 --------------------------------
OFPST_PORT_DESC reply (xid=0x2):
 1(s3-eth1): addr:be:bf:ed:3b:0a:08
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s3-eth2): addr:92:ce:4b:50:57:bf
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
```

```
 3(s3-eth3): addr:8e:a1:14:ee:65:d6
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 4(s3-eth4): addr:36:78:99:54:2c:55
     config:     NO_FLOOD
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 5(s3-eth5): addr:22:6c:a4:c9:ab:5a
     config:     NO_FLOOD
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 6(s3-eth6): addr:06:ea:c1:35:a7:cc
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s3): addr:be:9a:90:7b:6e:4e
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
*** s4 ----------------------------------
OFPST_PORT_DESC reply (xid=0x2):
 1(s4-eth1): addr:16:cb:3f:0e:a1:b0
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
```

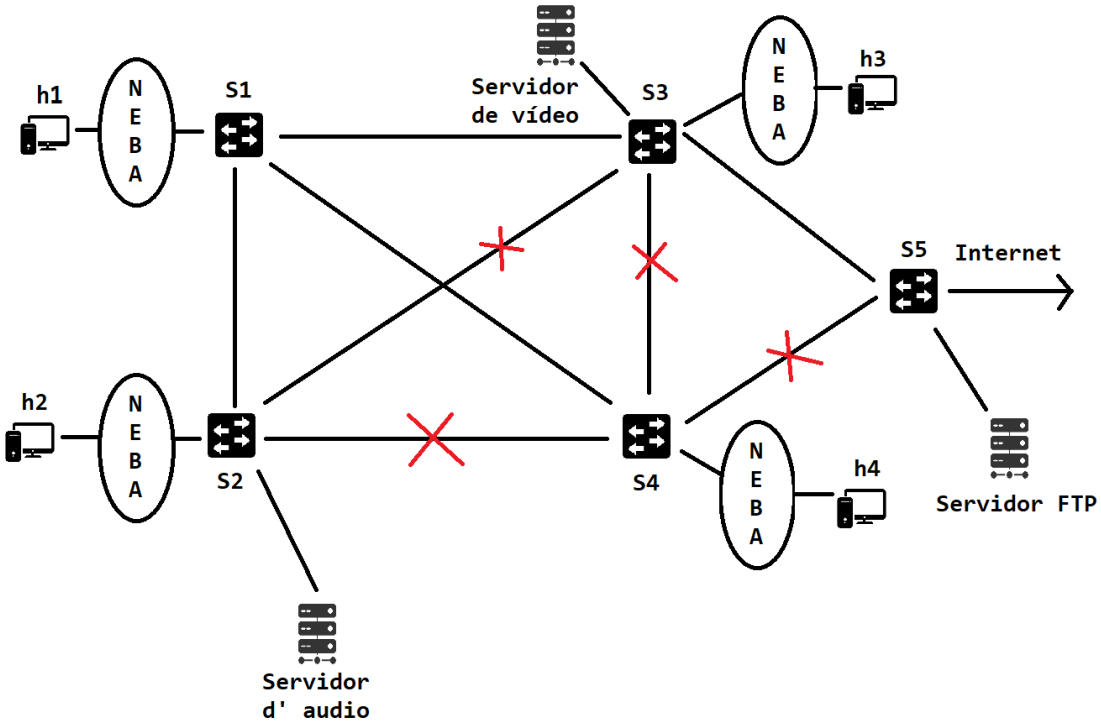




Podem veure que els ports tenen una velocitat per defecte de 10 Gbps. També ens permet veure el circuit que fa spanning tree per evitar bucles. Veiem que els ports que

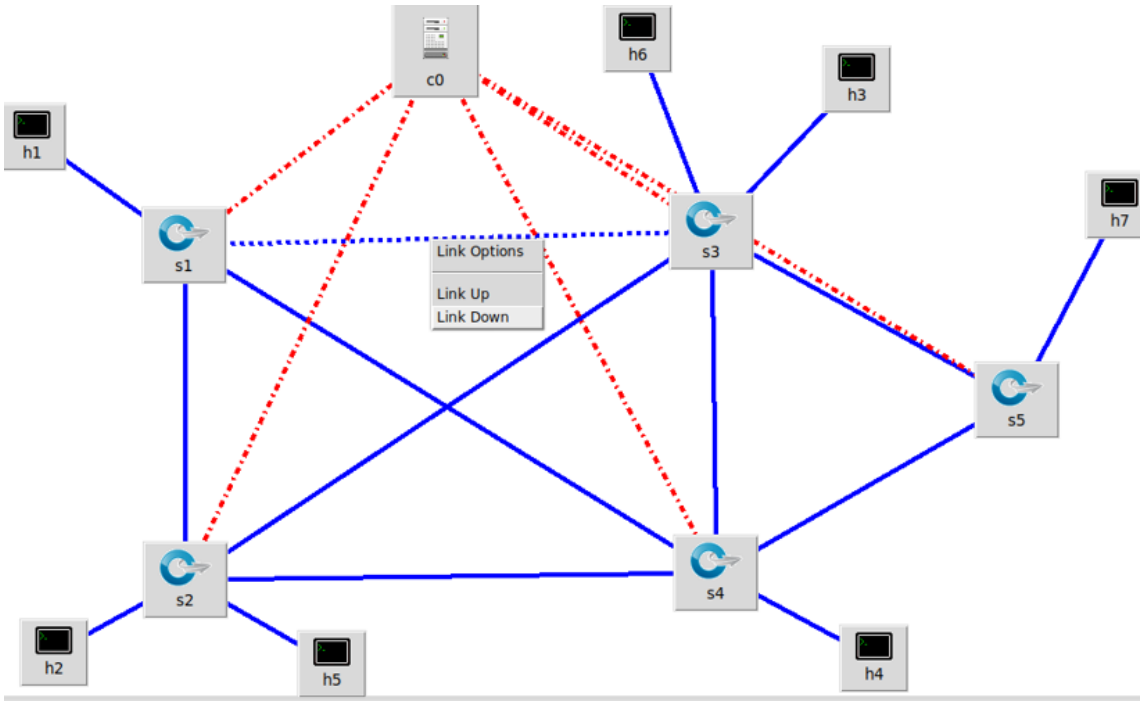
estan desactivats tenen la configuració NO_FLOOD, mentre que els ports actius tenen un estat de configuració 0, o correcte.

Així doncs, la configuració de la nostra topologia amb spanning tree queda de la següent manera:



A continuació provarem la funcionalitat de spanning tree de la nostra xarxa. Farem caure l' enllaç que hi ha entre el commutador 1 i el commutador 3 i veurem si spanning tree recalcula la ruta.

Farem clic dret sobre l' enllaç que hi ha entre el commutador 1 i el 3 i farem clic a "link down".



Veiem que l' enllaç ens apareix amb una línia discontinua. Això vol dir que ja està desconnectat.

Ara tornem a provar a fer un pingall per comprovar que hi ha connectivitat entre tots els hosts.

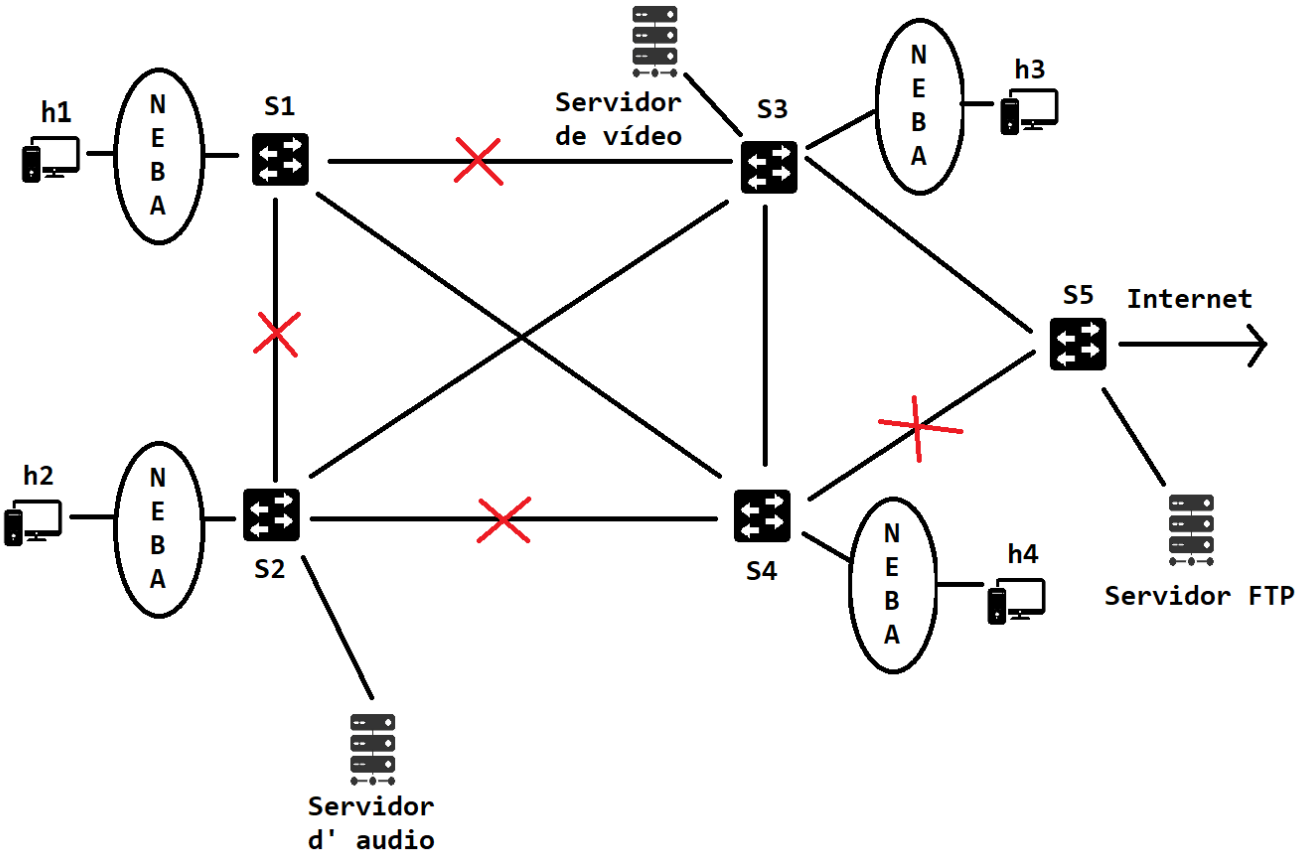
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7
h2 -> h1 h3 h4 h5 h6 h7
h3 -> h1 h2 X h5 h6 h7
h4 -> h1 h2 h3 h5 h6 h7
h5 -> h1 h2 h3 h4 h6 h7
h6 -> h1 h2 h3 h4 h5 h7
h7 -> h1 h2 h3 h4 h5 h6
*** Results: 2% dropped (41/42 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7
h2 -> h1 h3 h4 h5 h6 h7
h3 -> h1 h2 h4 h5 h6 h7
h4 -> h1 h2 h3 h5 h6 h7
h5 -> h1 h2 h3 h4 h6 h7
h6 -> h1 h2 h3 h4 h5 h7
h7 -> h1 h2 h3 h4 h5 h6
*** Results: 0% dropped (42/42 received)
```

Veiem que necessita uns segons per acabar de guanyar connectivitat amb tots els hosts, però al final acaba tenint connectivitat.

Tornem a executar la comanda "dump ports description" (dpctl dump-ports-desc):

```
1(s1-eth1): addr:06:a9:42:3b:0d:4c
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
2(s1-eth2): addr:ba:ae:8f:ac:f2:c6
     config:     PORT_DOWN NO_FLOOD
     state:      LINK_DOWN
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
3(s1-eth3): addr:4a:5d:88:c0:15:ce
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
4(s1-eth4): addr:b6:e9:33:d4:ac:4a
     config:     NO_FLOOD
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
```

Efectivament, veiem que ara l' enllaç eth2 que és el que connectava amb el commutador 3 apareix com a link_down. Fent el mateix procés que hem fet anteriorment, podem determinar com queda ara l' spanning tree de la topologia:



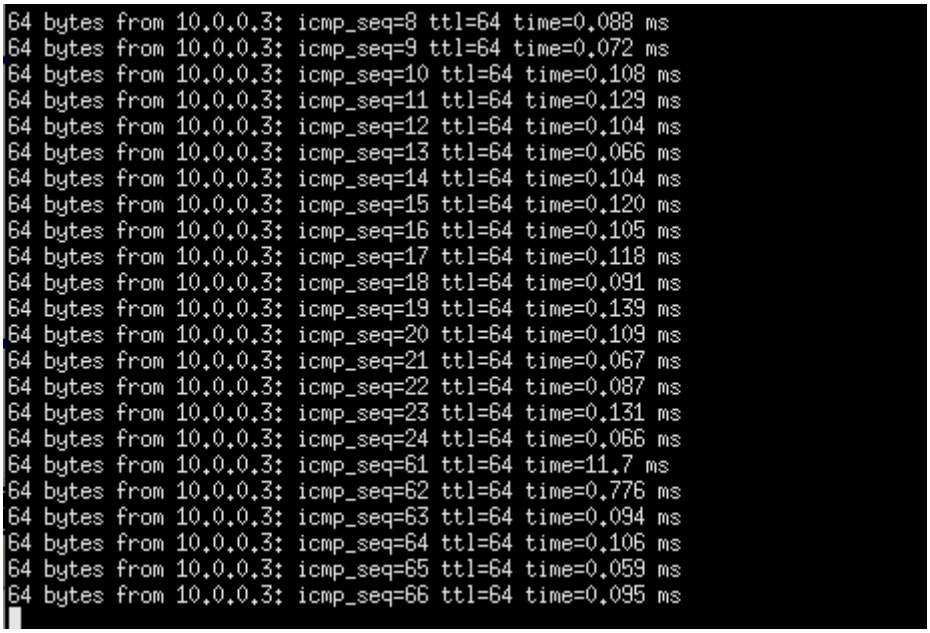
A continuació volem provar què passa quan estem enviant dades i cau un link.

Per fer-ho obrirem un terminal al host 1 amb la comanda

- xterm h1

des de Mininet.

Enviarem un ping al host 3 i tot seguit tot seguit seguirem el mateix procediment que hem fet abans. Farem caure el link entre el commutador 1 i el commutador 3 i veurem els resultats:



Veiem que l' spanning tree recupera la connectivitat uns 40 segons després de perdre-la. Podem veure que el ping envia fins a la seqüència 24 i tot seguit recupera el ping a la seqüència 61.

6.3.1.3 Xarxa NEBA

A continuació farem la comprovació de que la xarxa SDN és capaç de complir amb els paràmetres de la xarxa NEBA.

Fem servir la xarxa NEBA per connectar els hosts a la xarxa SDN.

La xarxa NEBA consta de tres tipus de connexions, i cada connexió té uns paràmetres que ha de complir.

A la nostra topologia hi ha tres servidors, cadascun d' ells ofereix un tipus de connexió de la xarxa NEBA. El servidor de vídeo ofereix servei "Real Time". El servidor d' àudio ofereix servei "Best Effort", mentre que el servidor FTP pertany a una empresa, i ofereix servei ORO.

Començarem fent la comprovació del throughput. Aquests són els paràmetres de cada tipus de connexió.

Modalidad	BE		ORO		RT	
	DOWN	UP	DOWN	UP	DOWN	UP
f1	300M	300M	50M	50M	2M	2M
f2	600M	600M	50M	50M	2M	2M

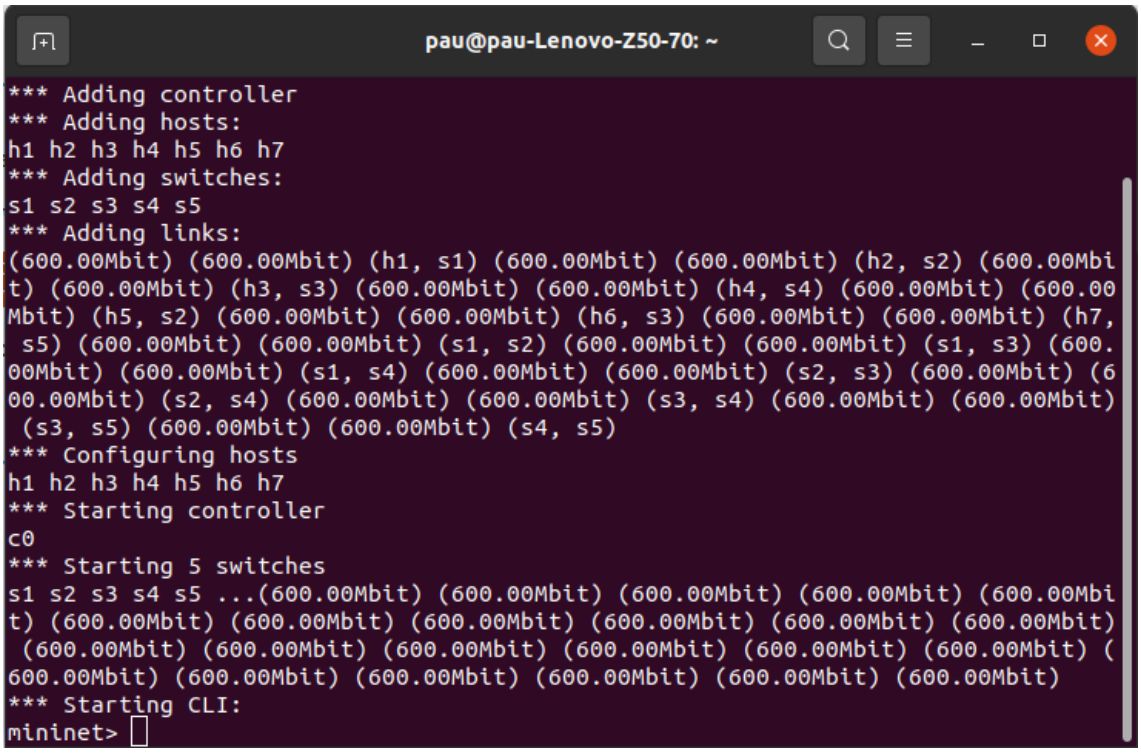
Agafarem la modalitat f2 perquè és la que té el throughput més gran, i això vol dir que si comprovem la modalitat f2, la modalitat f1 també queda comprovada.

Per fer-ho, iniciarem Mininet modificant la comanda. Especificarem que volem que tots els links tinguin 600Mbps de velocitat entre ells.

La comanda modificada és la següent:

- sudo mn --custom ~/mininet/custom/5switches.py --topo mytopo --controller=remote,ip=127.0.0.1,port=6633 --link tc,bw=600

Hem afegit el paràmetre "—link" que ens permet especificar la velocitat a la qual volem que treballin els components de xarxa.



Podem veure en aquesta imatge com la xarxa s' engega amb els links a 600 Mbps.

Per comprovar si de veritat va a 600 Mbps, utilitzarem IPERF per mesurar el throughput entre el host 1, que és un host normal, i el host 5, que és el servidor d' àudio.

Per fer-ho, obrirem un terminal a cada un dels hosts. Al host 5 engegarem l' IPERF en mode servidor, perquè estigui esperant connexions i al host 1 engegarem l' IPERF en mode client.

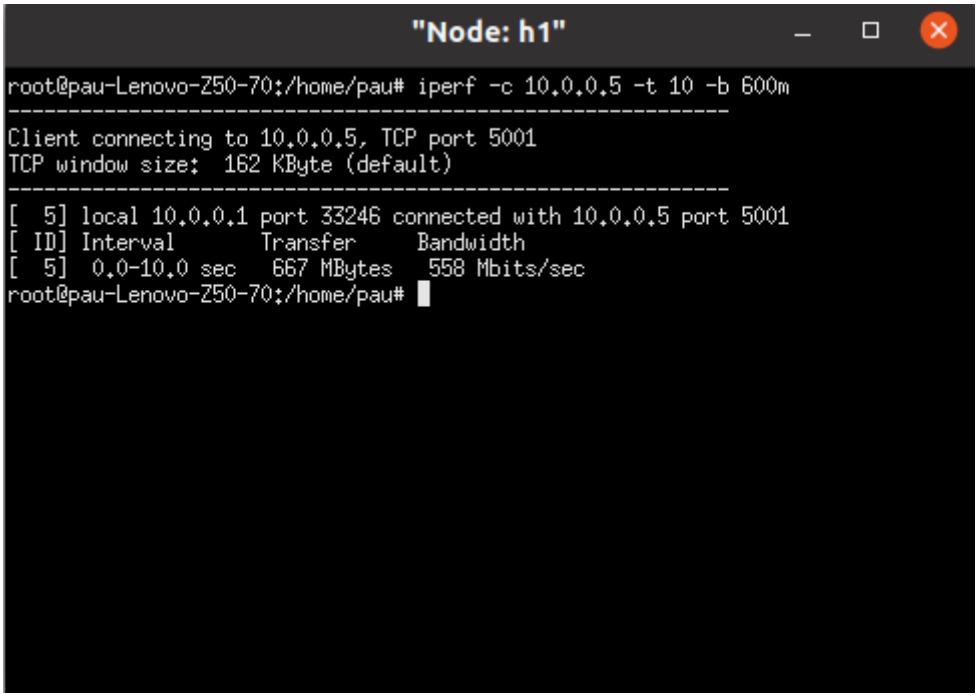
Al host 5:

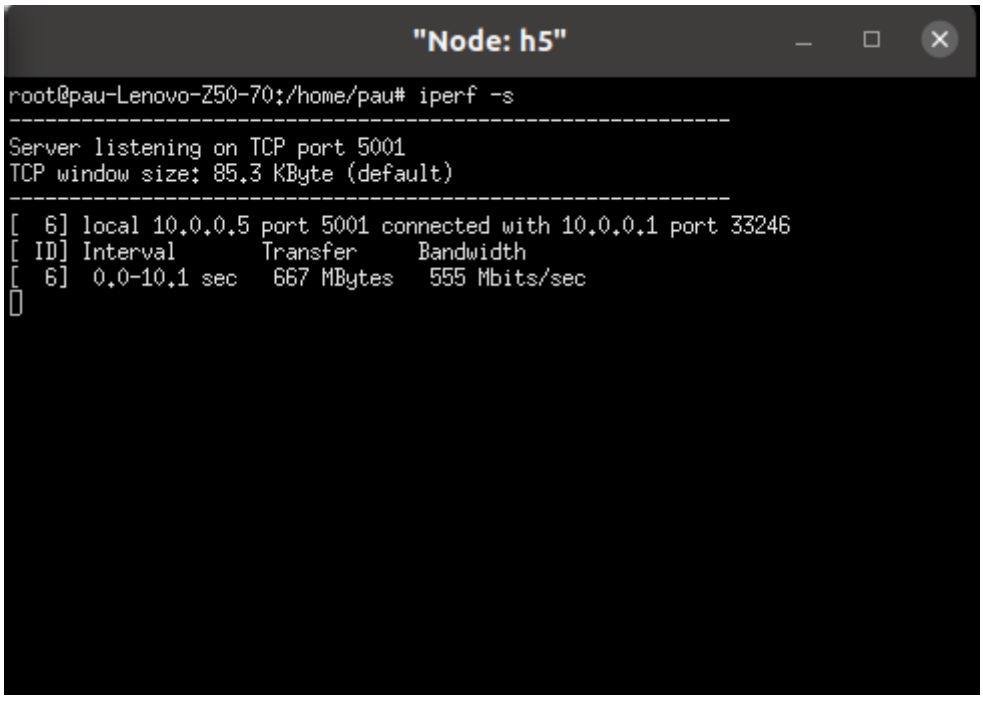
- iperf -s

Al host 1:

- iperf -c 10.0.0.5 -t 10 -b 600m

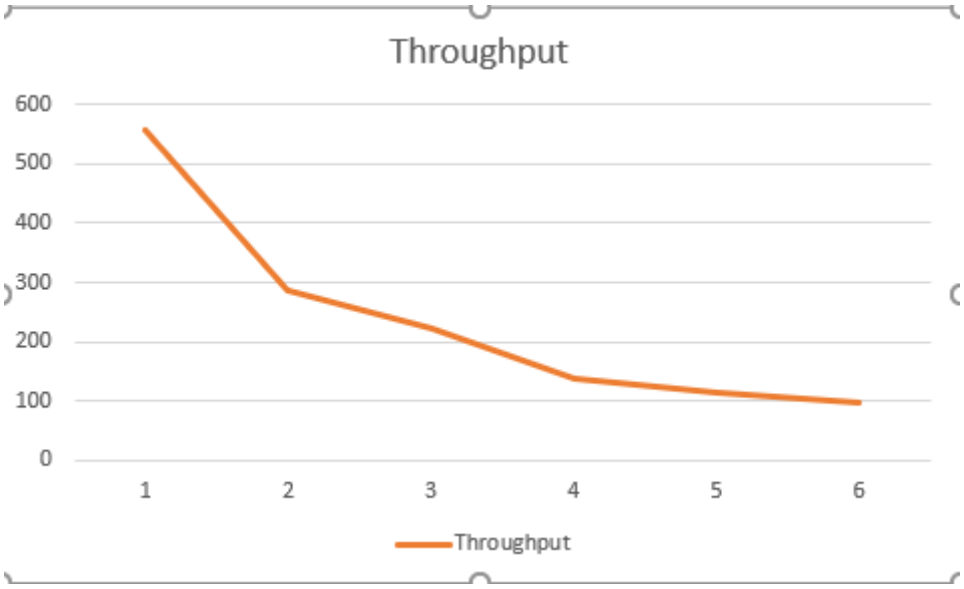
Especifiquem la IP del servidor, que és la 10.0.0.5, i especifiquem el temps durant el qual volem comprovar la connexió, en el nostre cas 10 segons, i la velocitat que volem comprovar, en el nostre cas 600 Mbps.





Podem veure que la connexió s' ha realitzat amb èxit, i hi ha hagut un throughput de 558 Mbps durant 10 segons. No ha arribat a 600 Mbps, però es queda molt a prop.

Tot seguit, provarem de fer varies connexions amb IPERF a la vegada, i elaborarem una taula amb les mitjanes de les velocitats que aconseguim:

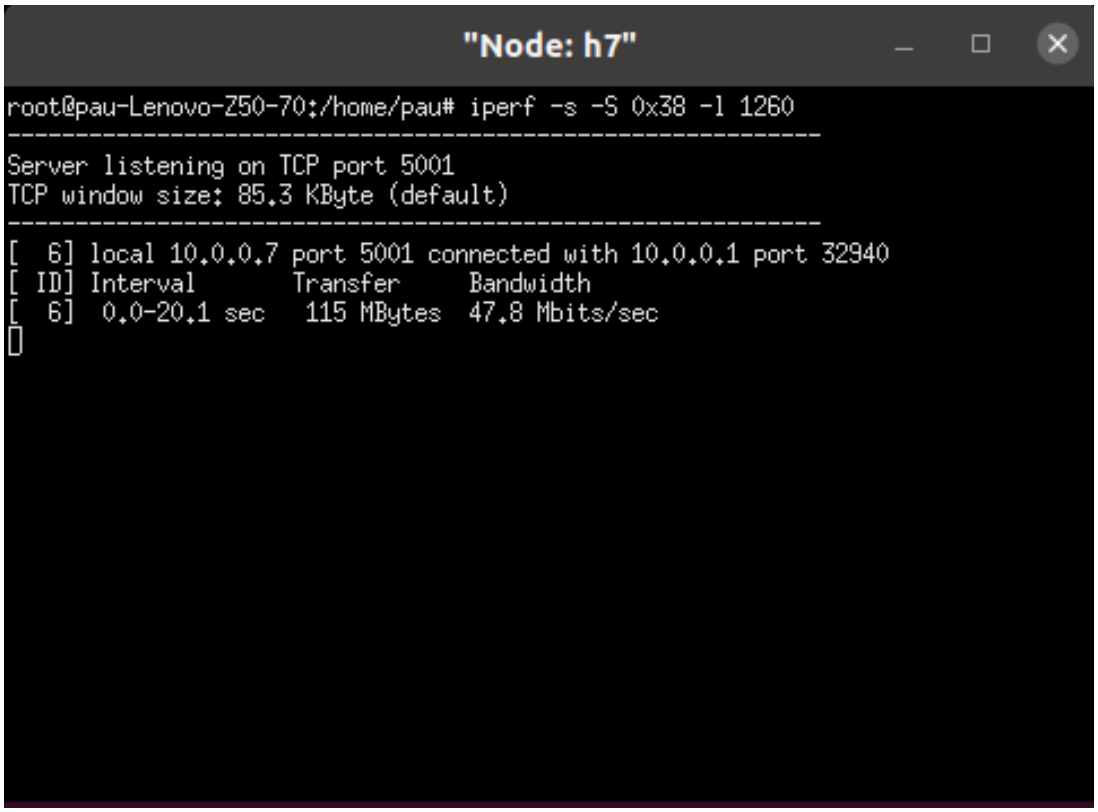


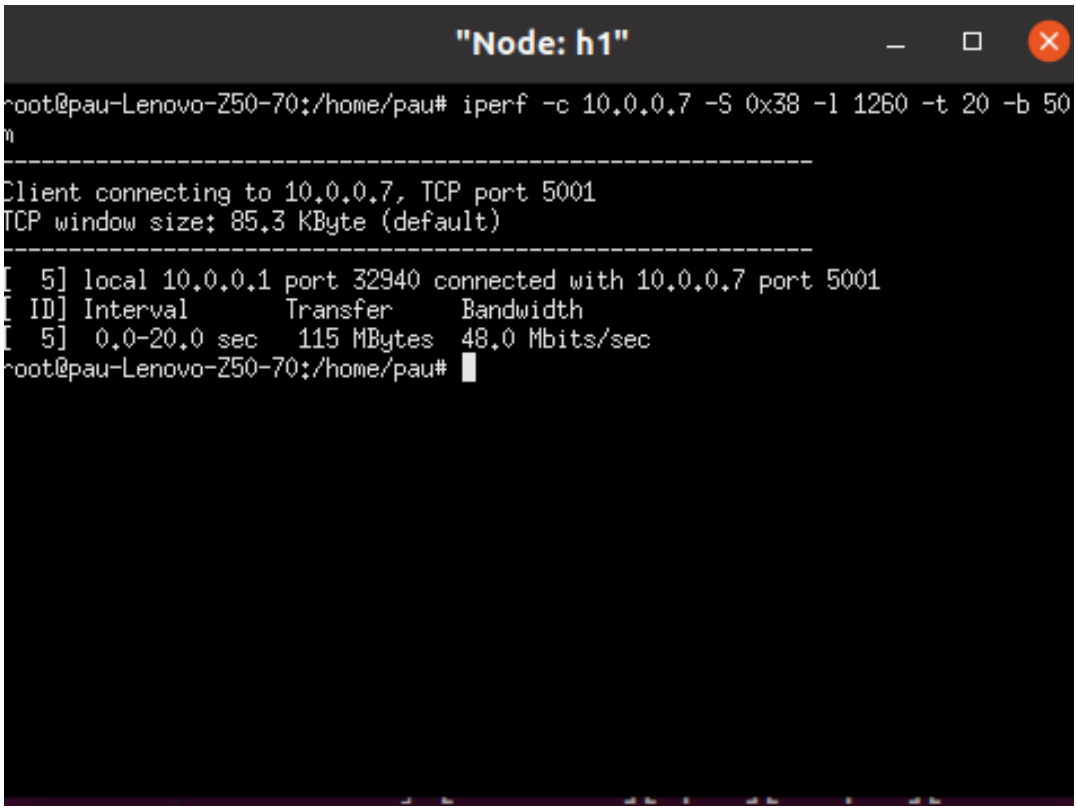
A continuació farem el mateix procediment, però per el tipus de connexió ORO i Real Time.

Per ORO, veiem que hem de comprovar que podem fer un throughput de 50 Mbps amb el host 7, que és el servidor FTP.

Fem servir la mateixa comanda, però en comptes d' especificar 600 Mbps, n' especifiquem 50. També afegirem una mida de paquet de 1260 bytes, que és la mida dels paquets FTP. Per últim, afegirem un "type of service" de 0x38, que ens dona un servei prioritari respecte els paquets best effort.

Podem veure que tots els links treballen a 50 Mbps.

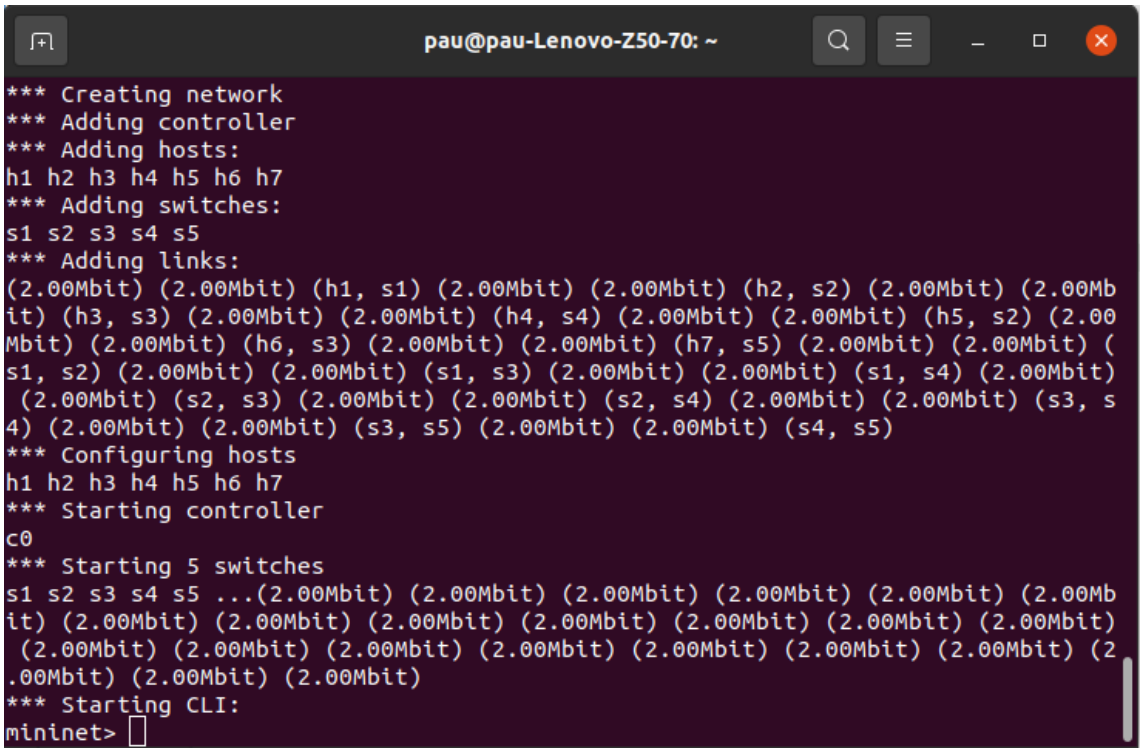




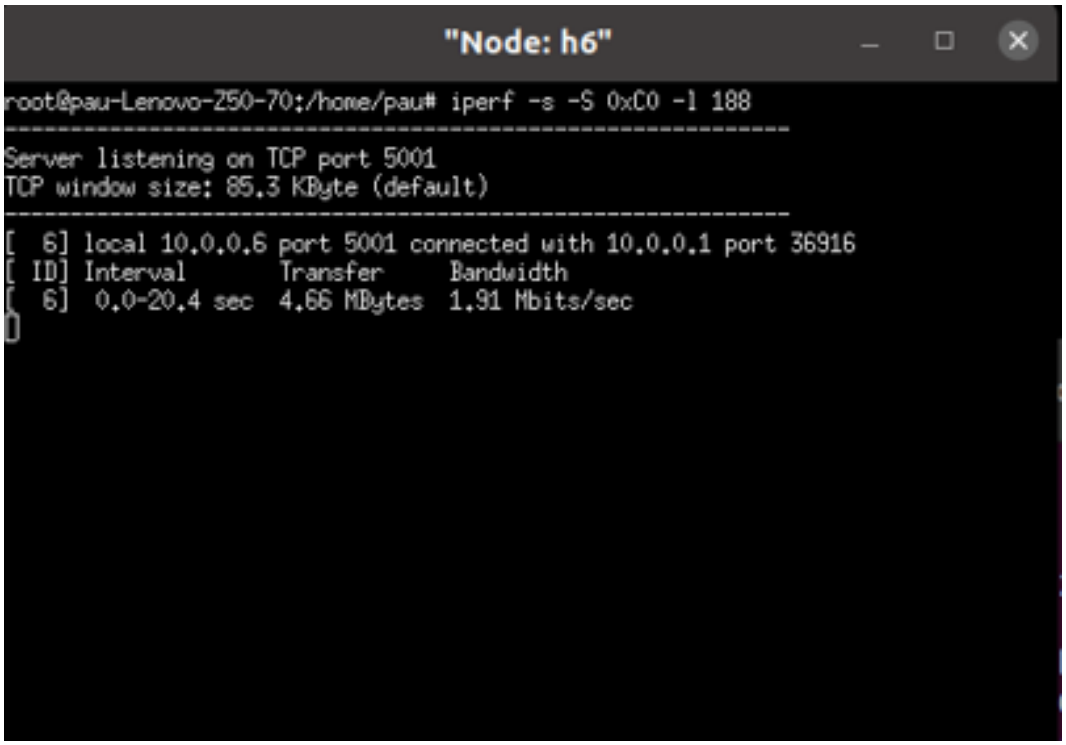
Veiem que aconseguim un throughput de 47,8 Mbps, molt proper als 50 Mbps que necessitem.

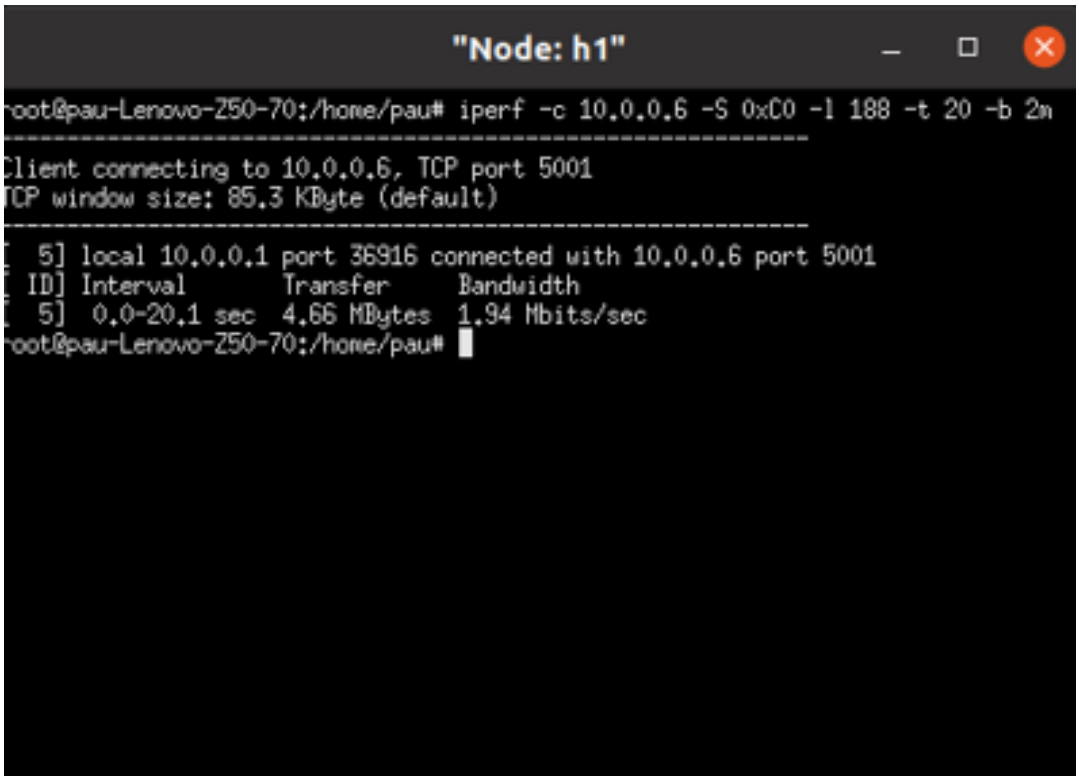
Finalment, farem la mateix comprovació per als paquets Real Time, que seran comprovats amb el host 6, que és el servidor de vídeo, i amb 2 Mbps.

A més, afegirem el paràmetre -S de IPERF que ens permet especificar el tipus de servei. El posarem al valor 0xCO per provar el servei de streaming. També farem servir la opció -l que ens permet especificar el tamany dels paquets. El posarem a 188 bytes, que és el que ocupa un paquet MPEG.



Tenim tots els links treballant a 2 Mbps.





Veiem que aconseguim un thorughput de 1,91 Mbps, un nombre molt semblant als 2 Mbps que requereix aquest servei.

A continuació, comprovarem que la xarxa SDN és capaç de respondre a la fluctuació de retard.

Els paràmetres de fluctuació de retard per la xarxa NEBA són els següents:

	Variación de retardo (percentil 95%) GPON
QoS BE	-
QoS ORO	40 ms
QoS RT	8 ms

Per fer-ho, també utilitzarem IPERF, però aquest cop farem servir paquets UDP, ja que amb paquets TCP no podem calcular la fluctuació de retard.

En aquest cas farem servir la comanda:

- iperf -s -u

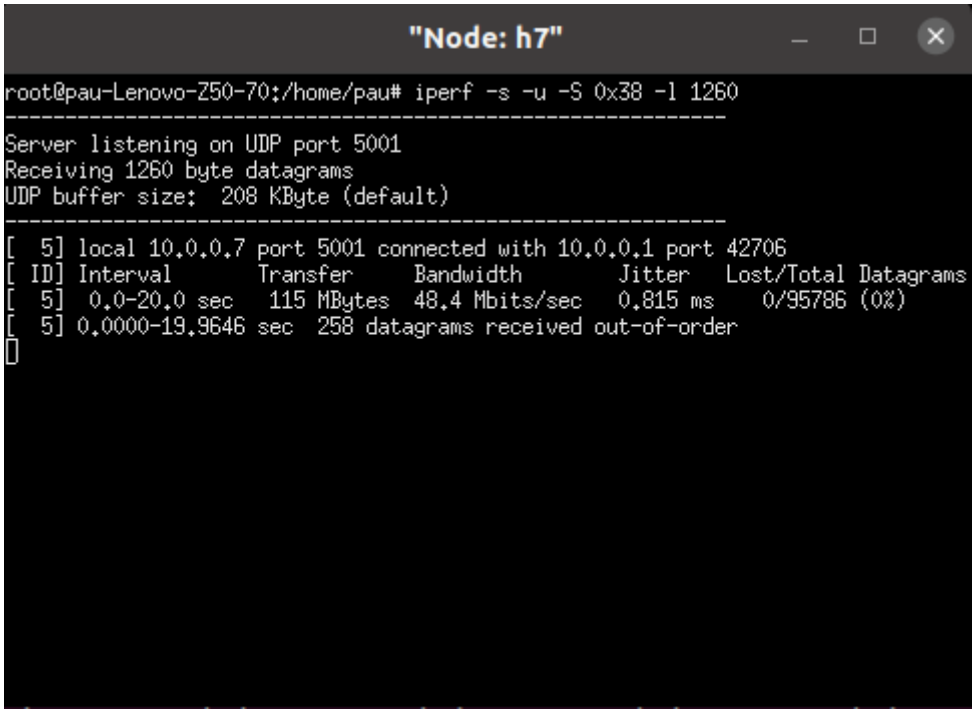
Al servidor, i la comanda:

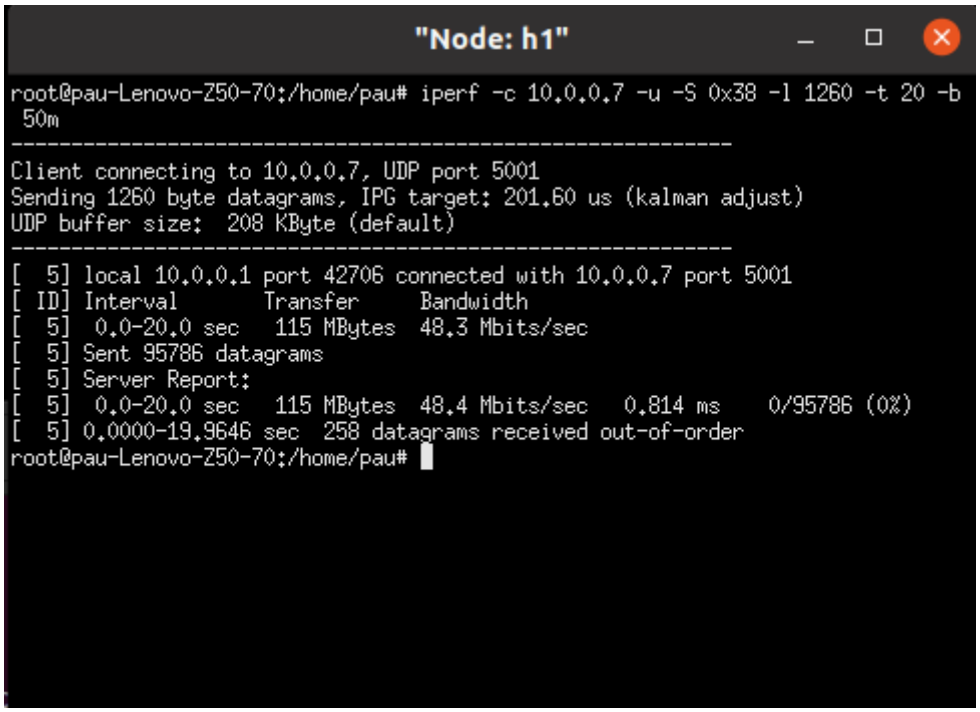
- iperf -c 10.0.0.2 -u

al client.

Només necessitem comprovar-ho per a ORO i Real Time, ja que NEBA no ens diu la fluctuació del servei Best Effort.

Primer ho farem pel servei ORO, amb el host 7, que és el servidor FTP. També farem servir els mateixos paràmetres que en l apartat de throughput.

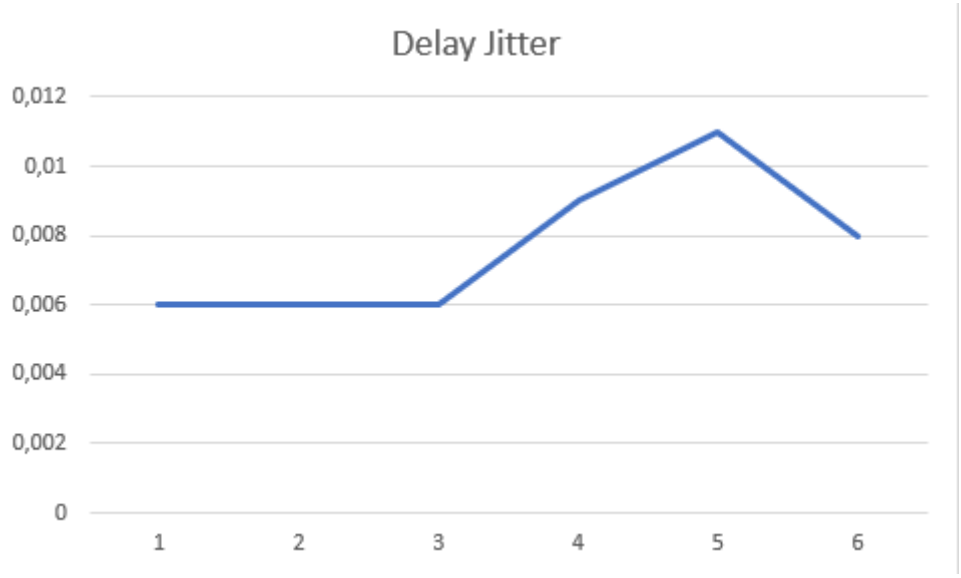




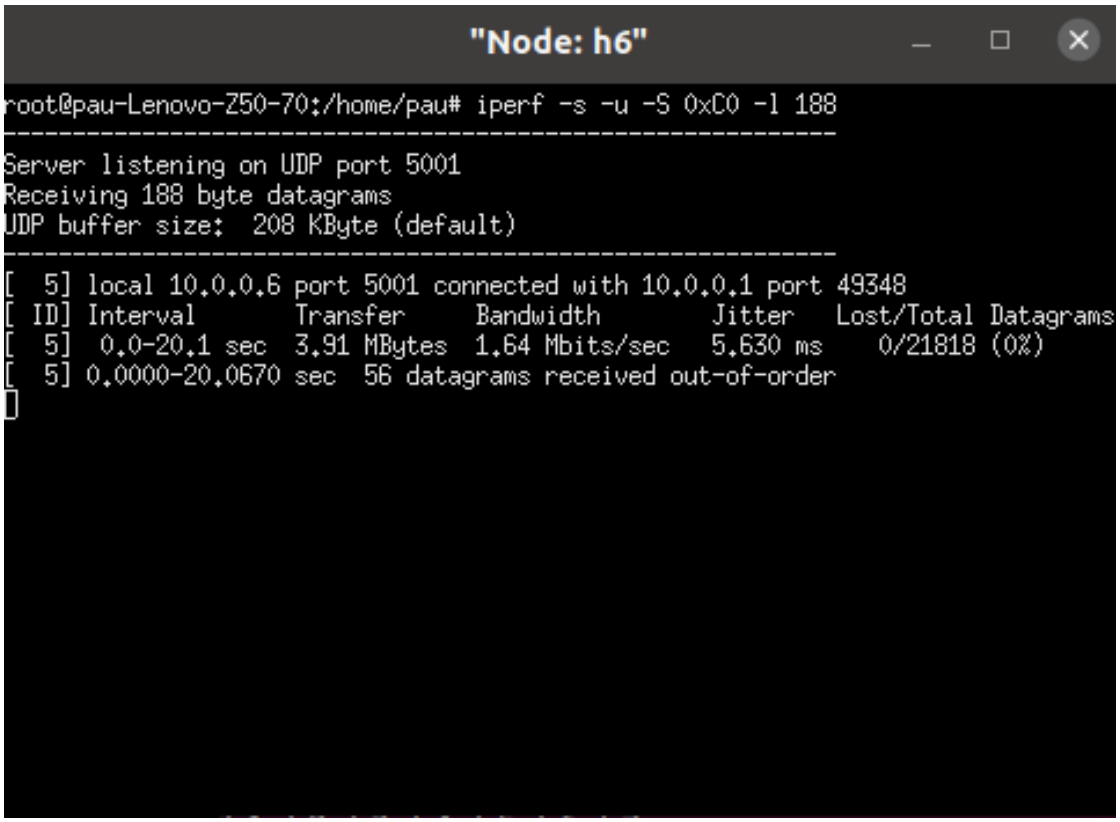
Veiem que la fluctuació de retard és 0,815

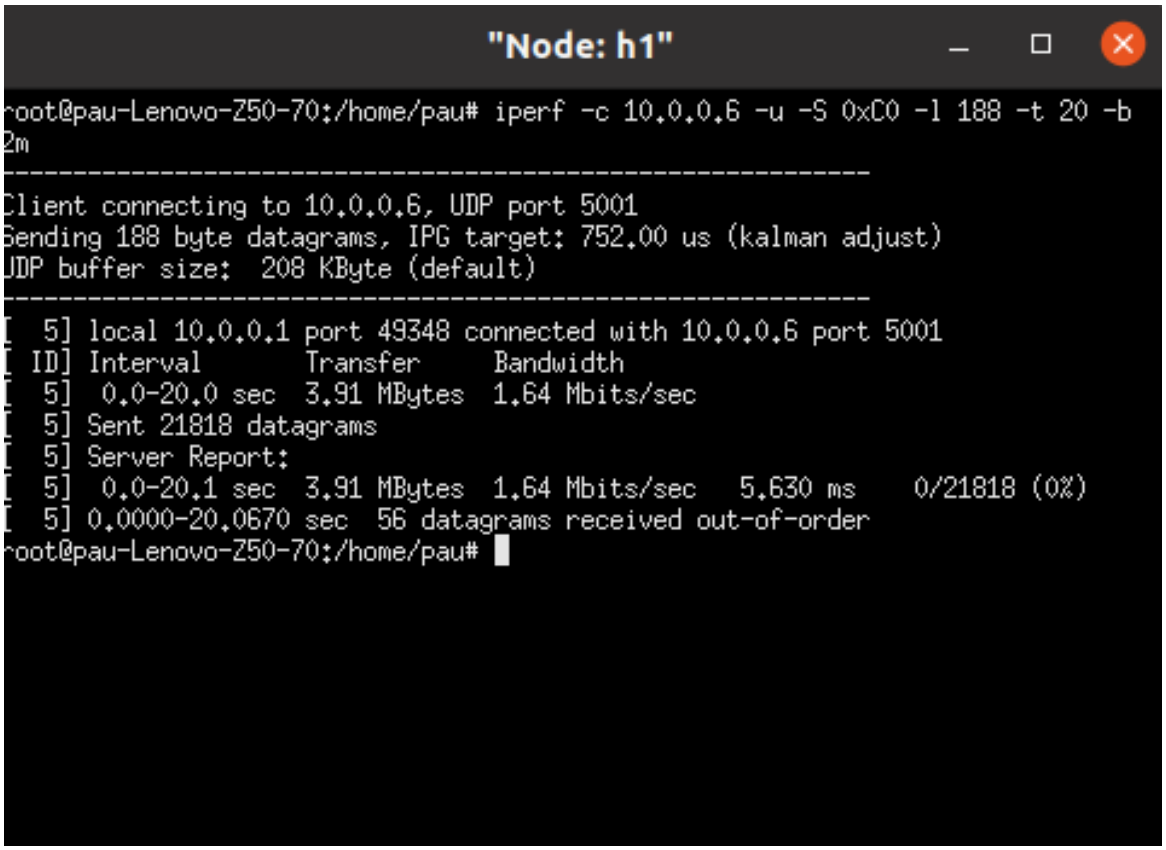
.0 ms, molt inferior als 43 ms que ens demana el servei.

Tot seguit, farem varies connexions amb IPERF al mateix host i veurem com evoluciona la fluctuació de retard:



Ara ho comprovarem per servei Real Time, que té un paràmetre de 8 ms. Per això, farem la connexió Iperf entre el host 1 i el host 6 que és el servidor de vídeo, i farem servir els mateixos paràmetres que hem fet servir per mesurar el throughput.





En aquest cas, la fluctuació de retard també és 5,630 ms, inferior als 8 ms que ens dona com a límit la xarxa NEBA.

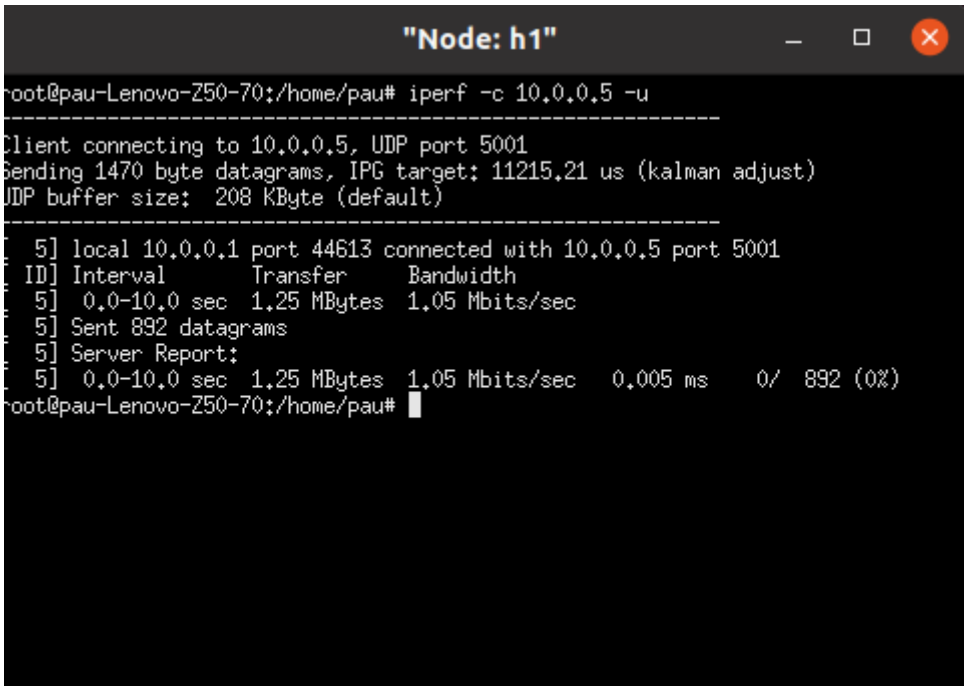
Un cop comprovat que la fluctuació de retard està dins dels paràmetres indicats, passem a comprovar que la pèrdua de paquets també ho està.

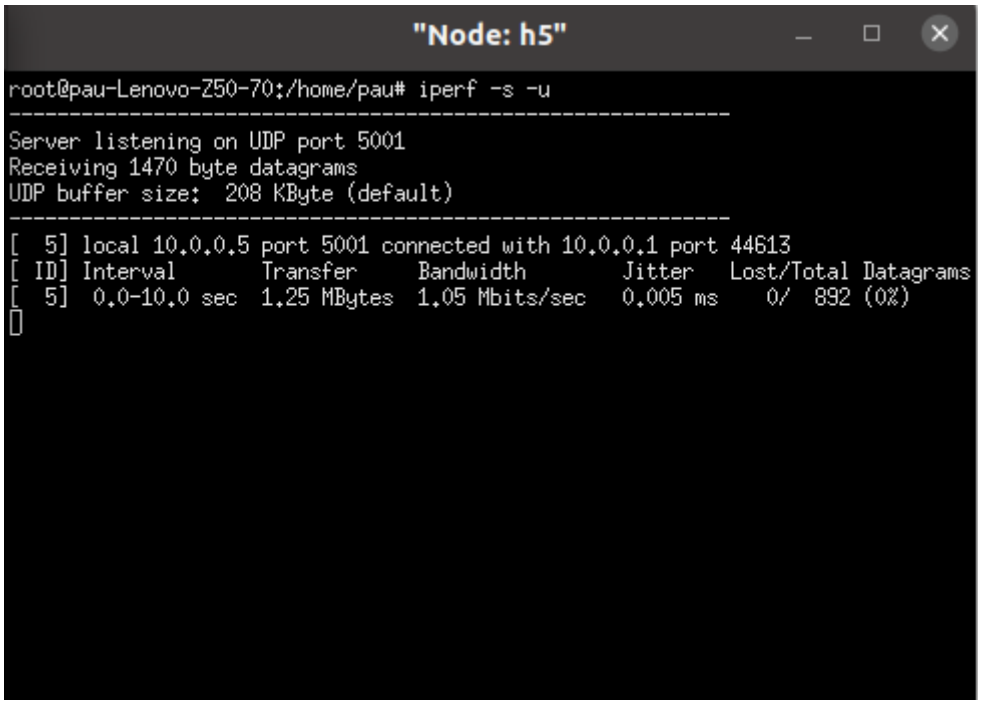
Els paràmetres per a la pèrdua de paquets són els següents:

	Valor máximo de pérdida de tramas GPON
QoS BE	0,6%
QoS ORO	0,02%
QoS RT	0,01%

En aquest apartat també farem servir paquets UDP.

Best Effort

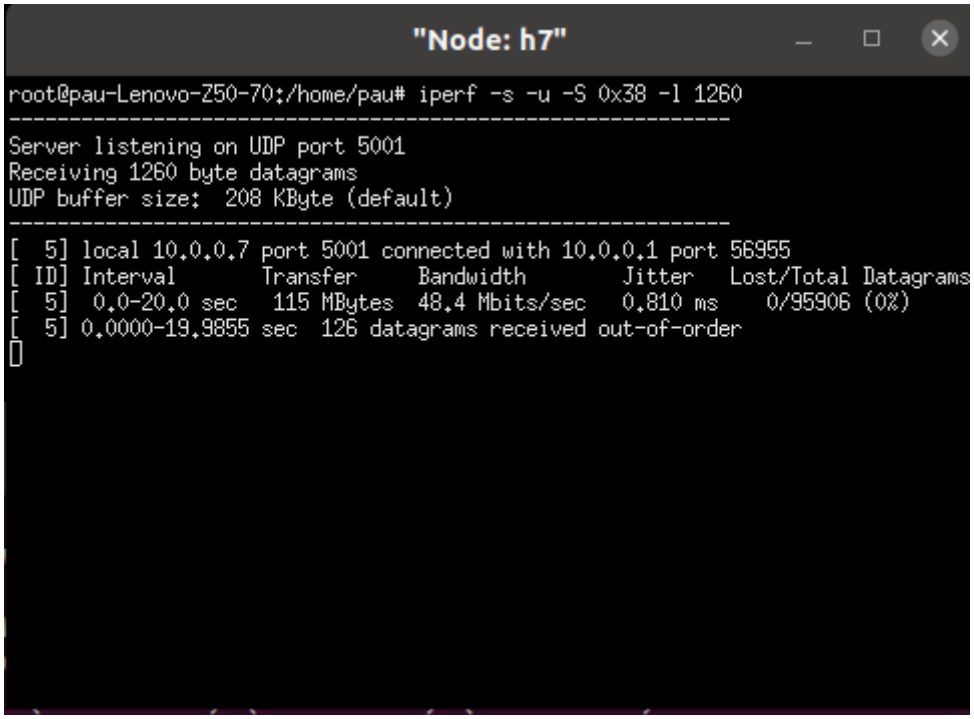


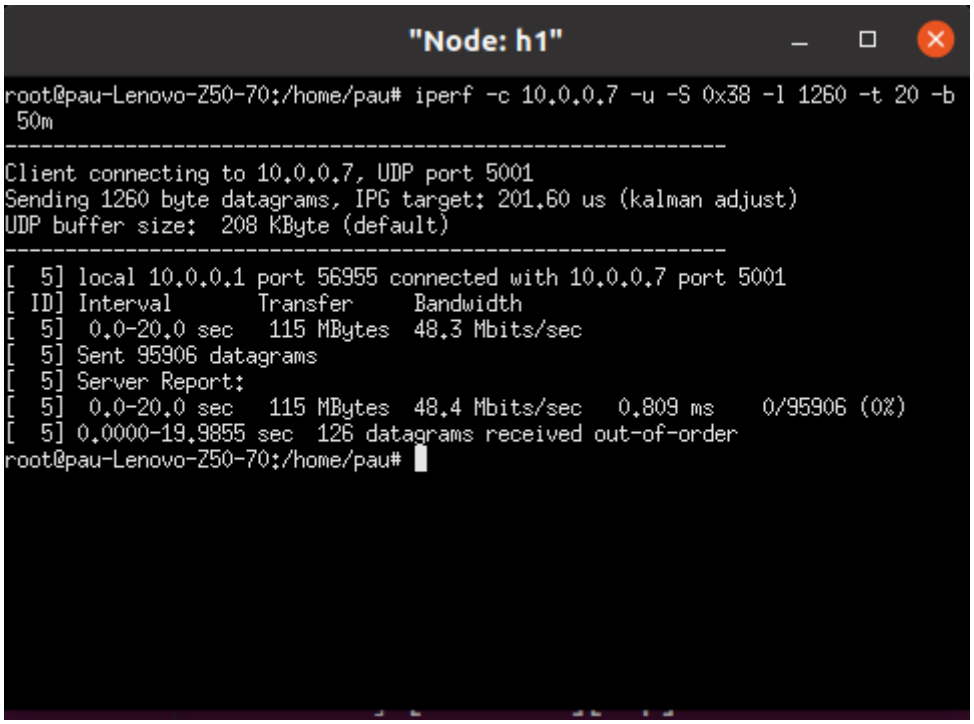


Per varies connexions a la vegada:

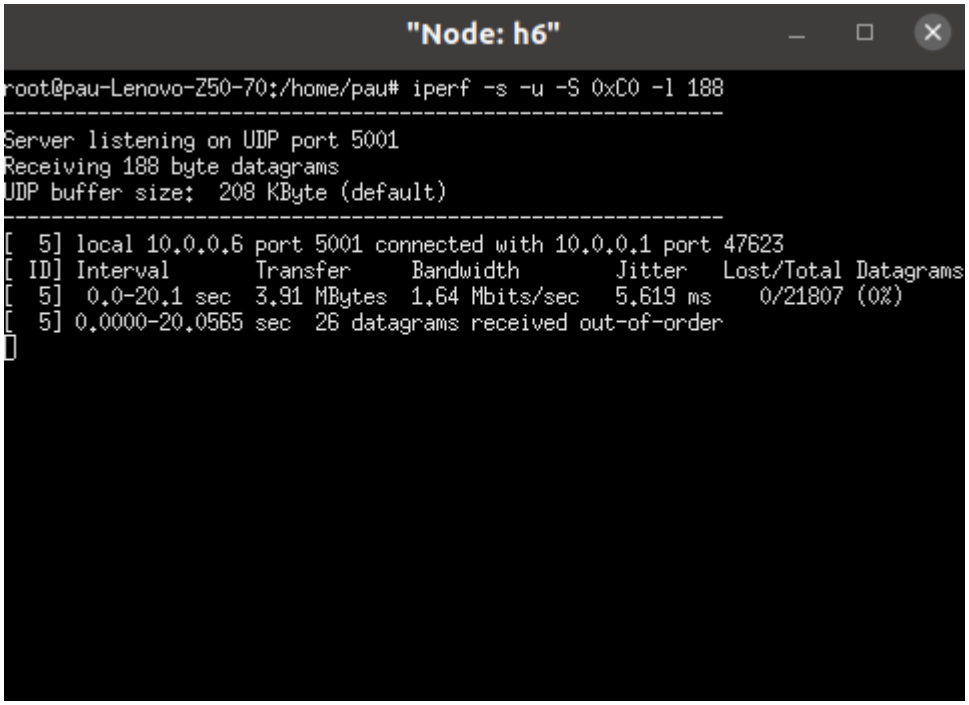

Pèrdua de paquets
1
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
1
2
3
4
5
6

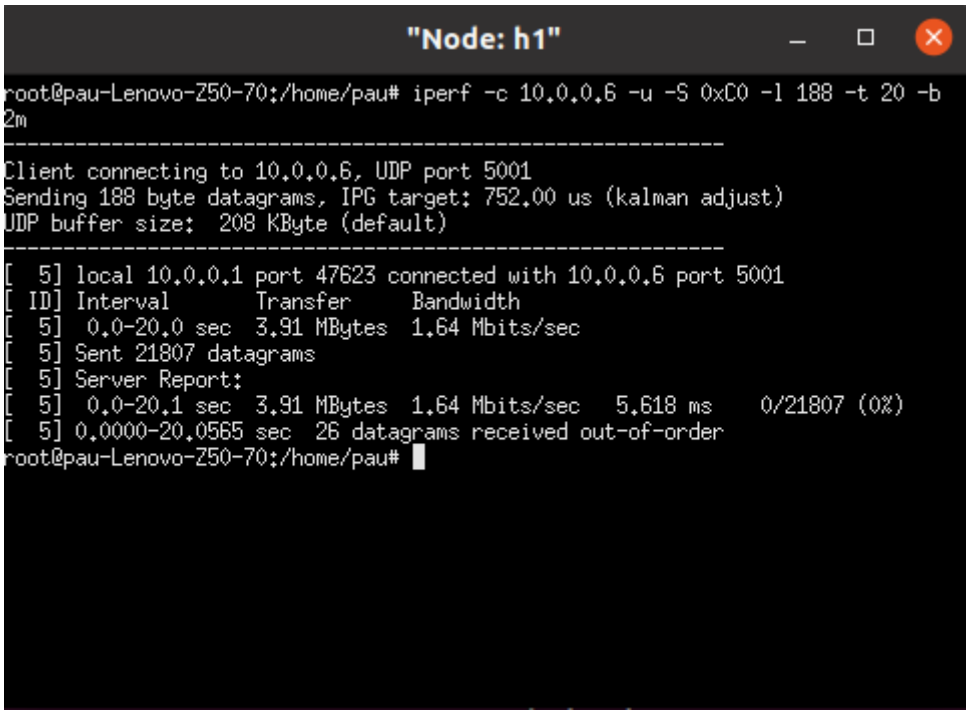
ORO


"Node: h7"
root@pau-Lenovo-Z50-70:/home/pau# iperf -s -u -S 0x38 -l 1260
Server listening on UDP port 5001
Receiving 1260 byte datagrams
UDP buffer size: 208 KByte (default)
[5] local 10.0.0.7 port 5001 connected with 10.0.0.1 port 56955
[ID] Interval Transfer Bandwidth Jitter Lost/Total Datagrams
[5] 0.0-20.0 sec 115 MBytes 48.4 Mbits/sec 0.810 ms 0/95906 (0%)
[5] 0.0000-19.9855 sec 126 datagrams received out-of-order


"Node: h1"
root@pau-Lenovo-Z50-70:/home/pau# iperf -c 10.0.0.7 -u -S 0x38 -l 1260 -t 20 -b
50m
Client connecting to 10.0.0.7, UDP port 5001
Sending 1260 byte datagrams, IPG target: 201.60 us (kalman adjust)
UDP buffer size: 208 KByte (default)
[5] local 10.0.0.1 port 56955 connected with 10.0.0.7 port 5001
[ID] Interval Transfer Bandwidth
[5] 0.0-20.0 sec 115 MBytes 48.3 Mbits/sec
[5] Sent 95906 datagrams
[5] Server Report:
[5] 0.0-20.0 sec 115 MBytes 48.4 Mbits/sec 0.809 ms 0/95906 (0%)
[5] 0.0000-19.9855 sec 126 datagrams received out-of-order
root@pau-Lenovo-Z50-70:/home/pau#

Real Time


"Node: h6"
root@pau-Lenovo-Z50-70:/home/pau# iperf -s -u -S 0xC0 -l 188
Server listening on UDP port 5001
Receiving 188 byte datagrams
UDP buffer size: 208 KByte (default)
[5] local 10.0.0.6 port 5001 connected with 10.0.0.1 port 47623
[ID] Interval Transfer Bandwidth Jitter Lost/Total Datagrams
[5] 0.0-20.1 sec 3.91 MBytes 1.64 Mbits/sec 5.619 ms 0/21807 (0%)
[5] 0.0000-20.0565 sec 26 datagrams received out-of-order



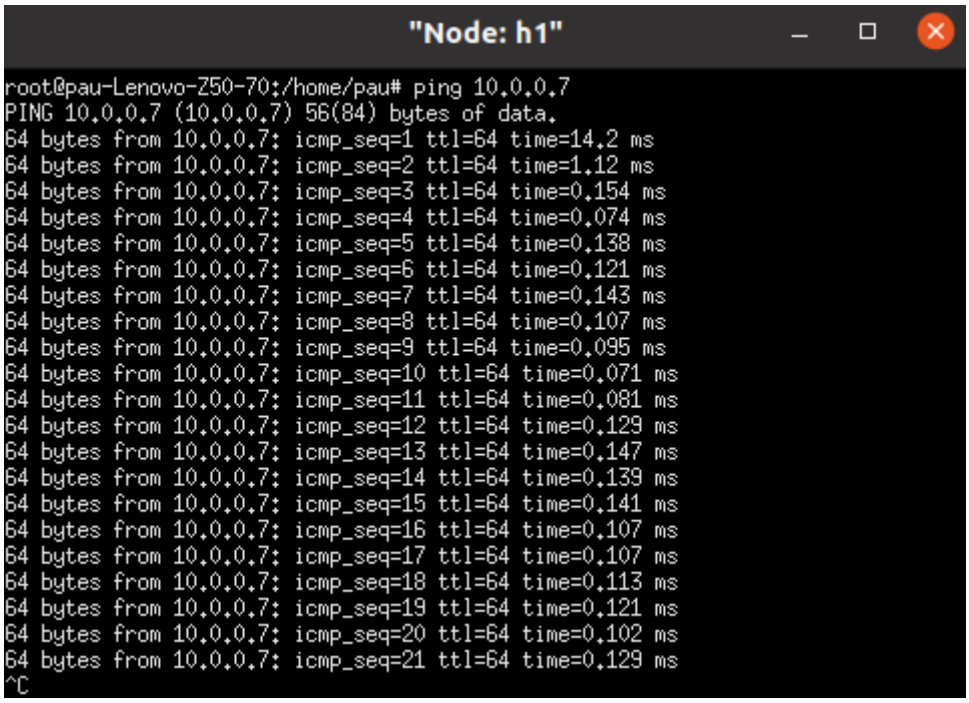
Veiem que en els tres casos, la pèrdua de paquets en la nostra simulació és del 0%. Per tant queda molt per sota dels paràmetres que estableix NEBA en qualsevol tipus de servei.

Per últim, farem la comprovació del retard. Aquesta es pot fer directament amb ping.

Els paràmetres de retard dels diferents serveis són els següents:

	Retardo medio unidireccional GPON
QoS BE	-
QoS ORO	50 ms
QoS RT	30 ms

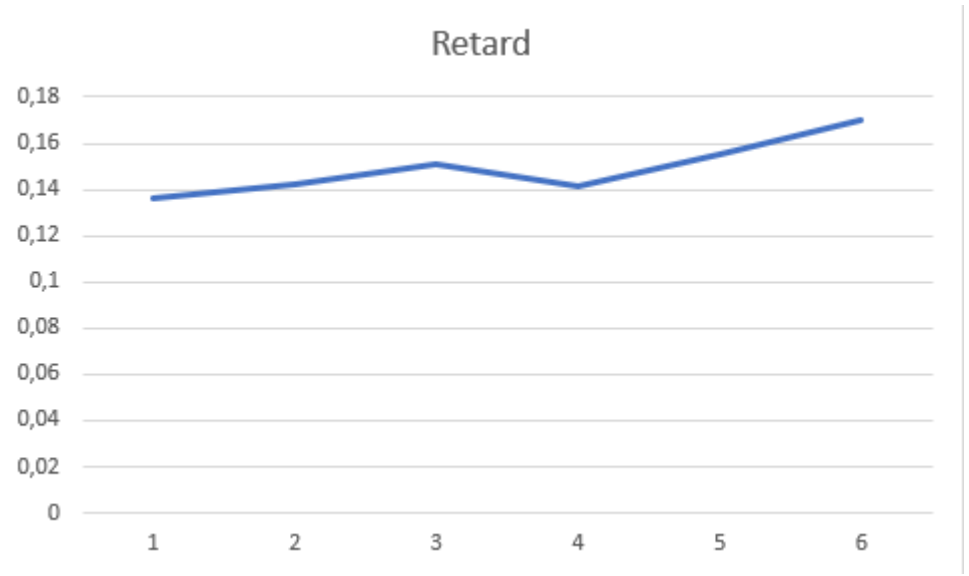
Així doncs, primer fem un ping entre el host 1 i el servidor FTP:



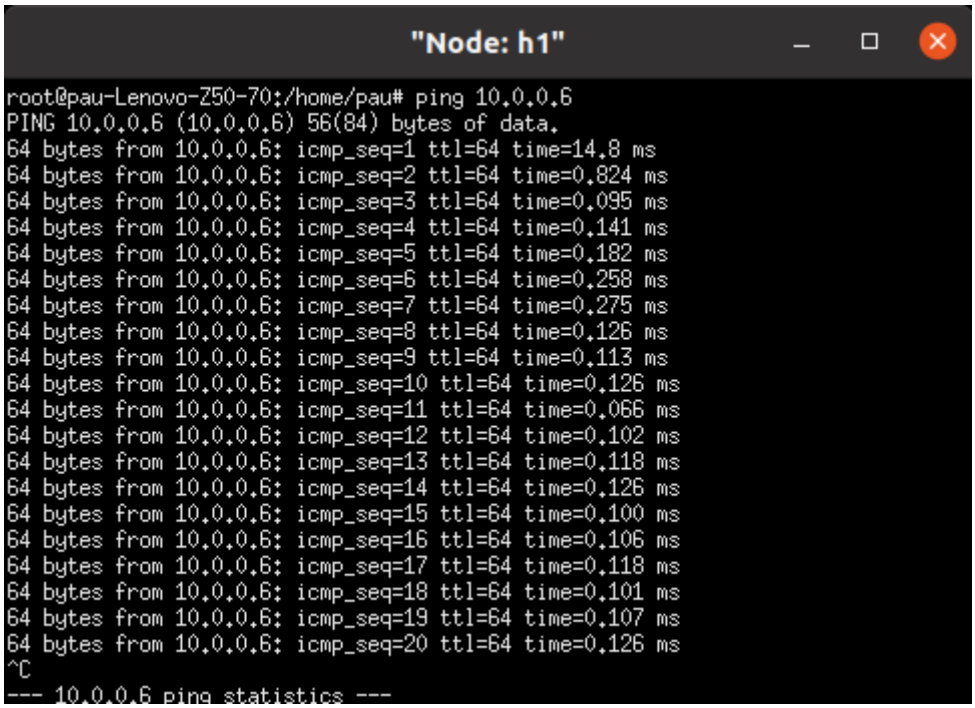
Veiem que el primer paquet té un retard de 14 ms, mentre que la resta no superen el 1 ms.

Això és degut a que el primer paquet és enviat al controlador SDN perquè s' especifiqui la ruta que ha de prendre el flux. Tot i així, tot queda molt per sota dels 50 ms que ens demana la xarxa NEBA.

Ara farem varis pings a la vegada i veurem com reacciona la xarxa:



Ara repetirem el procés per al tipus Real Time. En aquest cas no ha de superar els 30 ms de retard.



El resultat és el mateix. Ens fixem en que el primer paquet també triga 14 ms en arribar, però la resta no triga ni 1 ms. En cas cap supera els 30 ms de retard.

7. Planificació temporal

El projecte tindrà una durada aproximada de 18 setmanes. Començarà el dia 21 de Setembre de 2021 i acabarà el 27 de Gener de 2022, dia en que es farà la presentació del treball.

Les primeres 4 setmanes s' han dedicat a gestionar el projecte. Això inclou la gestió econòmica, la sostenibilitat, el context i la planificació temporal. Ens suposarà unes 80 hores de feina, que dividirem en 4 hores diàries, 5 dies a la setmana.

A partir de la cinquena setmana començarem la fase de documentació i tot seguit, el desenvolupament de la implementació, que constarà de dues fases diferenciades. Durant aquestes 14 setmanes es dedicaran 5 hores el dia, 6 dies a la setmana. Es treballaran 420 hores, per tant, el total d' hores treballades és de 500 hores.

7.1 Recursos

Han sigut necessaris una sèrie de recursos per a desenvolupar el projecte. Utilitzarem dos ordinadors. Un de portàtil, amb Ubuntu 20.04 per fer el desenvolupament i la programació de la xarxa SDN i un ordinador de sobretaula amb Windows 10, per fer la redacció dels lliuraments de GEP i la memòria final.

Els programes necessaris són:

- Editor de text: He fet servir l' editor Microsoft Word per escriure els lliuraments de GEP i la memòria.
- IDE de desenvolupament: He fet servir Sublime Text com a IDE per programar.
- GanttProject: eina de gestió de projectes que he utilitzat per fer el diagrama de Gant.
- Mininet: programa per simular la xarxa OpenFlow.

També hem de tenir en compte els recursos humans, en el nostre cas un programador i un tutor del projecte que ens ha ajudat fent reunions cada dues setmanes i ens ha assessorat en el desenvolupament del projecte.

Entre els recursos materials hi podem comptar la oficina i els mobles que hi hauran. També els dos ordinadors que he fet servir i els perifèrics que he utilitzat.

7.2 Definició de tasques

Agruparem les tasques en les diferents fases del projecte: gestió de projecte, fase de documentació, fase de desenvolupament 1 i fase de desenvolupament , i fase d' escriptura de la memòria.

7.2.1 Gestió del projecte

GP1 – Contextualització i abast

Descripció: es defineix l' abast i el context del treball. També es determinen els objectius i la metodologia a emprar.

GP2 – Planificació temporal

Descripció: es calendaritzen les tasques del projecte. També s' especifiquen els recursos necessaris i les dependències de cada tasca.

GP3 – Pressupost i sostenibilitat

Descripció: es valora el cost econòmic del projecte i se'n fa un estudi de sostenibilitat.

GP4 – Estructuració i finalització de l' etapa de gestió

Descripció: valorar la gestió del treball feta fins ara, arreglant i millorant la definició del treball.

GP5 – Reunions de control

Descripció: reunions cada dues setmanes amb el tutor per fer un seguiment de la feina feta i establir noves fites.

7.2.2 Fase de documentació

DO1 – Recerca sobre SDN

Descripció: Recerca sobre què és una SDN i quines són les millors maneres d' integrar-la.

DO2 – Recerca sobre MPLS

Descripció: Recerca sobre què és MPLS i quines són les seves desavantatges respecte SDN.

DO3 – Recerca sobre OpenFlow

Descripció: Recerca sobre què és OpenFlow i de quina manera s' integra amb SDN.

7.2.3 Fase de desenvolupament 1

DE1 – Familiarització amb GNS3

Descripció: Documentar-se sobre el funcionament del programa GNS3.

DE2 – Programació de MPLS en GNS3

Descripció: Programació d' un core network d' un ISP controlat per MPLS.

7.2.4 Fase de desenvolupament 2

DS1 – Familiarització amb Mininet

Descripció: Documentar-se sobre el funcionament del programa Mininet i Openflow en Mininet.

DS2 – Programació de OpenFlow (SDN) en Mininet

Descripció: Programació d' un core network d' un ISP controlat per SDN utilitzant el protocol OPenFlow.

7.2.5 Fase d' escriptura de la memòria

E1 – Escriptura de la fase de documentació

Descripció: Redacció de la part de memòria que correspon a la documentació.

E2 – Escriptura de la fase de desenvolupament 1

Descripció: Redacció de la part de memòria que correspon a la fase de desenvolupament 1.

E3 – Escriptura de la fase de desenvolupament 2

Descripció: Redacció de la part de memòria que correspon a la fase de desenvolupament 2.

E4 – Acabar la memòria

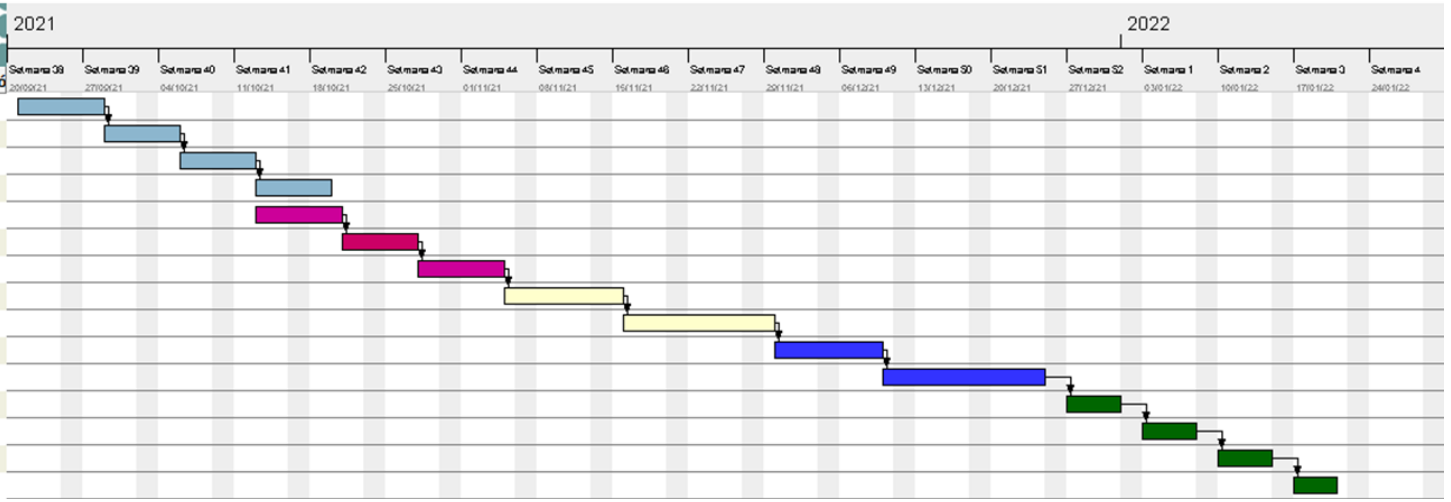
Descripció: Acabar la memòria del projecte i elaborar els documents per la lectura.

7.3 Estimació temporal

Podem veure una estimació temporal de les tasques a la següent taula:

Codi de la tasca	Temps estimat
GP1 – Contextualització i abast	20 hores
GP2 – Planificació temporal	15 hores
GP3 – Pressupost i sostenibilitat	15 hores
GP4 – Estructuració i finalització de la gestió	20 hores
GP5 – Reunions de Control	15 hores
DO1 – Recerca sobre SDN	30 hores
DO2 – Recerca sobre MPLS	30 hores
DO3 – Recerca sobre OpenFLow	30 hores
DE1 – Familiarització amb GNS3	40 hores
DE2 – Programació de MPLS en GNS3	77,5 hores
DS1 – Familiarització amb Mininet	40 hores
DS2 – Programació d' OpenFlow en Mininet	77,5 hores
E1 – Escriptura de la fase de documentació	20 hores
E2 – Escriptura de la fase de desenvolupament 1	20 hores
E3 – Escriptura de la fase de desenvolupament 2	20 hores
E4 – Acabar la memòria	30 hores

Nom	Inici	Finalització
GP1	21/09/21	28/09/21
GP2	29/09/21	05/10/21
GP3	06/10/21	12/10/21
GP4	13/10/21	19/10/21
DO1	13/10/21	20/10/21
DO2	21/10/21	27/10/21
DO3	28/10/21	04/11/21
DE1	05/11/21	15/11/21
DE2	16/11/21	29/11/21
DS1	30/11/21	09/12/21
DS2	10/12/21	24/12/21
E1	27/12/21	31/12/21
E2	03/01/22	07/01/22
E3	10/01/22	14/01/22
E4	17/01/22	20/01/22



7.4 Canvis respecte la planificació inicial

La planificació ha sofert canvis grans al llarg de tota la realització del projecte.

La idea inicial del treball era fer una comparativa entre una xarxa MPLS simulada amb GNS3 i una xarxa OpenFlow simulada amb Mininet. Això no ha pogut ser possible perquè només la part de SDN i OpenFlow ja han ocupat el temps disponible per fer el treball.

Així doncs, el treball final ha consistit en simular la xarxa SDN amb Mininet i comprovar que la xarxa és capaç de complir amb els requisits dels diferents serveis que ofereix la xarxa NEBA.

Crec que és un treball igual d' interessant, o inclús més, ja que el que ens interessa saber és si la xarxa OpenFlow és capaç de donar uns paràmetres de qualitat de servei elevats, i no ens interessa tant saber si és més eficient o menys que una xarxa MPLS.

Així doncs, la fase de desenvolupament 1 ha quedat eliminada i el temps emprat en aquesta fase ha sigut distribuït en les altres fases. La fase de documentació també ha perdut un apartat, el de MPLS. Les seves hores també han estat distribuïdes en altres etapes.

7.5 Gestió de riscos

Limitació temporal

Hem de tenir en compte que podem no tenir prou temps per fer alguna tasca. A cada etapa del treball cal valorar què farem a la següent etapa i cal mirar d' encabir allò que no ens ha donat temps de fer.

Aquest risc té un impacte baix si anem valorant periòdicament la feina feta a cada etapa. Sempre podem replantejar la feina i reordenar les tasques.

Inexperiència en les tecnologies emprades

S' ha sobredimensionat el nombre d' hores de desenvolupament les quals depenen de tecnologies on el desenvolupador del projecte no és expert.

8. Pressupost

8.1 Identificació de costos

Les despeses del projecte les podem dividir en les següents categories: costos materials i indirectes, recursos humans, contingència i imprevistos.

Els costos materials els podem dividir en dos: primer tenim els costos materials, on tindrem en compte el hardware que utilitzarem, el software i el mobiliari. Als costos indirectes hi assignarem el lloguer de l' espai de treball.

Els recursos humans del projecte són els costos dels treballadors segons el seu rol.

Finalment, tenim la contingència i els imprevistos. La contingència és un marge de seguretat per si hi ha algun sobrecost al nostre projecte, mentre que els imprevistos servirien per cobrir sobrecostos relacionats amb els riscos que hem identificat prèviament.

8.2 Estimació de costos

8.2.1 Recursos humans

Primer de tot, hem fet una identificació dels rols necessaris per dur a terme el nostre projecte. Podem distingir tres rols: cap de projecte, programador i analista de software.

El cap de projecte té la tasca de fer la gestió del projecte. El programador farà el desenvolupament de la part pràctica del treball, i l' analista de software farà l' anàlisis d' eines. En la taula següent veiem el sou per hora de cada rol, tenint en compte una jornada de 1764 hores l' any.

Rol	Sou/hora net (€)	Sou/hora brut (€)
Cap de projecte	18,24	26,07
Analista de software	15,87	22,67
Programador	7,93	11,33

Un cop definits els rols i els seus salaris, cal assignar cada rol a les tasques de planificació temporal. A la taula següent podem veure un càlcul del cost tenint en compte el nombre d' hores estimat per a cada tasca.

Codi de la tasca	Temps estimat	Rol	Preu (€)
GP1 – Contextualització i abast	20 hores	Cap de projecte	521,4
GP2 – Planificació temporal	15 hores	Cap de projecte	391,05
GP3 – Pressupost i sostenibilitat	15 hores	Cap de projecte	391,05

GP4 – Estructuració i finalització de la gestió	20 hores	Cap de projecte	521,4
GP5 – Reunions de Control	15 hores	Cap de projecte	391,05
DO1 – Recerca sobre SDN	30 hores	Analista	680,1
DO2 – Recerca sobre MPLS	30 hores	Analista	680,1
DO3 – Recerca sobre OpenFLow	30 hores	Analista	680,1
DE1 – Familiarització amb GNS3	40 hores	Programador	453,2
DE2 – Programació de MPLS en GNS3	77,5 hores	Programador	878,07
DS1 – Familiarització amb Mininet	40 hores	Programador	453,2
DS2 – Programació d' OpenFlow en Mininet	77,5 hores	Programador	878,07
E1 – Escriptura de la fase de documentació	20 hores	Cap de projecte	521,4
E2 – Escriptura de la fase de desenvolupament 1	20 hores	Cap de projecte	521,4
E3 – Escriptura de la fase de desenvolupament 2	20 hores	Cap de projecte	521,4
E4 – Acabar la memòria	30 hores	Cap de projecte	781,2

8.2.2 Costos materials i indirectes

En aquest apartat comentarem els costos indirectes i materials que hem tingut en compte per la realització del projecte.

Parlant de hardware, necessitarem ordinadors de gamma mitja-alta per poder executar els programes amb certa fluïdesa. No caldrà que tinguin processament gràfic, ja que no executarem cap programa que necessiti gran potència gràfica. L' ordinador podria ser perfectament l' HP EliteDesk 800 G6 (917 €), el web del qual es troba a les referències. Hauríem d' adquirir 3 ordinadors, un per a cada membre de l' equip.

Parlant de software, no caldrà fer cap despesa perquè tot el software que utilitzarem en el nostre projecte és gratuït.

Una de les despeses indirectes que hem de tenir en compte és el lloguer de l' espai on desenvolupar el projecte. Cal trobar un espai on puguin treballar 3 persones a la ciutat de Barcelona, aproximadament d' uns 30 metres quadrats, que és l' espai que calculo que necessitarem per encabir tots els equips. El preu mitjà del metre quadrat és de 14,5€ (està a les referències), seran 435€ al mes, i tindrà un cost total de 1740 € pels 4 mesos de feina.

8.2.3 Imprevistos

A continuació veurem cadascun dels possibles riscos i com de probable és. També en quantificarem el cost.

Limitació temporal: Els problemes de limitació temporal poden anar relacionats amb qualsevol de les tasques especificades anteriorment, per tant no podem saber a quin rol cal aplicar el sobrecost. Per tant, farem la mitjana dels salaris dels 3 rols i suposarem 20 hores de retard, el qual ens suposa un sobrecost de 400,46 €. El risc té una probabilitat mitjana.

Inexperiència en tecnologies emprades: Les tecnologies que fem servir en aquest projecte són bastant noves per a mi, i es pot produir un problema si la corba d' aprenentatge d' aquestes tecnologies no és com jo m' espero. Tot i així, aquest risc té una probabilitat baixa, perquè ja hem sobredimensionat les hores de feina per contrarestar aquest risc. Tot i així, sumem 20 hores en aquest apartat. Ens surt a 226,5 €.

8.3 Pressupost final

En aquesta taula es poden veure les despeses total del projecte. Hem calculat les contingències aplicant un 15% a cada cost.

Despesa	**Import (€)**
Cost del personal	9264,19
Costos materials	2751
Costos indirectes	1740
Contingències	2157,33
Imprevistos	627,06
TOTAL	16539,58

9.Informe de sostenibilitat

Quan fem un projecte d' aquesta envergadura és molt important pensar en l' impacte social i mediambiental que tindrà. Cal tenir en compte les conseqüències que tindrà el nostre projecte.

La dimensió social és la cara més visible de la sostenibilitat. Hi ha hagut moltes eines informàtiques que han contribuït a ajudar i millorar la societat en la qual vivim, i crec que hauria de ser la meta de qualsevol projecte que tingui en compte l' aspecte social. Cal fer projectes socialment responsables i que ajudin a equalitzar la nostra societat.

En l' apartat econòmic, cal saber dimensionar el cost d' un projecte informàtic i cal imaginar el projecte realitzat en un ambient empresarial.

Finalment, l' aspecte mediambiental el podem relacionar directament amb el consum energètic dels equips de hardware utilitzats per la realització del nostre projecte. Cal tenir una visió global i intentar entendre els beneficis i inconvenients dels projectes informàtics en termes de responsabilitat mediambiental. De primeres ens costa associar la producció d' electricitat amb un impacte negatiu pel medi.

9.1 Dimensió econòmica

Es cost estimat del projecte és força ajustat, per tant podem assegurar que no hi haurà cap malbaratament de diners, tot garantint un sou just i digne per als treballadors. Tot i així, podríem haver abaratit costos trobant un lloc de feina més barat, a una ciutat diferent de Barcelona.

9.2 Dimensió ambiental

El desenvolupament del projecte tindrà un impacte mediambiental més aviat baix perquè no involucra cap malbaratament d' energia addicional.

9.3 Dimensió social

A nivell personal, aquest treball m' ajudarà a entendre la importància d' internet i del progrés tecnològic aplicat a aquest.

Si podem millorar els protocols d' internet contribuirem a fer arribar la xarxa a més persones usuàries.

10. Conclusions

Un cop realitzat el treball de final de grau, podem concloure que les xarxes SDN són tan potents com esperàvem que fossin, i són capaces de complir amb els paràmetres necessaris per donar diferents tipus de servei.

També podem concloure que Mininet és una eina molt potent i àgil per posar a treballar simulacions de xarxes definides per software, en qüestió de minuts podem tenir llesta la nostra topologia i el desplegament triga segons en realitzar-se.

Els resultats que hem obtingut de les simulacions són molt positius. Veiem que a mida que anem augmentant el nombre de connexions el thoughput es divideix, però en el retard, la fluctuació de retard i la pèrdua de paquets no hem vist canvis significatius a mida que anem augmentant les connexions. Això ens indica que la xarxa SDN és molt robusta i que és capaç d' aguantar varies connexions simultànies sense patir.

Per últim, també hem vist que l' spanning tree és capaç de recalcular les rutes quan els enllaços cauen, però ho fa en un temps força gran. Aquí veiem que Mininet té encara marge de millora, i que no és un software perfecte. Si construíssim una xarxa SDN real segurament hauríem de resoldre aquest problema, ja que no ens podem permetre que la xarxa estigui caiguda durant 40 segons quan un enllaç falla.

11. Bibliografia

Stallings, W. (2014). *Data And Computer Communications* (10th ed.)

Kurose, J. F., & Ross, K. W. (2017). *Redes de Computadoras Un Enfoque Descendente* (7th ed.)

https://es.wikipedia.org/wiki/Redes_definidas_por_software

https://www.hackaboss.com/blog/salario-programador-espana

https://www.jobted.es/salario/analista-programador

https://testerhouse.com/procesos-testing/cuanto-cobra-un-tester-de-software-en-espana/

https://www.glassdoor.es/Sueldos/jefe-de-proyecto-sueldo-SRCH_KO0,16.htm

https://www.ccoo-servicios.es/ilunioncontactcenter/pagweb/2872.html

https://www.hp.com/es-es/shop/product.aspx?id=230C9AW&opt=ABE&sel=DTP

https://www.elindependiente.com/economia/2021/04/17/alquiler-en-barcelona-por-menos-de-900-euros-estos-son-los-precios-por-distritos/

https://www.paloaltonetworks.com/cyberpedia/what-is-quality-of-service-qos

http://www.oas.org/en/citel/infocitel/2007/junio/calidad_i.asp

https://en.wikipedia.org/wiki/Packet_loss

http://www.servervoip.com/blog/tag/tipos-de-qos/

https://es.wikipedia.org/wiki/Tasa_de_transferencia_efectiva

https://www.adslzone.net/reportajes/operadores/fibra-mayorista-que-es-neba/

https://es.wikipedia.org/wiki/Entrega_de_mejor_esfuerzo

https://www.cnmc.es/sites/default/files/2421046_8.pdf

https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

https://es.wikipedia.org/wiki/Topolog%C3%ADa_de_red

http://mininet.org/walkthrough/#custom-topologies

https://revistatelematica.cujae.edu.cu/index.php/tele/article/download/164/153/466

http://intronetworks.cs.luc.edu/current1/html/auxiliary_files/mininet/poxwiki.pdf

https://www.brianlinkletter.com/2015/09/using-pox-components-to-create-a-software-defined-networking-application/