Simulador d'enllaços WAN: desenvolupament i implementació

Miquel Ferrer i Amer
Grau en Enginyeria de Tecnologies i Serveis de Telecomunicació
Integració de Xarxes Telemàtiques

Antoni Morell Pérez
Pere Tusset Peiró

07 de juny de 2019

FITXA DEL TREBALL FINAL

Títol del treball:	*Emulador d'enllaços WAN: desenvolupament i implementació*
Nom de l'autor:	*Miquel Ferrer i Amer*
Nom del consultor/a:	*Antoni Morell Pérez*
Nom del PRA:	*Pere Tusset Peiró*
Data de lliurament (mm/aaaa):	*06/2020*
Titulació o programa:	*Grau d'Enginyeria de Tecnologies i Serveis de Telecomunicació*
Àrea del Treball Final:	*Integració de Xarxes Telemàtiques*
Idioma del treball:	*Català*
Paraules clau	*Emulació enllaços WAN* *Desenvolupament aplicacions web* *Interfícies de xarxa a Linux*

Resum del Treball (màxim 250 paraules): *Amb la finalitat, context d'aplicació, metodologia, resultats i conclusions del treball*

El Treball de Fi de Grau que s'exposa en aquest document és la clausura del l'estudi del Grau d'Enginyeria de Tecnologies i Serveis de Telecomunicació.

Aquest treball consisteix en el desenvolupament i la implementació d'un sistema d'emulació d'enllaços WAN a través del qual podrem emular un enllaç de gran distància a dins un entorn local, o xarxa LAN. D'aquesta manera, l'emulador aplicarà diferents efectes propis dels enllaços WAN a tots els paquets d'una xarxa IP que circulin pel sistema.

A més, l'aplicació que s'ha desenvolupat presenta una interfície web a través de la qual es poden gestionar les emulacions i els enllaços de xarxa del sistema d'emulació, oferint a l'usuari un entorn més amable de gestió. L'aplicació compta també amb una API REST a través de la qual altres sistemes i aplicacions poden interactuar amb l'emulació.

El desenvolupament de l'aplicació s'ha fet amb els llenguatges de programació PHP, HTML, JavaScript i CSS. Aquesta aplicació s'executa sobre un sistema operatiu Linux; concretament sobre la distribució Ubuntu Server 18.04 o superiors. A més, l'aplicació desenvolupada fa ús de diversos paquets de Linux com ara *iproute2*, *traffic control*, *ethtool* o *bridge-utils* per tal de poder aplicar els efectes desitjats als fluxos de paquets.

El treball resulta satisfactori, s'aconsegueix i es presenta un sistema d'emulació totalment funcional que permet definir els paràmetres de: latència, pèrdua i reordenament de paquets als enllaços IP que es vulguin emular.

Abstract (in English, 250 words or less):

The thesis exposed in this document it is a wrapper on the studies of the Technologies and Services of Telecommunications Engineering Degree coursed by the author.

This project consists in the development and implementation of a WAN link emulation system used to emulate long range IP links inside a local or LAN environment. Thus, the application will apply different characteristics from a WAN link to all the packets of an IP network going through the emulation system.

Furthermore, the application developed through this thesis presents a web interface through which the user can manage the emulations and network links of the emulation system, allowing the user to use a more friendly management environment. The application also presents an API REST interface though which other users can interact with the emulation system.

The application development is done using the following programming languages: PHP, HTML, JavaScript and CSS. The application is executed on top of a Linux operating system; more precisely over the distribution Ubuntu Server 18.04 or newer versions. The developed application uses a variety of Linux packages such as *iproute2*, *traffic control*, *ethtool* or *bridge-utils* so it can cause the desired effect to the packet flow.

The project results in exit, accomplishing and presenting a fully functional emulation system that allows the user to define the following parameters to the target IP links: latency, packet-loss and packet-reordering.

ÍNDEX

1. Introducció

Aquest document és la memòria del Treball de Fi de Grau titulat com "Emulador d'enllaços WAN: desenvolupament i implementació". El contingut que s'hi exposa és la descripció del treball: objectius, motivacions, procés de desenvolupament, procés de proves i resultats del mateix i conclusions.

El codi desenvolupat per al treball serà publicat a un repositori de codi extern en format de codi lliure, amb reconeixement i sense ús comercial. En qualsevol cas, en aquesta memòria es faran referències al codi i s'hi inclouran fragments del mateix.

1.1. Idea

La idea del present treball sorgeix d'una necessitat real en un entorn de laboratori per a demostració de productes de VMware. Concretament, aquest treball és fruit del període de pràctiques en empresa que vaig passar amb un conveni entre VMware i la UOC entre febrer de 2019 i març de 2020.

Al novembre de 2017 VMware va comprar l'empresa VeloCloud per 449 milions de dòlars. L'empresa adquirida oferia un sistema de SD-WAN molt competitiu que encaixa en el model de negoci de VMware, consistent en la definició per software de tots els serveis de centre de processament de dades: còmput, xarxa, emmagatzematge, etcètera.

Amb aquest pretext, i tenint en compte que la meva tasca durant el període de proves era la gestió d'un laboratori de VMware a Madrid per a la demostració de les solucions de l'empresa als potencials clients, ens trobem amb la necessitat d'utilitzar una eina per a emular un enllaç WAN dins un entorn local i poder demostrar les funcionalitats de VeloCloud.

Per a aconseguir l'emulació d'enllaços WAN, fins a la data, s'havia utilitzat una solució desenvolupada per TATA Consulting Services al 2014 anomenada WANem. El problema d'aquest software, com s'analitza més endavant és que està desactualitzat i obsolet, de forma que es fa necessària una alternativa.

Després d'un breu anàlisi de alternatives a l'ús de WANem, apareixen algunes opcions vàlides, però cap de les solucions s'adapta exactament als requisits,

de forma que finalment sorgeix la idea del desenvolupament d'una solució pròpia que s'adapti a les necessitats de l'entorn.

1.2. Anàlisi de solucions existents

Com s'ha esmentat, fins a la data s'havia fet ús de l'emulador d'enllaços WAN de TATA Consulting Services, una solució que ja no és vàlida, entre altres coses perquè està desenvolupada sobre una versió de Linux que ja no rep suport, de forma que els paquets del sistema operatiu no es poden actualitzar.

A més, WANem està desenvolupat fent ús de paquets que ja no estan presents a les darreres versions de Ubuntu, de forma que la instal·lació del codi de WANem sobre sistemes operatius amb suport no és possible sense una adaptació prèvia del codi o dels paquets del sistema operatiu.

Per aquests motius, la solució WANem s'ha descartat com a una solució d'emulació d'enllaços WAN i es fa necessària la recerca d'altres mètodes. El principal requisit que s'imposa és que, igual que amb WANem, l'alternativa escollida sigui de software lliure, ja que s'implementarà en més laboratoris de VMware a altres seus d'altres països, i no interessen els sistemes propietaris.

Amb aquest primer requisit ja es descarten totes aquelles solucions propietàries, pensades per a entorns diferents, més bé productius. Aquestes solucions poden esser, per exemple: NO-ONE o INE de ITRINERGY [1] o Netropy de APPOSITE Technologies[2].

Un altre requisit que s'imposa és que el sistema que s'utilitzi tingui una interfície gràfica accessible mitjançant un navegador web, que permeti visualitzar fàcilment els paràmetres de l'emulació i interactuar-hi. Així mateix, també es desitja poder tenir funcionalitats com ara la gestió de ponts (*bridge*) d'interfícies de xarxa.

Hem de buscar, per tant propostes de codi obert. Per això, tenint en compte que WANem es basa fonamentalment en el paquet "tc-netem", de Linux, busquem propostes que aportin una interfície gràfica a aquest paquet i hi

[1] https://itrinegy.com/networkemulators/
[2] https://www.apposite-tech.com/wan-simulation/

apareixen diverses solucions, molt semblants entre elles, com per exemple els projectes publicats a GitHub: AlexMeuer/tcgui[3] o tum-lkn/tcgui[4].

El problema que presenten aquests projectes, és un excés de senzillesa, ja que, si bé ofereixen una interfície gràfica per al paquet "tc-netem", no ofereixen cap altre funcionalitat, i són força limitats. A més, presenten un codi excessivament senzill, difícilment escalable.

Arribats a aquest punt, l'opció que es planteja ja és el desenvolupament propi o la modificació d'un dels projectes existents. Malauradament, aquesta segona opció es descarta ja que els projectes esmentats estan desenvolupats en Python i no és un llenguatge de programació del qual tingui prou coneixement com per embarcar-me en un projecte d'aquestes dimensions.

1.3. Motivació personal

Aprofitant aquesta situació, se'm desperta un interès en el desenvolupament d'una aplicació pròpia servint-me dels coneixements previs que ja es tenen sobre desenvolupament d'aplicacions web, adquirits en experiències laborals anteriors.

Aquest desenvolupament, si resulta d'èxit, podria esser utilitzat en l'entorn de laboratori, per resoldre els problemes existents. A més, altres laboratoris de xarxes amb unes característiques similars es podrien beneficiar d'aquesta aplicació.

Finalment, el desenvolupament d'una aplicació de codi obert i software lliure que pugui rebre aportacions, millores, i correccions d'altres usuaris, i que sigui gratuït, s'alinea amb la filosofia de l'autor i desenvolupador del treball.

[3] https://github.com/AlexMeuer/tcgui
[4] https://github.com/tum-lkn/tcgui

2. Objectius

El principal objectiu d'aquest Treball de Fi de Grau és el desenvolupament d'un emulador d'enllaços WAN que, de moment, permeti parametritzar, fonamentalment, tres característiques d'aquest enllaç: latència, pèrdua i reordenament de paquets.

Es pretén desenvolupar un projecte tan modular com sigui possible, de forma que sigui un codi escalable i s'hi puguin anar implementant més característiques a posteriori, com ara: creació i gestió d'interfícies pont, gestió de taules d'encaminament, o gestió d'adreçament de les interfícies de xarxa.

Aquest projecte es planteja com un projecte de llarg termini i de codi obert, de forma que el recorregut del desenvolupament de funcionalitats no es limita al que es presenti en aquest treball; en qualsevol cas, però, el treball suposa la definició de l'esquelet de l'aplicació i de les seves funcionalitats claus.

L'aplicació d'emulació d'enllaços WAN es caracteritza per tenir una interfície web a través de la qual l'usuari podrà interactuar amb l'aplicació i gestionar de forma gràfica els paràmetres de l'emulació. A més, s'hi incorpora una interfície API REST per poder gestionar l'emulació a través d'*scripts* de tercers.

Coneixent altres entorns de laboratori, es determina que l'aplicació desenvolupada ha de poder funcionar en quatre topologies de xarxa diferent que s'introdueixen a continuació, però tanmateix s'hi aprofundirà més endavant en aquest document:

- *Single arm*: el dispositiu que actua d'emulador d'enllaços només té una interfície de xarxa.
- *Dual arm*: el dispositiu emulador té dues interfícies de xarxa.
- *Triple arm*: el dispositiu emulador té tres interfícies de xarxa.
- *Multiple arm*: el dispositiu emulador té més de tres interfícies de xarxa.

En resum, aleshores, podem determinar la següent llista d'objectius:

- Desenvolupar una aplicació que permeti emular el comportament d'un enllaç WAN establint alguns paràmetres característics d'aquests tipus d'enllaç. L'aplicació s'ha de desplegar sobre una màquina amb Ubuntu Server i els paquets de l'enllaç que es vulgui emular hauran de passar a

través d'aquest dispositiu, ja sigui amb mètodes d'encaminament o fent ús d'interfícies pont.

- L'aplicació ha de permetre introduir, com a mínim, els següents paràmetres en l'emulació: retard i variació de retard; reordenament de paquets i correlació; pèrdua de paquets i correlació. Tots els paquets d'un enllaç IP determinat es veuran afectats segons aquests paràmetres.
- L'aplicació ha de tenir un portal web perquè l'usuari pugui interactuar-hi i una interfície API REST perquè sistemes externs també hi puguin interactuar. Aquestes dues interfícies permetrien establir els paràmetres de les emulacions i gestionar-les.
- L'aplicació s'ha de poder desplegar en dispositius amb múltiples configuracions d'interfícies de xarxa, des d'una interfície, fins a més de quatre. D'aquesta forma, s'aconsegueix un emulador més flexible i adaptable als diferents entorns i laboratoris.
- El codi de l'aplicació ha de ser escalable. S'han de poder ampliar les funcionalitats del projecte fàcilment, sense interferir en les funcionalitats existents.

3. Viabilitat

Com s'ha esmentat en el principi del treball, l'eina que s'ha desenvolupat a través d'aquest treball es distribueix en format de programari lliure i de codi obert, responent a la filosofia que permet l'ús, estudi, distribució i millora del programari de forma totalment lliure i gratuïta.

En qualsevol cas, però, es pretén presentar un projecte econòmicament viable, tot i que el desenvolupament de l'aplicació ja està fet i no se n'espera cap rendiment econòmic, com ja s'ha explicat.

En els capítols anteriors s'ha explicat que el projecte naix de la necessitat de poder demostrar dins un entorn de laboratori unes característiques determinades per a un enllaç IP. La necessitat de caracteritzar aquest enllaç resideix en poder demostrar als potencials clients d'altres solucions (com en el cas de solucions SD-WAN) el seu correcte funcionament.

D'aquesta forma, per tant, la justificació econòmica d'aquest projecte es pot encabir dins la necessitat d'una eina que permeti emular un enllaç WAN per tal de poder demostrar el correcte funcionament d'un producte comercial que sí té un cost per al client final.

Per tant, es pot considerar que el cost de desenvolupament d'aquesta aplicació serà contemplat dins el cost del procés comercial de la venta d'una solució diferent.

Per altra banda, es proposen dues formes de monetització addicionals que podrien cobrir les totes les despeses del desenvolupament en un termini curt:

La primera solució consisteix en els desenvolupaments específics, que inclou la possibilitat de contractar el desenvolupament de funcionalitats addicionals de forma específica per a un client determinat. Aquests desenvolupaments, però, podrien esser afegits a posteriori al codi font original de l'aplicació.

La segona solució consisteix a oferir suport. S'oferiria als clients que requereixin suport per part de la pròpia entitat desenvolupadora de l'aplicació. Aquest suport es donaria per a qualsevol qüestió relacionada amb la pròpia aplicació, fugint d'oferir serveis de consultoria en matèria d'arquitectura de xarxes.

4. Arquitectura i funcionament de l'emulador

4.1. Fonaments de l'emulació

Abans d'entrar en el desenvolupament de l'aplicació, convé entendre el funcionament de l'emulació, i com es pretén que aquesta funcioni.

L'emulador que es pretén desenvolupar treballa dins d'una xarxa d'ordinadors connectats entre si per una xarxa basada en la *suite* de protocols d'Internet. La idea fonamental resideix a intercalar el dispositiu d'emulació enmig de l'enllaç que es vol emular.

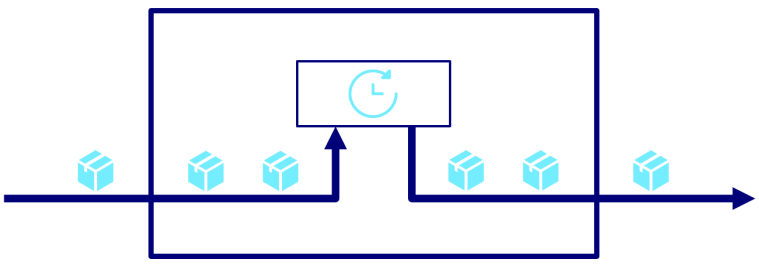
D'aquesta forma, l'emulador quedaria a mig camí entre dues o més interfícies de xarxa d'un o més equips i permetria, d'aquesta forma, interactuar amb els paquets que hi circulin, produint els efectes que es desitgin introduir en el procés d'emulació.

En concret, aquests efectes són: retard, reordenament de paquets i pèrdua de paquets; tres efectes molt comuns dins un enllaç d'una xarxa d'Internet de gran abast, o WAN. Veiem a continuació, breument, en què consisteixen aquests efectes:

4.1.1. Retard

Entenem el retard com a el temps que triga un paquet a anar d'un punt a un altre. El retard en un enllaç WAN pot esser variable, però dins una xarxa LAN, sol ser inferior a un mil·lisegon o de pocs mil·lisegons.

Per tant, quan a l'emulació s'hi afegeixi retard, el comportament esperat del dispositiu serà interceptar cada paquet, guardar-lo el temps que s'hagi determinat, i tornar-lo a posar a la xarxa perquè continuï el seu camí.

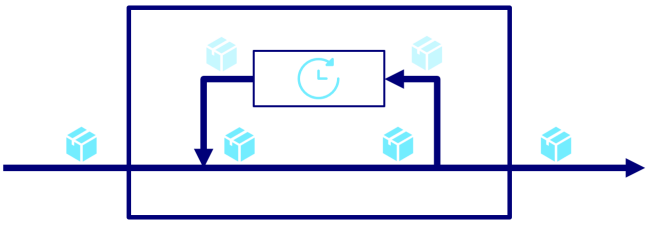


Il·lustració 1: Retard

4.1.2. Reordenament de paquets

Aquest efecte es produeix per la naturalesa multicamí dels enllaços WAN. Com bé es coneix, per interconnectar dues màquines que es troben allunyades, es poden seguir múltiples camins. Això produeix que en ocasions, els paquets arribin al destinatari de forma desornada, obligant-lo a reordenar-los o a sol·licitar una nova transmissió que arribi amb l'ordre establert.

D'aquesta manera, quan a l'emulador se li determini que ha de afegir reordenament de paquets, haurà de reproduir aquest comportament. Això es pot aconseguir, senzillament, agafant un nombre de paquets de la transmissió i afegint-hi un retard, de forma que els paquets seleccionats arribaran més tard al destinatari.

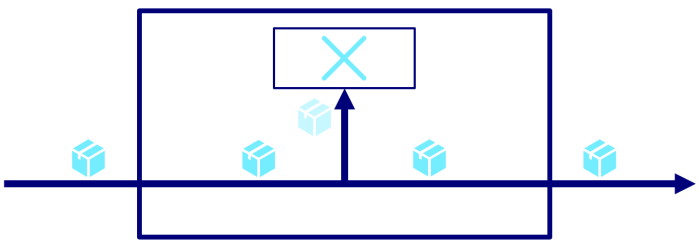


Il·lustració 2: Reordenament

4.1.3. Pèrdua de paquets

Un altre efecte que es produeix en enllaços de gran abast és la pèrdua de paquets. Ja sigui per un error en algun dels nodes que interconnecten els equips, per un problema en la transmissió o recepció d'un paquet, o per un excés de càrrega d'un enllaç, es pot donar el cas que un paquet es perdi a la xarxa.

Aquest efecte és molt senzill d'emular, ja que simplement consisteix en que un nombre determinat de paquets siguin eliminats del flux de transmissió i no es tornin a posar a la xarxa.



Il·lustració 3: Pèrdua

4.2. Dispositiu d'emulació

Com s'ha vist, és necessari interceptar l'enllaç que es vol emular per tal de poder introduir les diferents característiques i paràmetres de l'emulació. Per aconseguir això, podem utilitzar diferents mecanismes en funció de la xarxa de què es disposi.

4.2.1. Físic o virtual

En primer lloc, convé determinar si l'enllaç que s'ha d'interceptar és físic o virtual. És a dir, si ens trobem en un entorn amb segments de xarxa virtuals, o si, per contra, la xarxa és física.

Igualment, hem de veure si l'origen o el destí del trànsit són dispositius físics o virtualitzats i si hi tenim accés o no. Amb això, podrem determinar si podem desplegar l'emulador com a un dispositiu virtual en un entorn de virtualització com pugui esser VirtualBox, Parallels, VMware Virtualization o similars.

Si per contra l'enllaç és físic i no podem interceptar l'enllaç de forma virtual, haurem de desplegar l'emulador en format físic amb les seves respectives connexions. En qualsevol cas, el codi del projecte, pot esser desplegat en qualsevol dels dos formats sense fer distinció en la configuració del sistema.

4.2.2. Nombre d'interfícies de xarxa

Tal i com s'ha introduït anteriorment en aquest mateix document, una altra consideració a l'hora de desplegar el dispositiu d'emulació, és el nombre d'interfícies de què es disposen.

El dispositiu desenvolupat pot funcionar amb una o més interfícies de xarxa, com es veurà més endavant, però, en qualsevol cas, hi ha una sèrie de condicions que s'han de valorar abans d'establir el nombre d'interfícies.

En primer lloc, s'ha de saber si es pot disposar d'una interfície específica per a la gestió de l'emulació. Això permetrà aïllar la gestió de l'emulació de la pròpia emulació, evitant d'aquesta manera que els paràmetres establerts, afectin el rendiment de la gestió del dispositiu.

També s'ha de considerar si es pot fer ús d'interfícies en mode promiscu. Això permetria establir un pont que traspassi els paquets d'una interfície a una altra amb els paràmetres de l'emulació, treballant només en capa 2 i

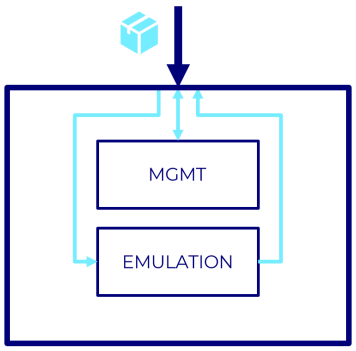
permetent no parar esment en paràmetres de la capa de xarxa com ara l'encaminament.

És important considerar aquest punt a l'hora de fer el desplegament ja que, com s'ha indicat, si no es pot establir una interfície pont, caldrà tenir en ment els paràmetres d'encaminament involucrats en la xarxa.

Veiem aleshores les diferents topologies possibles amb més detall:

- *Single arm* o una única interfície

Aquesta topologia representa el desplegament més senzill possible pel que fa a la configuració de la màquina, per contra, és el desplegament menys recomanat ja que una única interfície haurà de servir com a entrada i sortida del trànsit de l'enllaç WAN emulat, a més d'encabir el trànsit de gestió.



Il·lustració 4: Arquitectura d'una sola interfície

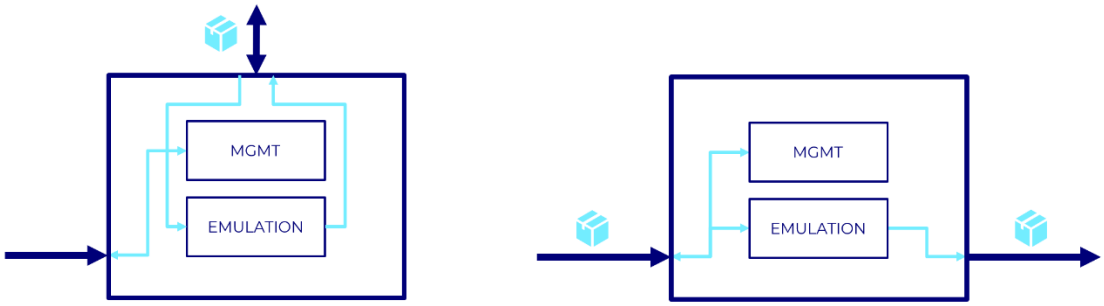
A més, aquest tipus de desplegament obliga a interactuar amb la capa 3, de forma que, en determinats entorns pot complicar molt la gestió de rutes d'encaminament.

- *Double arm* o dues interfícies

Aquesta topologia utilitza dues interfícies de xarxa. Aquest desplegament es pot entendre de dues formes: per una banda es pot fer ús d'una de les interfícies exclusivament per a gestió, mentre que l'altra s'utilitza per l'emulació. D'aquesta forma s'aconsegueix aïllar els dos tipus de trànsit, fet que permet no interferir en la gestió.

Per altra banda, es pot desplegar usant una interfície de gestió i que aquesta mateixa interfície formi part de l'enllaç WAN com a entrada, per exemple;

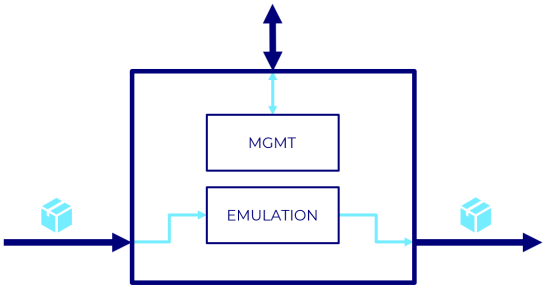
mentre que l'altra interfície és la sortida del trànsit. Això permet no haver d'establir taules d'encaminament, però s'haurà d'assignar una direcció IP al pont per poder accedir a la interfície de gestió.



Il·lustració 5: a) Gestió i emulació independents. b) Gestió i entrada d'emulació combinats

- *Triple arm* o tres interfícies

Aquesta és la topologia més adient. Amb aquesta topologia s'aïlla la interfície de gestió de les interfícies d'emulació. A més, permet escollir si les interfícies d'emulació es volen usar com a interfícies d'encaminament, amb la complexitat afegida de configuració que això comporta; o si les interfícies d'emulació es volen usar com a interfícies en mode pont, simplificant notablement el desplegament.



Il·lustració 6: Arquitectura de triple interfície

- Més de quatre interfícies

En ocasions, el desplegament es pot fer amb més de quatre interfícies de xarxa. Aquesta topologia consistirà en la combinació de varies de les topologies anteriors.

Això permetria executar de forma paral·lela més d'un procés d'emulació, de forma que, amb un únic emulador, es podran gestionar múltiples enllaços WAN.

4.3. Sistema operatiu i paquets utilitzats

Tot el codi que es presenta en aquest treball ha estat desenvolupat sobre una màquina virtual amb el sistema operatiu Ubuntu Server 18.04 LTS com a base, i testejat sobre Ubuntu Server 20.04 LTS. Aquesta versió del sistema operatiu és, en el moment de la presentació d'aquesta memòria, la darrera versió amb suport de llarg termini (*Long Time Support, LTS)* per part de Canonical o Ubuntu Foundation, els organismes que desenvolupen i mantenen aquesta distribució de Linux.

El codi que es desenvolupa i presenta, necessita executar ordres de terminal per tal de poder interactuar amb la gestió de les interfícies de xarxa del sistema operatiu i poder aplicar els canvis que es determinin des de la interfície d'usuari servida per web.

D'aquesta manera, veiem a continuació quins paquets es requereixen per garantir que el codi funcioni i en quina versió. És possible que el codi funcioni també en versions anteriors dels paquets que s'esmenten, però sempre es recomana utilitzar la mateixa versió sobre la que s'ha fet el desenvolupament o una posterior.

- **Apache/2.4.29:** cal tenir en compte que l'emulador WAN proporciona la seva interfície d'usuari a través d'una pàgina web. Això vol dir que és necessari instal·lar un servidor web, en aquest cas Apache. Es fa ús de característiques específiques d'Apache, de forma que el projecte no és compatible amb altres servidors web com Nginx.
- **PHP 7.2.24:** com veurem amb més detall més endavant en aquest document, el codi de la banda del servidor s'ha desenvolupat en PHP, d'aquesta forma, es requereix un intèrpret d'aquest llenguatge.
- **iproute2 4.15.0:** aquest paquet ve preinstal·lat en les darreres versions de Ubuntu Server. Proporciona la funcionalitat de gestió de paràmetres de xarxa com interfícies, taules d'encaminament i altres.
- **ethtool 1:4.15:** aquest paquet també ve preinstal·lat en les darreres versions de Ubuntu Server, i afegeix funcionalitats de gestió de les targetes de xarxa del dispositiu.
- **bridge-utils 1.5:** aquest paquet incorpora funcionalitat de control d'interfícies pont, permet crear-les, gestionar-les i esborrar-les. Com veurem, serà una ordre clau per aquest projecte.

Val a dir que no es fa ús de totes les funcionalitats que incorporen tots aquests paquets, però val la pena esmentar algunes utilitats que venen dins el paquet iproute2.

Com s'ha explicat, iproute2 és un paquet d'utilitats relacionat amb la gestió de les interfícies de xarxa, taules d'encaminament i d'altres, però una funcionalitat molt important per aquest projecte és el *traffic control* (tc), la funcionalitat que ens permetrà emular l'enllaç WAN.

La utilitat tc permet afegir i modificar característiques al comportament d'un dispositiu de xarxa (o interfície) determinat del nostre equip. Aquesta utilitat, aleshores, és la que ens permet afegir retard, pèrdua i reordenament de paquets, i altres paràmetres.

En definitiva, aquest és el llistat de paquets que cal tenir instal·lats en una màquina sobre la que es vulgui desplegar l'emulador d'enllaços WAN que s'ha desenvolupat. Tots són requisits indispensables, de forma que la manca d'un paquet o una utilitat de les que s'han esmentat, impediran que el sistema es pugui executar correctament.

5. Criteris de desenvolupament

5.1. Llenguatges de programació

Seguint amb el que es va especificar en la proposta del projecte, el llenguatge que s'ha escollit per al codi del servidor és PHP. Aquesta elecció no respon a un criteri tècnic, sinó a l'experiència prèvia i coneixement del propi llenguatge de programació, permetent, d'aquesta forma, centrar els esforços en el desenvolupament de la pròpia aplicació i no en el llenguatge de programació.

En la presentació del projecte es va indicar que es faria ús d'*scripts* en *bash* per a la interacció de l'aplicació amb el sistema operatiu. Finalment, això no ha estat necessari, ja que aquesta interacció s'ha pogut fer mitjançant PHP. D'aquesta forma, no s'inclou codi *bash* en el projecte.

Per al codi que s'executa a la banda del client, com que l'aplicació és una interfície web, s'ha utilitzat HTML, CSS i JavaScript:

- El primer, HTML, és el llenguatge elemental del desenvolupament web, un estàndard gestionat pel World Wide Web Consortium (W3C).
- El següent, CSS, també és un llenguatge mantingut pel W3C i permet donar estil als elements HTML.
- Finalment, JavaScript, permet afegir elements dinàmics als elements HTML i CSS.

5.2. Elecció d'entorns de treball

5.2.1. Servidor

Donada la simplicitat del codi *backend*, finalment s'ha optat per no fer ús de cap entorn de treball específic, ni de tampoc cap llibreria d'ajuda. Això permet limitar-se a l'ús de les eines pròpies del llenguatge de programació escollit, permetent simplificar de forma notable la comprensió del codi.

Aquesta decisió, a més, permet que el codi, en estar completament basat en les pròpies eines del llenguatge, és més escalable, ja que qualsevol persona amb unes mínimes nocions de PHP, pot entendre el codi i fer-hi aportacions.

5.2.2. Client

A diferència del codi de la banda del servidor, a la banda del client sí és fa ús de múltiples llibreries externes que simplifiquen notablement el desenvolupament i permeten afegir funcionalitats que, sense aquestes llibreries, no es podrien afegir.

La primera llibreria a esmentar és Bootstrap en la seva darrera versió en el moment de la redacció d'aquest treball: 4.4.1. Aquesta llibreria incorpora funcionalitats per a CSS i per a JavaScript. Concretament, proporciona un entorn que simplifica notablement el desenvolupament de la interfície gràfica. Tant el cos HTML com algunes interaccions de la interfície fan ús d'aquesta llibreria.

Una altra llibreria de què se'n fa ús per al desenvolupament del codi JavaScript és jQuery en la versió 3.4.1. Aquesta llibreria afegeix múltiples funcionalitats sobre el llenguatge JavaScript que permeten simplificar la interacció amb el codi HTML i el DOM, així com afegir esdeveniments. Es considera la llibreria JavaScript més utilitzada del món.

També s'inclou la llibreria Popper, que, igual que jQuery, és un requisit per al funcionament de Bootstrap. Aquesta llibreria es pot afegir al projecte de forma combinada amb Bootstrap, de forma que no cal descarregar-la i afegir-la de forma independent, com sí passa amb jQuery.

S'afegeix, també, la llibreria per a JavaScript Handlebars en la seva versió 4.7.6. Aquesta llibreria permet renderitzar i gestionar plantilles d'elements HTML, de forma que es poden recarregar i actualitzar seccions de la interfície d'usuari sense la necessitat de recarregar tota la pàgina.

Finalment, la darrera llibreria que s'afegeix és l'anomenada FontAwesome en la seva versió 5.13.0. Aquesta llibreria afegeix un conjunt d'icones que permeten representar de forma més visual les funcions dels diferents menús de la interfície d'usuari.

5.3. Publicació del codi font

Com s'ha esmentat ja diverses vegades en aquest treball, tot el codi font de l'aplicació serà d'accés públic. El codi es pot trobar al portal web GitHub, sota el projecte miquelfa/emuWAN:

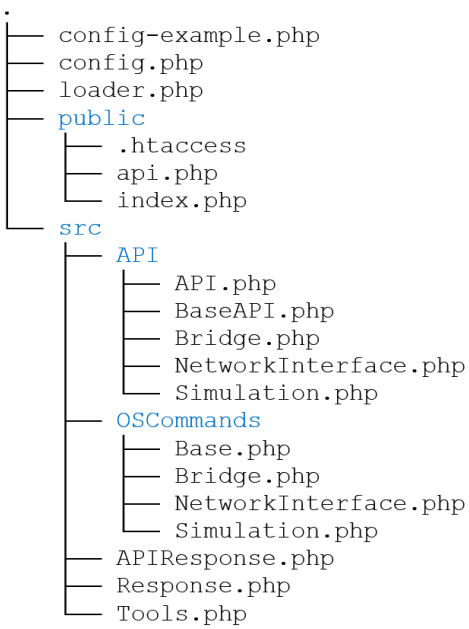
https://github.com/miquelfa/emuWAN

El repositori estarà en continua actualització, així com també el codi de la mateixa aplicació per corregir possibles problemes i afegir-hi noves funcionalitats.

6. Desenvolupament del *backend*

6.1. Estructura de fitxers

L'estructura de fitxers del backend del projecte és la següent:



Il·lustració 7: Estructura de fitxers backend

Com es pot observar en la imatge hi ha un màxim de tres nivells de fitxers. En el primer nivell hi ha tres fitxers i dues carpetes:

- El fitxer `loader.php` és un fitxer que carrega de forma dinàmica totes les classes necessàries perquè el codi funcioni correctament. Aquest fitxer s'ha d'incloure a l'inici de qualsevol execució del codi.
- Els fitxers `config-example.php` i `config.php` són fitxers de configuració. Com que la configuració depèn de cada instal·lació, el fitxer de configuració no es publica en el repositori, però sí es publica la configuració d'exemple perquè l'usuari pugui particularitzar-la segons els seus paràmetres.

Les carpetes que hi ha en aquest primer nivell són `src` i `public`. I s'expliquen amb més profunditat en el següents punts d'aquest capítol.

6.1.1. Fitxers públics

Aquesta carpeta inclou els fitxers que se serviran públicament a través del servidor web i el fitxer de configuració d'Apache2 `.htaccess`:

- El fitxer `index.php` és el punt d'entrada a través de qualsevol navegador web. Aquest fitxer senzillament serveix el cos principal de la interfície gràfica de gestió del dispositiu d'emulació. També valida que tots els paquets necessaris estiguin degudament instal·lats. Com que el *frontend* s'ha desenvolupat en format d'una única pàgina, com veurem més endavant, aquest fitxer no ha de fer cap altra acció.
- El fitxer `api.php` és el punt d'entrada de la API de l'aplicació. Aquest fitxer validarà quin controlador i quin mètode s'ha d'executar i, si els paràmetres són correctes, en llançarà l'execució.
- El fitxer `.htaccess` serveix per interpretar les direccions URL a què s'accedeix, de forma que l'aplicació té un comportament o un altre en funció de la ruta. D'aquesta forma, per tant, veiem el contingut d'aquest fitxer:

```
1  RewriteEngine on
2  RewriteBase /
3
4  # add trailing slash
5  RewriteCond %{REQUEST_FILENAME} !-f
6  RewriteRule .*[^/]$ /$0/ [L,R=301]
7
8  RewriteRule ^api/$                                                    api.php [L,QSA]
9  RewriteRule ^api/([networkinterface|simulation|bridge]*)/$            api.php?endpoint=$1 [L,QSA]
10 RewriteRule ^api/([networkinterface|simulation|bridge]*)/([a-z0-9]*)/$ api.php?endpoint=$1&id=$2 [L,QSA]
11 RewriteRule ^api/([networkinterface|simulation|bridge]*)/([a-z0-9]*)/([a-zA-Z]*)/$ api.php?endpoint=$1&action=$3&id=$2 [L,QSA]
```

Il·lustració 8: Fitxer .htaccess

Com es pot observar, les regles que s'inclouen en el fitxer permeten que les URL de l'API siguin més fàcils de llegir d'un cop de vista. Així doncs, el primer nivell de la URL és el controlador que s'executa, el segon és la interfície de xarxa sobre la qual s'opera, i el tercer l'acció que s'executa. És important tenir en compte que el mètode HTTP també determina l'acció que s'executarà.

D'aquesta forma, per exemple, una sol·licitud tipus GET es pot fer a la mateixa URL que una sol·licitud tipus POST, però l'acció executada serà diferent. Això ho veurem amb més detall en l'especificació de l'API.

6.1.2. Fitxers privats i codi font

L'altra carpeta que podem trobar en el primer nivell de fitxers és la carpeta `src`, que conté tot el codi que s'executa per realitzar les funcions requerides

en les peticions a l'API. Aquesta carpeta conté tres fitxers i dues carpetes, que s'expliquen a continuació:

- El fitxer `Tools.php` conté una classe amb mètodes estàtics. Aquests mètodes són eines complementàries que es poden requerir des de qualsevol altre punt del codi. Per exemple, un mètode que comprova si una direcció en el format CIDR és vàlida o no.
- El fitxer `Response.php` és un fitxer que gestiona les respostes que s'envien al client. Estableix paràmetres com ara la capçalera HTTP *Content*-Type en funció del tipus de resposta que s'està enviant.
- El fitxer `APIResponse.php` és una extensió del fitxer `Response.php` que permet una gestió més específica dels errors que es poden llançar a través de l'API. Així mateix, també determina un *Content-Type* fix com a *text/plain*.

Les carpetes que trobem en aquest nivell són `OSCommands` i `API`. La primera inclou tota la interacció entre l'aplicació i el sistema operatiu, i la segona inclou els controladors de les accions de l'API REST que s'inclou. Ambdues carpetes s'expliquen amb més detall a continuació.

6.2. Controladors de l'API

Com s'ha explicat, la carpeta `src/API` conté tots els controladors de l'API, és a dir, les classes a on es defineixen quines accions s'han d'executar quan es realitza una crida. Dins aquesta carpeta podem trobar un fitxer per a cadascuna de les entitats que es pot editar:

- `NetworkInterface.php` permet interactuar amb les interfícies de xarxa: obtenir informació com ara l'adreça IP o MAC, o si està connectada o desconnectada. A més, permet connectar-la, desconnectar-la o, fins i tot, assignar-hi una IP.
- `Simulation.php`, donada una interfície, permet interactuar amb l'emulació corresponent, afegint paràmetres o aturant-la.
- `Bridge.php` permet crear, llegir o destruir interfícies pont.

En aquesta carpeta també hi trobem els fitxers `API.php` i `BaseAPI.php`. El primer és el que fa la validació de la cridada i l'executa en cas de ser correcte. A més, també gestiona la resposta i els errors. El segon fitxer, és una base per a totes les altres classes de la carpeta; afegeix funcionalitats comunes.

Cadascun d'aquests controladors de l'API conté una sèrie de mètodes públics que corresponen a les diferents cridades i accions possibles. La nomenclatura d'aquests mètodes o funcions ha de ser molt concreta i ha de seguir el següent esquema:

```
mètodeHTTP_acció()
```

A on el `mètodeHTTP` es correspon als mètodes que estableix l'estàndard HTTP com puguin esser: GET, POST, DELETE, etcètera; i la `_acció` es correspon amb el nom de l'acció del tercer paràmetre de la URL. Aquest segon paràmetre pot esser opcional, donant lloc d'aquesta manera a la possibilitat d'incloure funcions que responen només al mètode HTTP.

Així per exemple, a la classe `Simulation`, podem trobar els mètodes: `get()`, `post()` o `post_reset()`.

6.3. Interacció amb el sistema operatiu

Aquesta és, possiblement, la clau del projecte. Les classes presents en la carpeta `src/OSCommands` són les encarregades de fer la interacció entre l'aplicació web i el sistema operatiu.

En aquesta carpeta, trobem una classe `Base.php` i una classe per a cadascuna de les entitats involucrades en aquest projecte: `NetworkInterface.php`, `Simulation.php` i `Bridge.php`. Totes les classes corresponents a les entitats són extensions de la classe `Base.php`, que conté elements comuns.

6.3.1. `Base.php`

Totes les classes de la carpeta `src/OSCommands` són extensions de la classe `Base`. Aquesta classe inclou principalment el mètode que llança l'execució d'ordres cap al sistema operatiu. Aquest mètode es diu `execute`, i és aquest:

```php
/**
 * Executes an OS command
 * @param string    $binary     The program to be executed
 * @param string    $arguments  The arguments to be passed
 * @param bool      $sudo       Whether execution must be in superuser
 * @return string               The full output
 */
public function execute($binary, $arguments, $sudo = false)
{
    if (!in_array($binary, $this->getAllBinaries())) {
        throw new \Exception("Invalid binary");
    }

    $command = $sudo ? 'sudo ' : '';
    $command .= $binary . ' ' . $arguments;
    return shell_exec($command);
}
```

Il·lustració 9: Mètode Base -> execute

Com es pot comprovar, aquest mètode rep tres arguments: el binari que s'ha d'executar, és a dir, la utilitat del paquet del sistema operatiu que es llançarà; els arguments d'aquesta cridada, i, si és necessària, la elevació de permisos o no.

La funció, un cop ha comprovat que el binari sigui vàlid, construeix l'ordre, i l'executa amb la funció `shell_exec` i retorna el resultat d'aquesta. La funció `shell_exec` és pròpia del llenguatge PHP, i executa l'ordre mitjançant l'intèrpret d'ordres i en retorna la sortida com una cadena de text.

6.3.2. `NetworkInterface.php`

Aquesta classe fa la interacció entre l'aplicació i el gestor d'interfícies de xarxa. Permet establir la direcció IP d'una interfície, connectar-la, desconnectar-la i llegir informació sobre l'estat de la mateixa.

Una de les característiques que s'ha de tenir en compte per aquesta classe, i que també es dona en les altres classes d'aquesta carpeta, és l'ús d'expressions regulars per llegir informació referent a les interfícies.

La lectura de l'estat de les interfícies es fa a través de l'ordre que es mostra a continuació, amb la seva sortida:

```
emuwan@emuwan:~$ ip address show enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:17:4b:b0 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.240/24 brd 192.168.0.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe17:4bb0/64 scope link
       valid_lft forever preferred_lft forever
```

Il·lustració 10: Obtenció d'informació d'una interfície de xarxa

D'aquesta forma, es fa necessari l'ús d'expressions regulars per tal de poder extreure la informació que ens interessa d'aquesta sortida. Així doncs, per obtenir l'adreça MAC d'una interfície, es fa ús de la següent fórmula:

```php
if (preg_match('/link\/ether\ (([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2}))/', $this->getCommandShow(), $mac)) {
    $this->MAC = $mac[1];
}
```

Il·lustració 11: Obtenció de l'adreça MAC en PHP

Es pot veure com dins la comanda retornada per `$this->getMAC()` es busca una expressió regular que comenci per `link/ether` i, a continuació, tingui 5 blocs composats per dos caràcters entre 0 i 9 i A i F en majúscules i minúscules, seguit del caràcter `:`. Finalment, hi ha de haver un bloc igual que els anteriors però que no acabi amb el caràcter `:`.

6.3.3. `Simulation.php`

Aquesta classe és la que fa la interacció entre el sistema operatiu i l'aplicació per allò referent a l'emulació d'una determinada interfície de xarxa. Les accions que executa són: la lectura d'una emulació existent, el restabliment de la mateixa i l'establiment d'una nova emulació.

La lectura dels paràmetres es fa d'una forma molt similar al que s'ha explicat en la classe `NetworkInterface`, però d'aquesta classe, val la pena destacar la creació i execució d'una nova emulació, que es fa amb les següents funcions:

```php
private function buildArguments()
{
    $args = sprintf("qdisc add dev %s root netem", $this->interface);

    if (!is_null($this->delay)) {
        $args .= sprintf(" delay %dms", $this->delay);
        if (!is_null($this->delayVariation)) {
            $args .= sprintf(" %dms distribution normal", $this->delayVariation);
        }
    }
    if (!is_null($this->reorder)) {
        if (!is_null($this->delay)) {
            $args .= sprintf(" reorder %d%%", $this->reorder);
        } else {
            $args .= sprintf(" delay 10ms reorder %d%%", $this->reorder);
        }
        if (!is_null($this->reorderCorrelation)) {
            $args .= sprintf(" %d%%", $this->reorderCorrelation);
        }
    }
    if (!is_null($this->loss)) {
        $args .= sprintf(" loss %d%%", $this->loss);
        if (!is_null($this->lossCorrelation)) {
            $args .= sprintf(" %d%%", $this->lossCorrelation);
        }
    }

    return $args;
}

private function run()
{
    try {
        $args = $this->buildArguments();
        $this->_reset();
        $this->execute(Base::TC, $args, true);
        return true;
    } catch (\Exception $e) {
        return false;
    }
}
```

Il·lustració 12: Codi de creació i execució de l'emulació

En primer lloc, analitzem la funció `run`, aquesta funció, primerament, fa una cridada a la funció `buildArguments()` que retorna una cadena de text amb els arguments necessaris, per a executar l'emulació. A continuació, fa una cridada a la funció `_reset()` que restableix l'emulació que hi pugui haver executant-se. Finalment, crida a la funció `execute()` que hem vist en l'apartat que fa referència al fitxer `Base.php`.

Per altra banda, la funció `buildArguments()` és la que prepara els arguments per executar l'emulació i els retorna com a cadena de text. Veiem que:

1. Si s'ha definit el retard, s'afegeix a l'ordre.
    a. Si s'ha definit una variació de retard, s'afegeix a l'ordre.
2. Si s'ha definit el reordenament s'afegeix a la comanda.

a. Si s'ha definit un retard, s'estableix aquest retard com a retard de reordenament; contràriament, el retard de reordenament serà de 10 mil·lisegons per defecte.
b. Si s'ha definit una correlació, aquesta s'afegeix a la comanda.

3. Si s'ha definit la pèrdua de paquets, s'afegeix a la comanda.
    a. Si s'ha definit una correlació en la pèrdua de paquets, aquesta s'afegeix a la comanda.

6.3.4. `Bridge.php`

La classe `Bridge`, és probablement la més senzilla de les tres. Aquesta classe crea, llegeix i destrueix les interfícies tipus pont. Realitza totes aquestes funcions fent ús de les fórmules explicades anteriorment.

6.4. Flux de funcionament

Dels punts anteriors d'aquest mateix capítol podem veure que el *backend* de l'aplicació presenta diverses capes independents que ofereixen una gran modularitat al codi. Aquesta divisió del codi pot dificultar-ne, fins a cert punt, la comprensió.

Per tal d'aclarir possibles dubtes sobre l'estructura del *backend* de l'aplicació, a continuació es mostra un exemple del flux de funcionament del codi quan es realitza una determinada acció. En aquest cas, l'acció que es demostra és l'enviament d'una comanda per desconnectar o connectar una interfície de xarxa.

A continuació es mostra un esquema amb l'ordre de cridades de les classes i els mètodes involucrats en l'acció:

Il·lustració 13: Flux de funcionament backend

En l'esquema anterior veiem que l'ordre que s'envia a l'API, encapsulada dins un paquet HTTP, és una acció `POST` a l'endpoint `status` dins de l'entitat `interface`.

Un cop rebut el paquet HTTP, el fitxer `api.php` interpreta aquest paquet, i determina quin controlador i quin mètode s'ha d'executar. En aquest cas, com

que el paquet es dirigeix a `networkinterface/{}/status/` i amb el mètode HTTP `POST`, el fitxer `api.php` determinarà que s'ha d'executar el controlador API `NetworkInterface.php` i el mètode `post_status()`.

Dins aquest mètode, a part de gestionar la recepció dels paràmetres i la resposta, es farà la crida a l'ordre del sistema operatiu corresponent per realitzar l'acció desitjada. Concretament es fa una crida a la classe `NetworkInterface.php` de l'espai de noms corresponent a la gestió de comandes del sistema operatiu, és a dir `OSCommands`.

Concretament es crida al mètode estàtic `interfaceStatus()` que fa la corresponent crida a l'ordre del sistema operatiu: `sudo link set dev` **`interfície up/down`**.

Al punt segon de l'ANNEX III – Diagrames de flux s'inclou un conjunt de diagrames de flux molt detallats amb el funcionament del *backend*. Tots aquests diagrames segueixen l'explicació anterior amb major precisió. Val a dir que s'han omès alguns detalls que no són rellevants per a la comprensió dels fluxos dels diferents processos executats a l'API de l'emulador.

6.5. Elevació de permisos

Com és natural en sistemes Linux, no totes les ordres poden esser executades per tots els usuaris. D'aquesta forma, es fa necessària l'elevació de permisos per poder executar determinades comandes.

Aquesta elevació de permisos es pot aconseguir de moltes formes, com per exemple afegir a l'usuari del servidor web al grup de superusuaris, de forma que podria executar lliurement qualsevol ordre. Paradoxalment, però, aquesta idea pot representar un risc de seguretat.

La fórmula que s'ha escollit per a l'elevació de permisos en el desenvolupament d'aquesta aplicació és una mica més conservadora, però també pot representar certs riscos de seguretat. Val a dir, però, que la seguretat no és un element que s'hagi tingut en especial consideració en aquest treball, donat que l'aplicació no està pensada per ser executada en entorns productius.

D'aquesta forma, s'ha optat per afegir permisos específics perquèl'usuari del servidor web, qui fa les operacions d'interacció entre l'aplicació i el sistema operatiu, pugui fer cridades amb elevació de permisos només a les comandes que siguin necessàries per al funcionament de l'aplicació.

Aquesta elevació de permisos es pot definir en el conegut fitxer `sudoers` de la distribució Linux pertinent. Per tant, el fitxer `sudoers` resultant en un sistema que tingui aquest emulador instal·lat, tindrà una forma similar al següent:

```
#
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults        env_reset
Defaults        mail_badpass
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin"

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "#include" directives:

#includedir /etc/sudoers.d

#emuWAN lines
www-data  ALL=NOPASSWD:  /sbin/ethtool
www-data  ALL=NOPASSWD:  /bin/ip
www-data  ALL=NOPASSWD:  /sbin/tc
www-data  ALL=NOPASSWD:  /sbin/brctl
```

Il·lustració 14: Fitxer sudoers de l'emulador WAN

Es pot veure que el darrer bloc de línies proporciona elevació de permisos sense contrasenya a l'usuari `www-data`, representant d'Apache2 al sistema operatiu, per a les ordres: `ip`, `tc`, `brctl` i `ethtool`.

6.6. L'estructura modular

Amb la finalitat d'aconseguir l'objectiu de proporcionar una aplicació modular i amb un codi fàcilment escalable, s'ha optat per l'estructura que s'ha explicat en els punts anteriors d'aquest capítol.

Així doncs, aquesta estructura permet afegir noves funcionalitats d'una forma molt senzilla. Per a cada mòdul nou que s'afegeixi s'haurien de crear dos fitxers: un per a la interacció entre el sistema operatiu i l'aplicació, dins la carpeta `src/OSCommands`; i un altre com a controlador API dins la carpeta `src/API`.

A més, s'hauria de corregir el fitxer `.htaccess`, per permetre un nou punt de finalització de la interfície API que permeti fer la corresponent crida al controlador.

6.7. Referència de l'API REST

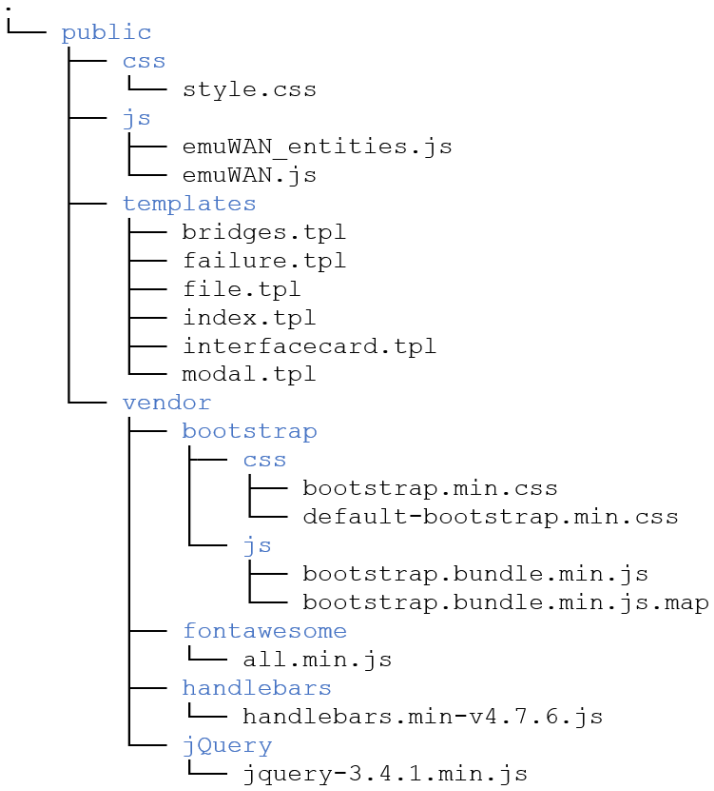
S'inclou una referència a totes les possibles cridades a l'API REST desenvolupada. Aquesta referència completa es pot trobar a l'Annex 1 d'aquest mateix document.

Cadascuna de les possibles cridades es presenta en una taula a on es pot visualitzar la URL de destí; els paràmetres que accepta la cridada, si és el cas; el mètode HTTP que s'ha de fer servir; el cost de la cridada, i el cos de la resposta.

7. Desenvolupament del *frontend*

7.1. Estructura de fitxers

El *frontend* de l'aplicació també té una estructura pensada per tal de poder continuar el desenvolupament d'una forma modular:



Il·lustració 15: Estructura de fitxers frontend

Com es pot observar, l'estructura de fitxers és lleugerament més complexa que en el cas del *backend*. Això és a causa que, en aquest cas, s'han d'encabir les llibreries externes que s'utilitzen.

Així aleshores, podem trobar, sota la carpeta `public`, que conté tots els fitxers accessibles públicament a través del servidor web, una carpeta anomenada `vendor` a on hi trobem els fitxers de les diferents llibreries externes que s'utilitzen.

A aquesta carpeta `vendor` podem trobar una carpeta per a cada llibreria. Així doncs, veiem les carpetes `bootstrap`, `FontAwesome`, `Handlebars` i `jQuery`, amb els seus respectius fitxers. El criteri de selecció d'aquestes llibreries s'ha explicat anteriorment en aquest document.

A més, podem observar tres carpetes més:

- `css`: aquesta carpeta conté els fulls d'estil CSS específics i propis de l'aplicació. Amb l'ús de la llibreria bootstrap, que simplifica molt el treball d'estilització, només cal un sol fitxer molt senzill.
- `js`: aquesta carpeta conté tots els fitxers de JavaScript. Aquests són els fitxers que aporten dinamisme al contingut, i permeten el format de navegació amb una sola pàgina, com veurem més endavant.
- `templates`: aquesta carpeta conté les plantilles que permeten renderitzar els diferents components de la pàgina i mostrar-los a l'usuari.

7.2. Càrrega de fitxers requerits

Com s'explica en el capítol anterior, quan es fa la primera sol·licitud al servidor web, aquest serveix, a través del fitxer `index.php`, la plantilla `index.html`. Aquesta plantilla tan sols conté el cos HTML principal de la pàgina, amb els contenidors corresponents a on es renderitzaran els diferents elements de la interfície gràfica.

La plantilla d'inici també fa la càrrega dels fitxers necessaris perquè l'aplicació pugui funcionar. Els fitxers que es carreguen són els fulls d'estil i les llibreries JavaScript externes. Aquesta càrrega es fa en el bloc de capçalera com es pot veure a continuació:

```
1  <!DOCTYPE html>
2  <html lang="en">
3      <head>
4          <title>emuWAN</title>
5          <meta charset="utf-8">
6          <meta name="viewport" content="width=device-width, initial-scale=1">
7          <link rel="stylesheet" href="/vendor/bootstrap/css/bootstrap.min.css">
8
9          <link rel="stylesheet" href="/css/style.css">
10
11         <script src="/vendor/jQuery/jquery-3.4.1.min.js"></script>
12         <script src="/vendor/bootstrap/js/bootstrap.bundle.min.js"></script>
13         <script src="/vendor/handlebars/handlebars.min-v4.7.6.js"></script>
14         <script defer src="/vendor/fontawesome/all.min.js"></script>
15     </head>
16     <body>
```

Il·lustració 16: Capçalera fitxer HTML base

Al final d'aquesta plantilla `index.html` podem trobar les següents dues línies:

```
76      </body>
77
78      <script src="/js/emuWAN_entities.js"></script>
79      <script src="/js/emuWAN.js"></script>
80  </html>
```

Il·lustració 17: Càrrega dels scripts principals

Aquests dos scripts són els que contenen tota la lògica per al funcionament de la pàgina sense navegació. D'aquesta forma, quan s'accedeix a la interfície web de l'emulador i se serveix aquest fitxer `index.html`, el navegador que el rebi haurà de fer la càrrega de tots aquests scripts que hem vist. Podem veure aquest procés de càrrega en la següent imatge:

Name	Status	Type	Initiator	Size	Time	Waterfall
emuwan.miquelfa.com	200	document	Other	1.2 kB	8 ms	
bootstrap.min.css	200	stylesheet	(index)	22.5 kB	23 ms	
style.css	200	stylesheet	(index)	1.0 kB	5 ms	
jquery-3.4.1.min.js	200	script	(index)	31.0 kB	18 ms	
bootstrap.bundle.min.js	200	script	(index)	23.1 kB	19 ms	
handlebars.min-v4.7.6.js	200	script	(index)	24.5 kB	14 ms	
emuWAN_entities.js	200	script	(index)	1.5 kB	15 ms	
emuWAN.js	200	script	(index)	3.7 kB	18 ms	
all.min.js	200	script	(index)	433 kB	99 ms	

Il·lustració 18: Càrrega inicial de fitxers

7.3. Lògica aplicada al *frontend* i càrrega de components

Com s'ha esmentat nombroses vegades, la interfície web que s'ha plantejat no té cap tipus de navegació, és a dir, el servidor web públic tan sols serveix el primer fitxer a donar una estructura bàsica de l'aplicació.

Un cop el navegador processa aquest fitxer i carrega totes les dependències que aquest presenta, es delega tota la gestió lògica als dos scripts que es carreguen en les darreres línies. Aquests scripts interactuen amb el servidor web i l'emulador a través de l'API REST. D'aquesta forma, l'aplicació desenvolupada no mostra navegació per avançar o anar enrere.

Veiem, primer, amb més detall quin és el contingut d'aquests dos fitxers:

7.3.1. `emuWAN_entities.js`

Aquest fitxer conté una col·lecció de classes que es correspon amb les mateixes entitats que trobem en l'API REST que s'ha explicat en el capítol

anterior. Aquestes classes són: `NetworkInterface`, `Simulation`, `Bridge` i, finalment, una classe d'ajuda, que s'anomena `AjaxWrapper`.

Aquestes classes són les que fan d'intermediari entre el *frontend* i el *backend*, es corresponen amb les entitats que es mostren en la interfície gràfica i amb què treballa també el backend.

Cadascuna d'aquestes classes corresponents a les diferents entitats conté un nombre de mètodes que fan les crides als diferents *endpoints* de l'API. Aquestes crides es fan a través de la tècnica AJAX, que permet fer sol·licituds al servidor de forma asíncrona i transparents per a l'usuari, eliminant d'aquesta forma la navegació.

7.3.2. `emuWAN.js`

Aquest és el fitxer que conté tota la lògica de navegació i interacció de l'aplicació. En aquest fitxer podem trobar les classes que es corresponen als diferents mòduls en els que s'ha dividit tota la interfície gràfica:

- **`NetworkInterface_Module`**: aquesta classe conté tot el mòdul que gestiona les interfícies de xarxa. Les accions de renderització d'interfícies, edició, connexió i desconnexió i gestió d'emulacions es gestionen a través d'aquest mòdul.
- **`Templates_Module`**: aquesta classe conté el mòdul que gestiona la càrrega dinàmica de plantilles. Com hem vist, tots els components de la interfície estan dispersos en múltiples plantilles. Aquestes plantilles es carreguen de forma dinàmica a mesura que es van requerint per renderitzar els distints components. Aquest mòdul gestiona la càrrega de plantilles i assegura que una mateixa plantilla no se sol·liciti més d'una vegada.
- **`Bridges_Module`**: aquest mòdul gestiona el component corresponent a la gestió de les interfícies pont. En renderitza el seu contingut i gestiona l'addició i eliminació de les interfícies pont.
- **`Modal_Module`**: aquest mòdul gestiona el funcionament del component modal que es mostra sobreposant-se a tot el contingut de la pàgina, per exemple, quan s'obri algun formulari o algun quadre d'advertència.

Aquest fitxer, a més, gestiona també la inicialització de l'aplicació. Aquesta inicialització es llança al final de tot del fitxer, i un cop el document està totalment carregat. Aquesta inicialització es fa a través de la següent fórmula:

```
442  $(function(){
443      emuWAN.startApp();
444  });
```

Il·lustració 19: Inicialització de la lògica del frontend

Aquesta funció gestiona tota la càrrega dels components necessaris per poder iniciar l'aplicació. Aquests components són, bàsicament, la càrrega de totes les interfícies i les seves emulacions, la càrrega de les interfícies pont, i la càrrega de les plantilles bàsiques. D'aquesta forma, quan aquesta funció és cridada, es produeix una càrrega asíncrona i en paral·lel de tots aquests components:

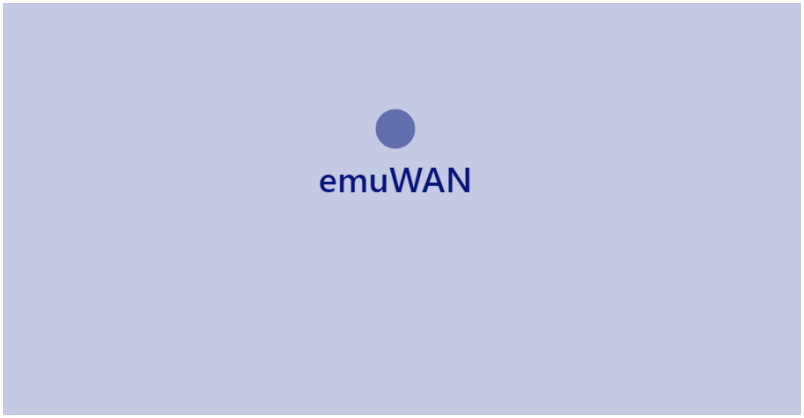
	Name	Status	Type	Initiator	Size	Time	Waterfall
1.	emuwan.miquelfa.com	200	document	Other	1.3 kB	9 ms	
2.	bootstrap.min.css	200	stylesheet	(index)	22.5 kB	19 ms	
	style.css	200	stylesheet	(index)	1.0 kB	7 ms	
	jquery-3.4.1.min.js	200	script	(index)	31.0 kB	21 ms	
	bootstrap.bundle.min.js	200	script	(index)	23.1 kB	12 ms	
	handlebars.min-v4.7.6.js	200	script	(index)	24.5 kB	17 ms	
	emuWAN_entities.js	200	script	(index)	1.5 kB	14 ms	
	emuWAN.js	200	script	(index)	3.7 kB	14 ms	
	all.min.js	200	script	(index)	433 kB	94 ms	
3.	interfacecard.tpl	200	fetch	emuWAN.js:249	5.4 kB	2 ms	
	modal.tpl	200	fetch	emuWAN.js:249	6.8 kB	3 ms	
4.	networkinterface/	200	xhr	jquery-3.4.1.min.js:2	476 B	30 ms	
5.	enp0s3/	200	xhr	jquery-3.4.1.min.js:2	371 B	22 ms	
	enp0s9/	200	xhr	jquery-3.4.1.min.js:2	371 B	19 ms	
	enp0s10/	200	xhr	jquery-3.4.1.min.js:2	371 B	17 ms	
6.	bridge/	200	xhr	jquery-3.4.1.min.js:2	330 B	10 ms	

Il·lustració 20: Càrrega de dades de la interfície de l'emulador

En la il·lustració anterior, podem veure el següent, per ordre cronològic:

1. Se sol·licita el fitxer `index.html`
2. Aquest fitxer provoca la càrrega de tots els components bàsics: scripts i llibreries externes.
3. A continuació, la càrrega de l'script `emuWAN.js`, provoca la càrrega de dues plantilles.
4. També provoca la sol·licitud d'un llistat de totes les interfícies de xarxa detectades per l'emulador.
5. La càrrega del llistat d'interfícies dispara, en aquest moment, la càrrega de tots els paràmetres d'emulació de cadascuna de les interfícies.
6. Per acabar, es carrega el llistat d'interfícies pont.

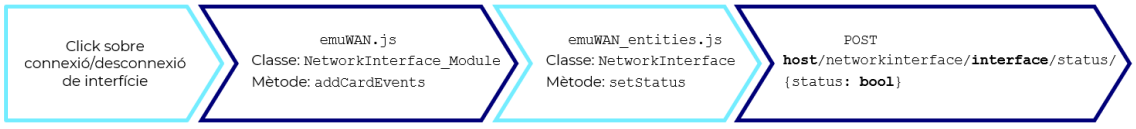
Durant tot aquest procés de càrrega, la pàgina es mostra en un estat de càrrega, perquè l'usuari percebi que s'estan carregant els diferents components necessaris per al funcionament de l'aplicació:



Il·lustració 21: Pàgina en estat de càrrega de components

7.3.3. Flux de funcionament

Un cop explicats tots els components del *frontend*, convé resumir el funcionament del codi de la banda del client. A continuació es mostra el flux d'acció i cridades al codi d'una acció d'exemple, en aquest cas, connectar o desconnectar una interfície de xarxa:



Il·lustració 22: Flux de funcionament frontend

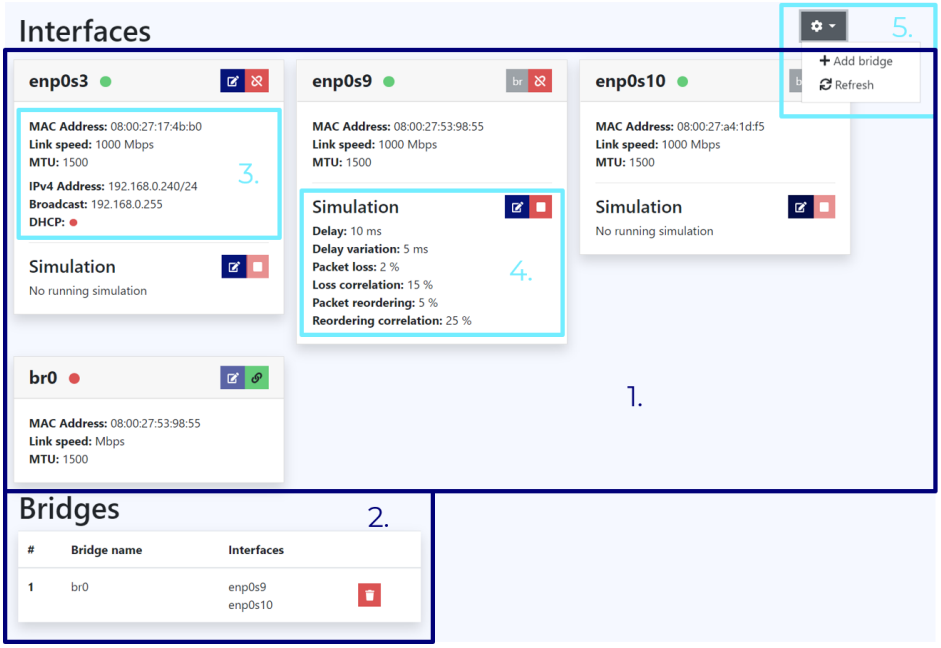
Com es pot comprovar, el flux es composa de 4 passes: a la primera l'usuari executa l'acció seleccionada, ja sigui enviar un formulari o fer *click* a una opció. A continuació, l'esdeveniment que s'ha llançat a partir de l'acció de l'usuari és captat per el mòdul afectat definit al fitxer `emuWAN.js`; en aquest cas, el mòdul `NetworkInterface_Module` i el mètode `addCardEvents`.

Aquest esdeveniment és tractat pel mòdul, que, a continuació, interactua amb l'entitat corresponent al fitxer `emuWAN_entities.js`, en aquest cas `NetworkInterface` al mètode `setStatus`, que farà les modificacions pertinents. En el cas de l'exemple, aquestes modificacions suposen enviar una petició tipus `POST` cap al *backend* de l'aplicació amb la URL i els paràmetres indicats en la il·lustració.

Al punt primer de l'ANNEX III – Diagrames de flux s'inclou un conjunt de diagrames de flux que expliquen el funcionament del *frontend*, des de la interacció de l'usuari fins al llançament de la corresponent crida a l'API de l'emulador. També es detalla el llançament i captura d'esdeveniments JavaScript.

7.4. Interfície d'usuari

Un cop s'ha fet tota la càrrega explicada en el punt anterior, l'aplicació ja passa a estar en un estat en què l'usuari pot interactuar amb la mateixa i llegir la informació de l'estat de l'emulador, a continuació veiem un exemple de la interfície:



Il·lustració 23: Vista general de la interfície d'usuari

En la il·lustració superior hi podem veure els següents components:

1. Panell d'interfícies de xarxa: aquest panell conté un requadre per a cadascuna de les interfícies de xarxa detectades pel dispositiu d'emulació.
2. Panell d'interfícies pont: aquest panell conté una taula amb totes les interfícies pont i quines interfícies de xarxa interconnecta cadascuna d'elles. S'ha de notar que les interfícies pont també es visualitzen en el llistat d'interfícies de xarxa.

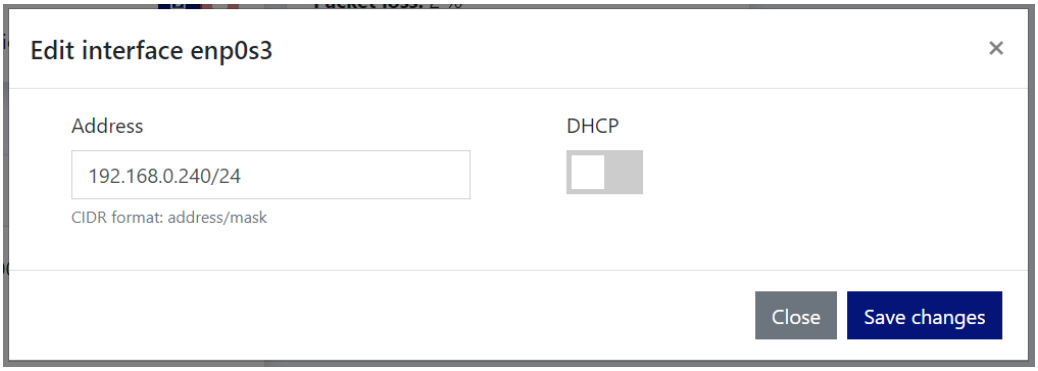
3. Informació sobre la interfície de xarxa: en aquest panell podem llegir la informació referent a una determinada interfície de xarxa.
4. Informació sobre l'emulació: en aquest panell podem visualitzar quins són els paràmetres de les emulacions que puguin estar executant-se.
5. Menú de gestió: en aquest menú podem trobar un botó que permet refrescar tota la interfície d'usuari i totes les dades que s'hi mostren, i un altre botó que permet afegir una nova interfície pont.

A continuació, veiem quines són les possibles accions que podem realitzar sobre els diferents components:

7.4.1. Edició d'interfície de xarxa

Cada interfície de xarxa, si no forma part d'una interfície de tipus pont té un menú d'edició. En aquest menú d'edició es permet establir la direcció IP de la interfície.

Si la interfície de xarxa pertany a una interfície pont; l'adreça IP establerta a la interfície serà ignorada. Per aquest motiu, quan la interfície de xarxa pertany a una interfície pont, l'edició està desactivada.



Il·lustració 24: Edició de la IP de la interfície de xarxa

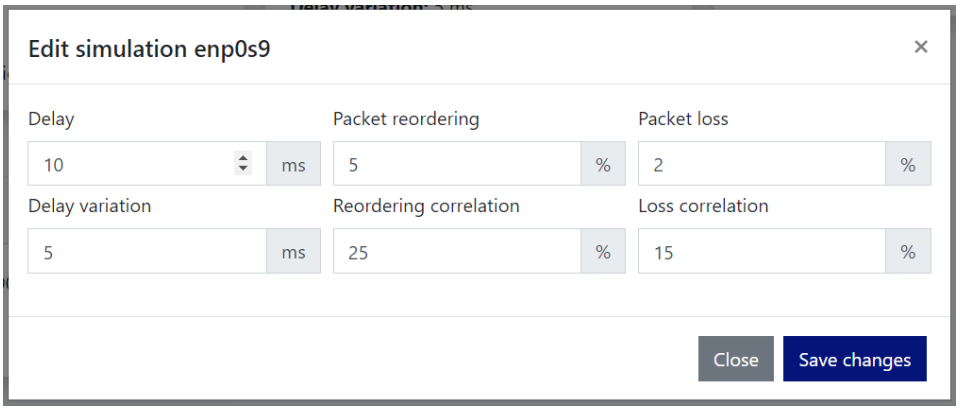
També s'afegeix l'opció de que la interfície rebi l'adreça a través de DHCP. En aquest cas, però, aquesta opció ha d'estar establerta en la configuració del gestor de xarxes del propi sistema operatiu. En el cas d'Ubuntu Server, a través del paquet `netplan`.

7.4.2. Connexió o desconnexió de la interfície de xarxa

Cadascuna de les interfícies de xarxa pot esser connectada i desconnectada amb un simple clic.

7.4.3. Edició de l'emulació

Evidentment, es pot gestionar l'emulació de cadascuna de les interfícies. Ja sigui per establir una nova emulació, o per a editar una emulació existent, el formulari sempre és el mateix:



Il·lustració 25: Edició de l'emulació

7.4.4. Restabliment de l'emulació

Com en el cas de la connexió o desconnexió d'interfícies de xarxa, l'emulació també pot esser restablerta amb un sol clic. Això provocarà que la interfície funcioni sense cap emulació.

7.4.5. Creació d'una interfície pont

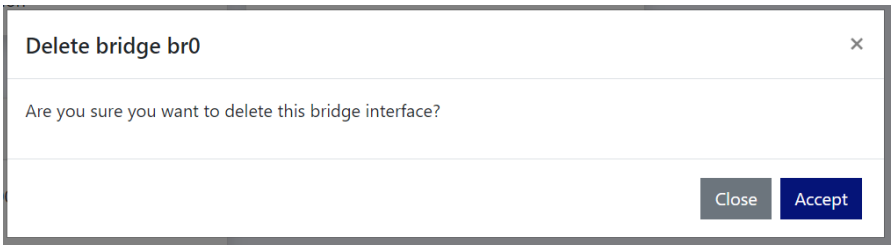
Tal i com hem vist en la captura de pantalla de la interfície, dins el menú de gestió trobem un botó que ens permet crear una nova interfície pont.

Per crear una interfície pont, tan sols s'ha d'indicar un nom i quines són les interfícies de xarxa que es desitgen incloure al pont:

Il·lustració 26: Creació d'interfície pont

7.4.6. Eliminació d'interfície pont

Igual que es poden crear interfícies pont, també es poden eliminar, rebent prèviament un missatge de confirmació:



Il·lustració 27: Eliminació d'interfície pont

7.4.7. Refresc de la interfície d'usuari

En el mateix menú de gestió trobem l'opció de refrescar la interfície. Això pot ser útil per aquells casos en què un altre usuari estigui fent canvis sobre l'emulació, cosa que provocaria desactualització en la informació presentada a la interfície.

8. Proves i resultats

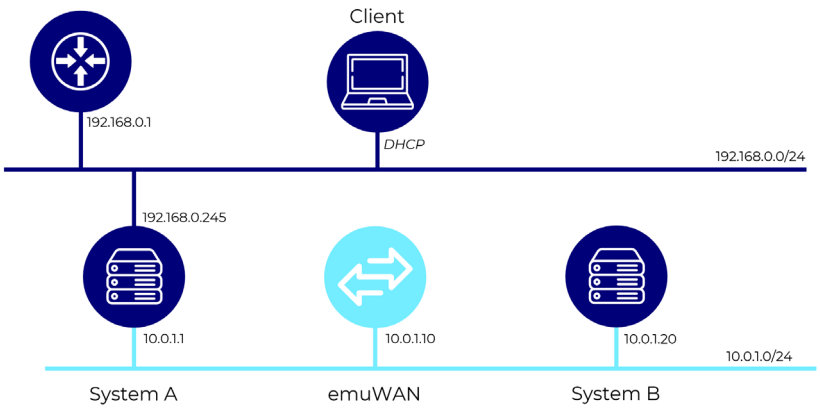
En aquest capítol veurem un conjunt de proves que ens permetran validar el correcte funcionament de l'emulador. Provarem les diferents funcionalitats que s'han explicat en aquest treball i farem emulacions d'enllaços WAN segons els paràmetres especificats.

El capítol es divideix segons les diferents arquitectures possibles en el desplegament de l'emulador d'enllaços WAN, d'aquesta forma, anirem repetint les proves amb les diferents arquitectures possibles.

8.1. Proves en arquitectura *single-arm*

Abans d'iniciar les proves en aquesta arquitectura, veiem quin és l'entorn del qual es disposa. Aquest entorn es compon de: un client des del qual gestionarem l'emulació, dos dispositius per emular els dos punts finals de la comunicació i el propi emulador.

D'aquesta forma, per tant, l'esquema que trobem en aquesta primera bateria de proves és el següent:



Il·lustració 28: Proves single-arm - Esquema de xarxa

En aquest entorn, podem comprovar que la màquina anomenada com a *System A* compta amb dues interfícies, això és així per poder accedir des del client al dispositiu d'emulació i visualitzar el sistema de gestió.

Evidentment, aquesta composició requereix de configuració addicional a totes les màquines per tal de poder compondre les rutes adequades perquè la comunicació entre el *System A* i el *System B* passi a través de la màquina d'emulació d'enllaços WAN, *emuWAN*. Per aconseguir aquest efecte afegim les següents rutes:

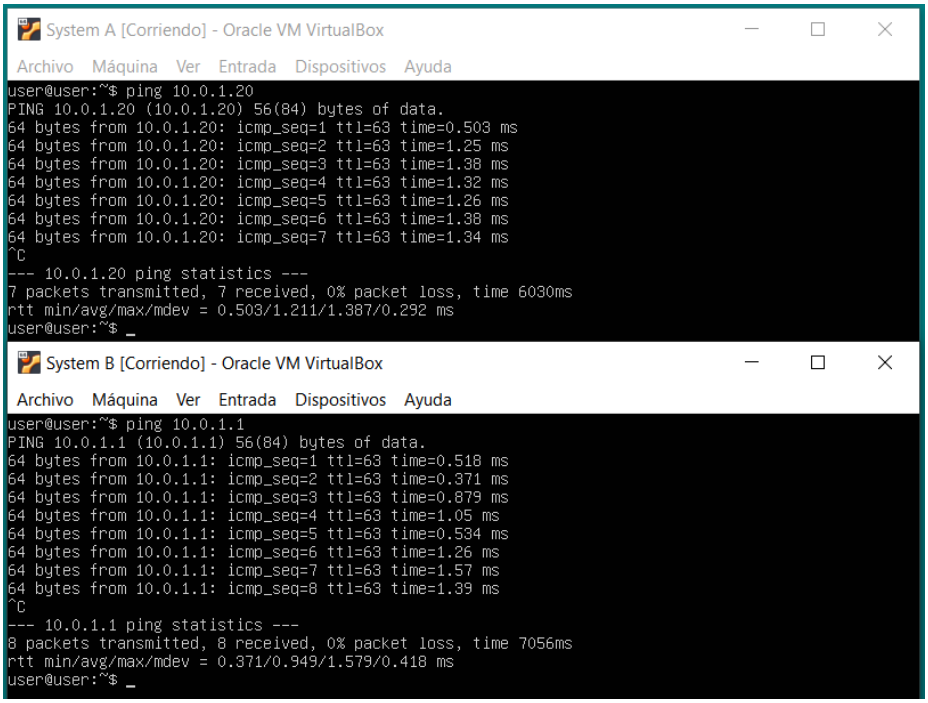
Al *System B*:

```
10.0.1.1/32 via 10.0.1.10
```

Al *System A*:

```
10.0.1.20/32 via 10.0.1.10
```

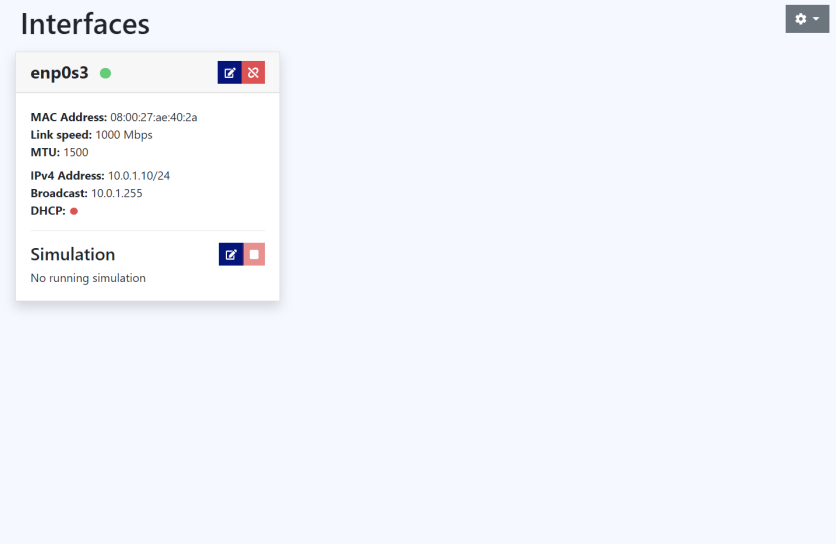
D'aquesta manera, podem comprovar que el dispositiu *emuWAN*, que té assignada l'adreça 10.0.1.10, es troba a mig camí entre els dos sistemes finals. Un cop establerta la configuració, podem comprovar que existeix connectivitat entre els dos sistemes a través d'un simple PING:



Il·lustració 29: Proves single-arm - Ping de comprovació de comunicació

Notem que el temps del PING és d'aproximadament 1 mil·lisegon. És important considerar que s'ha hagut de desactivar la redirecció ICMP en els dos sistemes que actuen dextrems de la connexió, i en el propi emulador WAN.

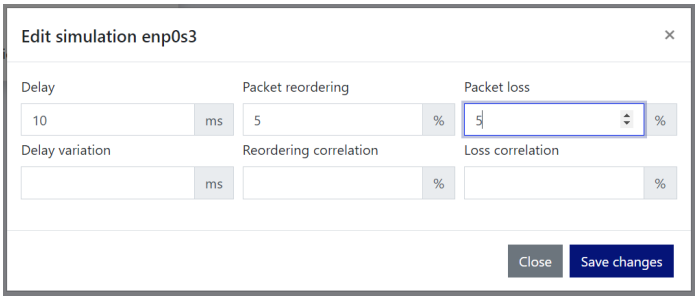
Arribats a aquest punt, un cop establerta la comunicació entre els dos sistemes, podem iniciar les proves de l'emulador d'enllaços WAN. Així doncs, primerament, veiem quin aspecte presenta la interfície gràfica de l'emulador WAN:



Il·lustració 30: Proves single-arm - Interfície gràfica

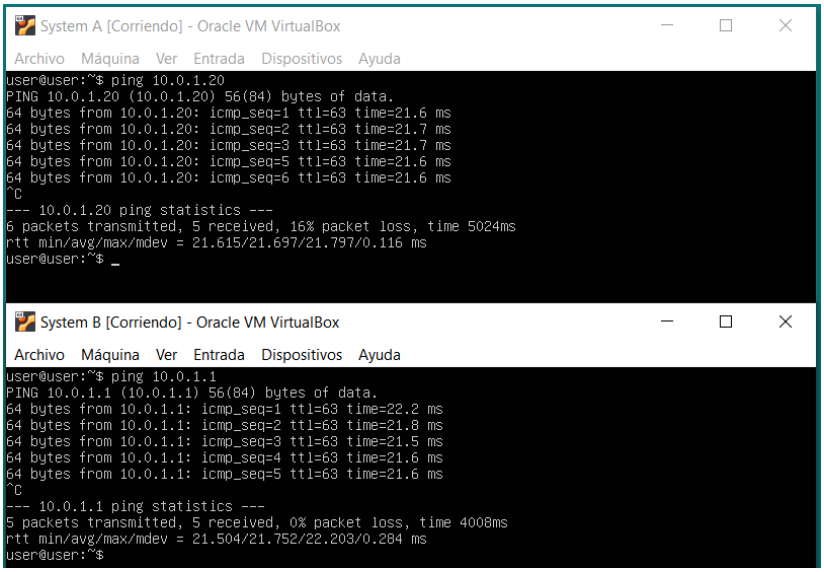
Com era previsible, l'únic component que ens mostra l'emulador és el que correspon a l'única interfície de xarxa present al dispositiu. Presenta l'adreça IP 10.0.1.10, tal i com hem definit manualment.

Així doncs, afegim a continuació els paràmetres d'una emulació per tal de comprovar l'efectivitat de l'emulador. Concretament afegirem: 10 mil·lisegons de retard, sense variació; un 5% de reordenament de paquets, sense correlació; i un 5% de pèrdua de paquets, sense correlació. Les proves amb correlació i variació les farem més endavant.



Il·lustració 31: Proves single-arm - Paràmetres d'emulació

Amb aquesta emulació establerta, tornem a llançar el PING entre els dos sistemes i obtenim els següents resultats:



Il·lustració 32: Proves single-arm - Prova de retard sobre ICMP

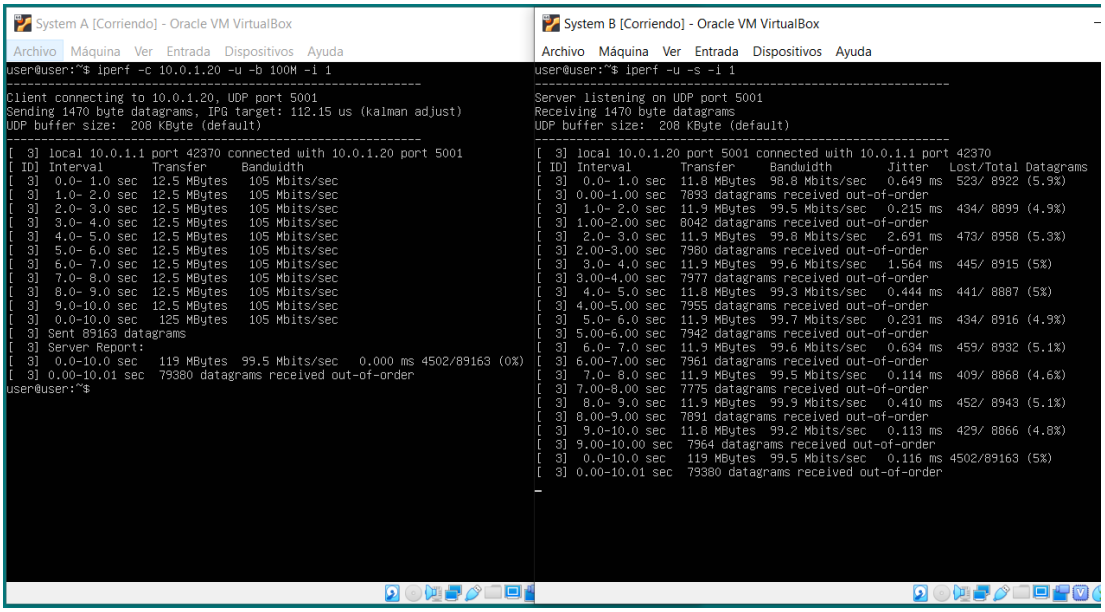
Com es pot observar, notem que la resposta del PING ara és de aproximadament 20 mil·lisegons, però hem establert l'emulació a 10 mil·lisegons.

Aquest fet es deu a que el retard s'aplica dos cops per cada paquet ICMP transmès. Hem de tenir en compte que els paràmetres d'emulació s'apliquen únicament als paquets que surten del dispositiu d'emulació. D'aquesta manera, la seqüència que observem és la següent:

1. *System A* envia el paquet ICMP a l'*emuWAN*.
2. *emuWAN* rep el paquet sense aplicar cap retard.
3. *emuWAN* envia el paquet aplicant 10 mil·lisegons de retard.
4. *System B* rep el paquet ICMP i respon el corresponent *echo response*.
5. *emuWAN* torna a rebre el paquet sense aplicar el retard.
6. *emuWAN* envia el paquet a *System A* aplicant 10 mil·lisegons de retard.
7. *System A* rep el paquet ICMP.

Com es pot observar, s'aplica un retard a l'anada i un altre retard a la tornada.

Arribats a aquest punt tan sols hem pogut comprovar l'efectivitat del retard, però no hem pogut observar la pèrdua i el reordenament de paquets. Per fer-ho, utilitzarem el paquet IPERF que crearà un flux de paquets UDP entre els dos extrems de la comunicació i analitzar-ne el rendiment.



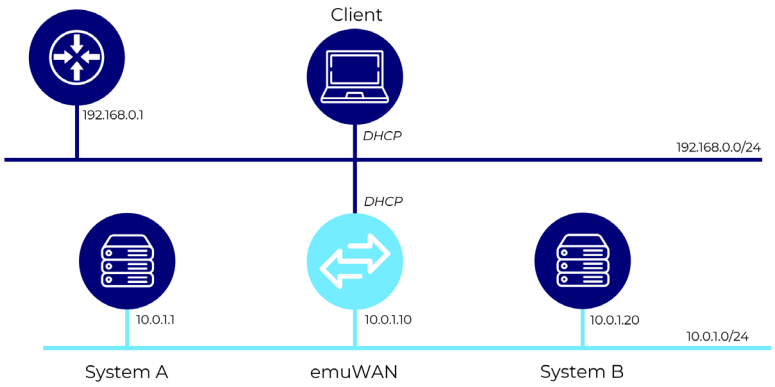
Il·lustració 33: Proves single-arm - Proves de reordenament i pèrdua de paquets

Com podem veure en les dues darreres línies del servidor, a l'interval entre els 0 i els 10 segons hem perdut 4502 dels 89163 paquets que s'han enviat, suposant un 5% de pèrdues. A més, podem observar que s'ha produït reordenament, tot i que no es dona una xifra percentual de reordenament ja que a partir del primer paquet fora d'ordre, tots són considerats desordenats.

8.2. Proves en arquitectura *double-arm* amb branca de gestió

Un cop comprovat el correcte funcionament de l'emulador en el format d'una sola branca, veiem a continuació les proves amb dues branques. Concretament, en aquest cas, establirem una interfície de xarxa per a gestió, i una altra interfície de xarxa per a l'emulació.

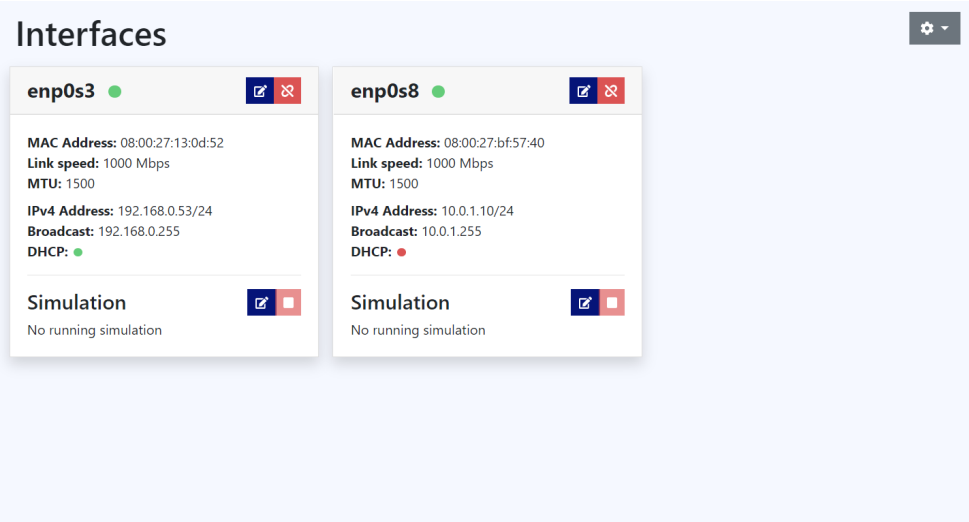
Això ens permet aïllar el trànsit de gestió del trànsit de l'emulació, i, d'aquesta manera, evitar que els paràmetres de l'emulació s'apliquin a la gestió del sistema. Per aquest desplegament, hem permès que la interfície de gestió obtingui la IP a través de DHCP de la xarxa local. L'esquema és el següent:



Il·lustració 34: Proves double-arm 1 - Esquema de xarxa

Amb aquest esquema, com es pot observar, haurem d'afegir les mateixes rutes que en l'apartat anterior en els dos equips *System A* i *System B*, perquè facin passar el trànsit a través de l'*emuWAN*.

Un cop establertes les rutes i comprovada la connectivitat entre els tres sistemes, la interfície gràfica que presenta l'emulador és la següent:

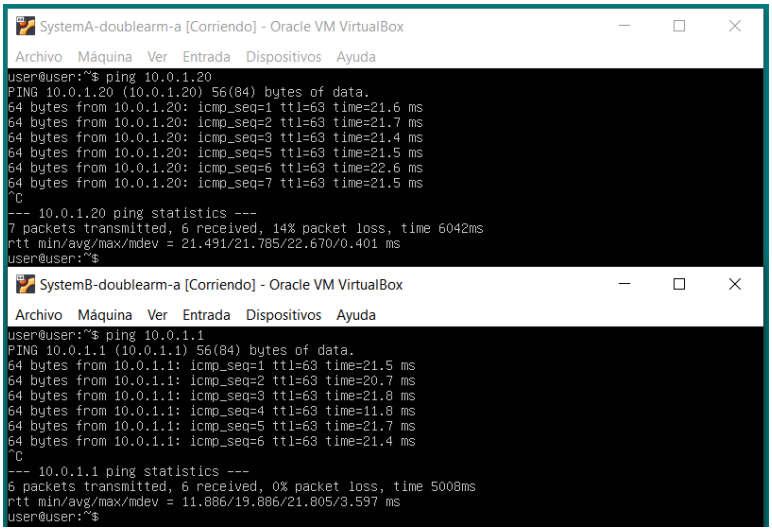


Il·lustració 35: Proves double-arm 1 - Interfície de l'emulador

Com s'observa, ja hi apareixen les dues interfícies de xarxa. La primera amb una adreça IP assignada per DHCP, i la segona amb l'adreça establerta manualment segons l'esquema anterior.

Amb aquesta configuració, per tant, establim la mateixa emulació que en el punt anterior, amb un retard de 10 mil·lisegons i amb una pèrdua i reordenament de paquets del 5%.

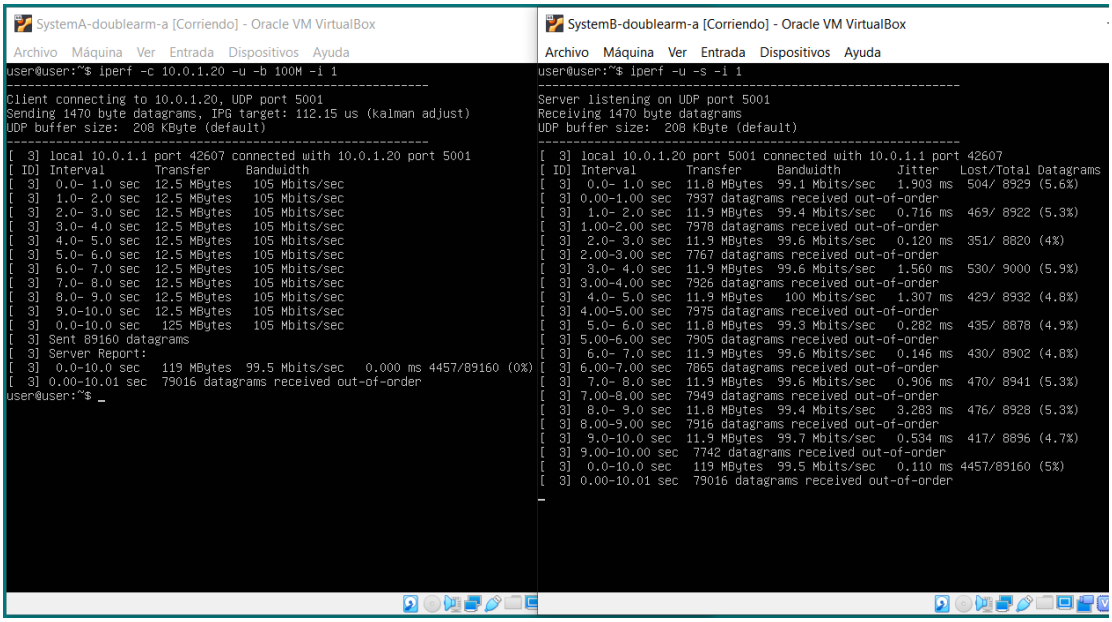
Visualitzem a continuació un llançament de PING entre els dos sistemes:



Il·lustració 36: Proves double-arm 1 - Prova retard sobre ICMP

Com es pot observar, novament, el retard total per a cada paquet ICMP és de 20 mil·lisegons, provocat pel retard en el paquet d'anada i de tornada tal i com s'ha explicat en el punt anterior.

Veiem, aleshores, els resultats de les proves amb IPERF:



Il·lustració 37: Proves double-arm 1 - Proves de reordenament i pèrdua de paquets

Com es pot observar en la captura, els resultat obtinguts són molt similars als que ja hem vist en l'apartat anterior, amb una mitjana d'un 5% de paquets perduts, i la xifra de paquets desordenats.

8.3. Proves en arquitectura *double-arm* amb interfície pont

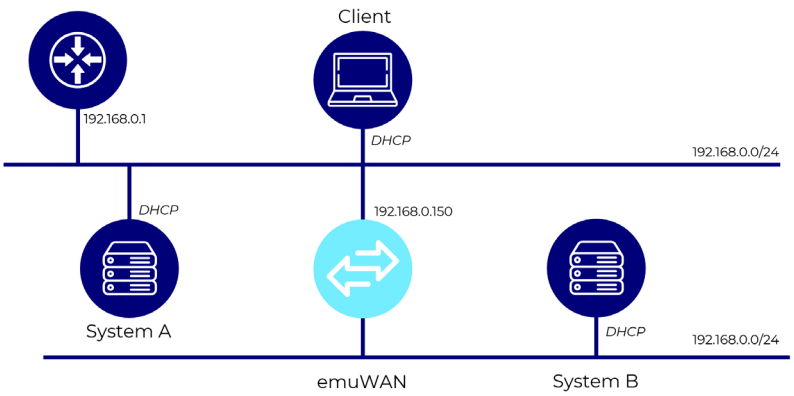
Aquest bloc de proves el farem amb una segona arquitectura de dues interfícies de xarxa per al dispositiu emulador. En aquest cas, una interfície de xarxa contactarà amb un segment de la xarxa a on hi trobarem el client i un dels extrems de la connexió; i una segona interfície de xarxa es connectarà a un segment de xarxa diferent a on trobem l'altre extrem de la connexió.

Aquesta arquitectura ens permet establir una interfície pont que interconnecti les dues interfícies de xarxa presents al dispositiu. D'aquesta forma, ens podem estalviar configuracions d'encaminament. Tanmateix, com que no tenim una interfície de xarxa dedicada a gestió, tots els processos de gestió es poden veure afectats per l'emulació.

Val a dir, que amb aquesta arquitectura s'han de tenir algunes consideracions prèvies:

1. La configuració de les interfícies de xarxa, s'ha de fer per línia d'ordres, ja que és necessari que aquestes no tinguin cap adreça IP.
2. L'adreça IP s'ha d'establir a la interfície de xarxa de tipus pont, que és la que gestiona el nivell 3 de la connexió.
3. En establir la interfície pont, podrem disposar de serveis com el DHCP en els dos segments de xarxa que interconnecta. S'ha de tenir cura que no hi hagi dos serveis DHCP que puguin generar conflicte.

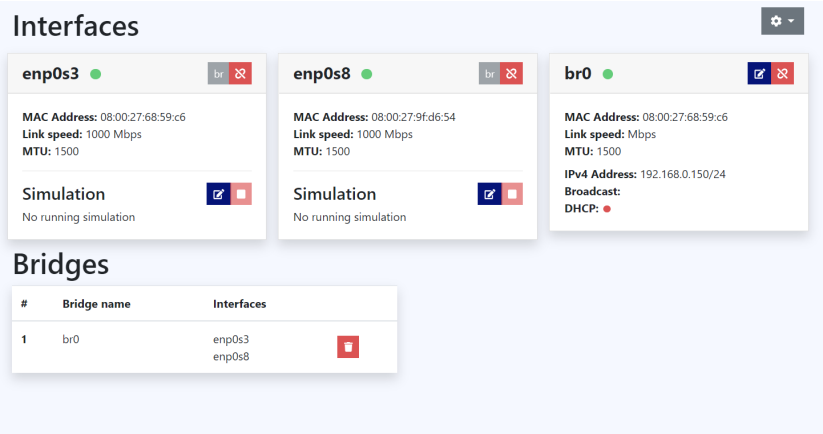
Amb aquestes consideracions, veiem l'esquema de xarxa d'aquest bloc de proves:



Il·lustració 38: Proves double-arm 2 - Esquema de xarxa

Tal i com veiem en la figura, el *System A* i el *System B* prenen les seves adreces de xarxa per DHCP. Així mateix, el dispositiu *emuWAN* rep la seva adreça IP de forma manual tal i com s'ha considerat anteriorment.

Un cop hem establert l'escenari segons es determina en la figura, i hem comprovat que la connexió entre els dos sistemes dels extrems funciona correctament, la interfície de gestió de l'emulador presenta el següent aspecte:



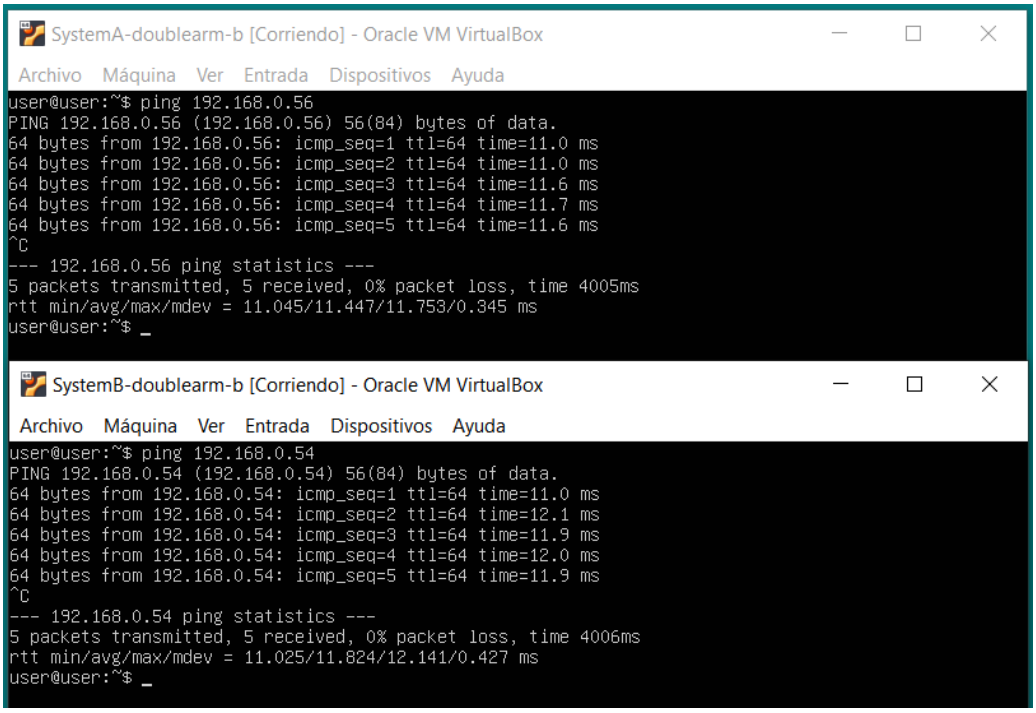
Il·lustració 39: Proves double-arm 2 - Interfície de l'emulador

Veiem en aquesta il·lustració, que es mostren tres interfícies de xarxa, les dues interfícies físiques per una banda; i per l'altra la interfície de tipus pont, que té configurada l'adreça IP del sistema. A més, ens apareix un nou panell amb la configuració de la interfície pont.

Ara doncs, a diferència dels escenaris anteriors, en aquest escenari que es planteja, hem d'escollir a quina de les dues interfícies volem establir els paràmetres d'emulació. Si escollim la interfície que es representa en la branca superior en l'esquema, l'emulació provocarà una degradació en el trànsit de gestió.

Si per contra escollim la interfície que es representa en la branca inferior, l'emulació no afectarà al tràfic de gestió. Establirem, per tant, l'emulació en aquesta branca, tot i que podríem establir els paràmetres en qualsevol de les dues branques o a totes dues.

Amb això, un cop establerta la simulació com en les proves anteriors, aquest és el resultat del PING:



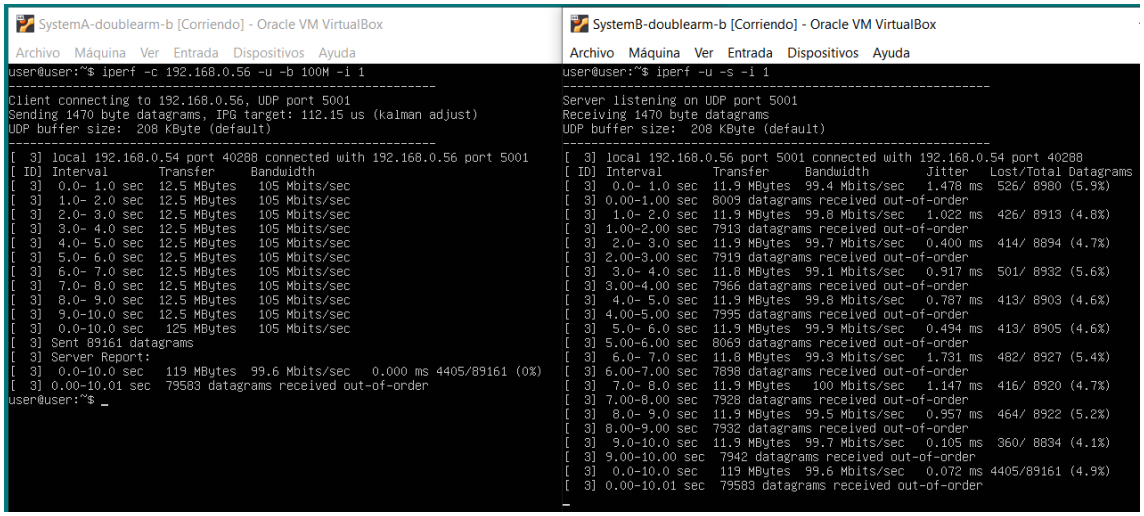
Il·lustració 40: Proves double-arm 2 - Prova retard sobre ICMP

Com es pot comprovar, en aquest cas podem veure una latència afegida de 10 mil·lisegons en els dos sistemes. Això és a causa que en aquest cas, el retard s'aplica només un cop en el temps d'anada i tornada (RTT, *round trip time*) de la prova ICMP.

Aprofitant l'explicació de la primera bateria de proves, veiem que el *System A* sofreix el retard quan el paquet ICMP s'envia cap al *System B*, però el *System B*, sofreix el retard quan la resposta de l'ICMP viatja cap a ell mateix.

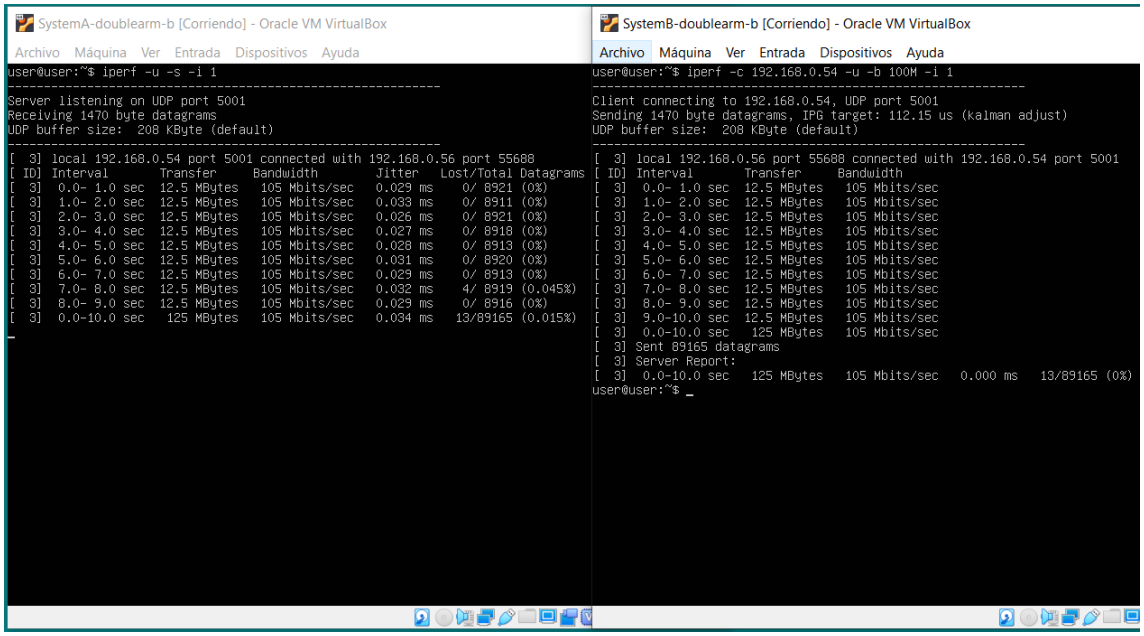
Arribats a aquest punt, passem a executar les proves IPERF. Mentre que en les proves anteriors era irrellevant quin extrem actuava de servidor i quin de client, en aquest cas no ho serà ja que, recordem-ho, estem executant proves amb la pila de protocols UDP, de forma que no hi ha paquet *ACK*.

A la primera prova, establirem el *System B* com a servidor, i al *System A* com a client:



Il·lustració 41: Proves double-arm 2 - Proves de reordenament i pèrdua de paquets 1

A la segona prova, establirem el *System A* com a servidor, i al *System B* com a client:



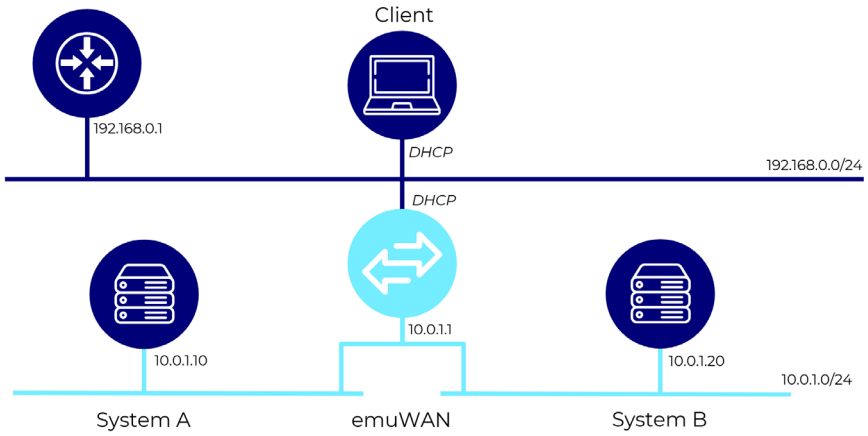
Il·lustració 42: Proves double-arm 2 - Proves de reordenament i pèrdua de paquets 2

Com podem observar, en la primera prova, l'afectació de l'emulació és clara, provocant l'efecte esperat de pèrdua i reordenament. En la segona prova, aquesta afectació és inexistent, ja que la interfície de l'emulador que envia els paquets no està executant cap emulació.

8.4. Proves en arquitectura *triple-arm*

Aquesta és, probablement, l'arquitectura més recomanada. L'arquitectura de tres interfícies de xarxa ens permet aïllar totalment el procés d'emulació de la gestió de la mateixa.

A continuació es mostra, i explica, l'entorn que s'ha desplegat per executar les proves:



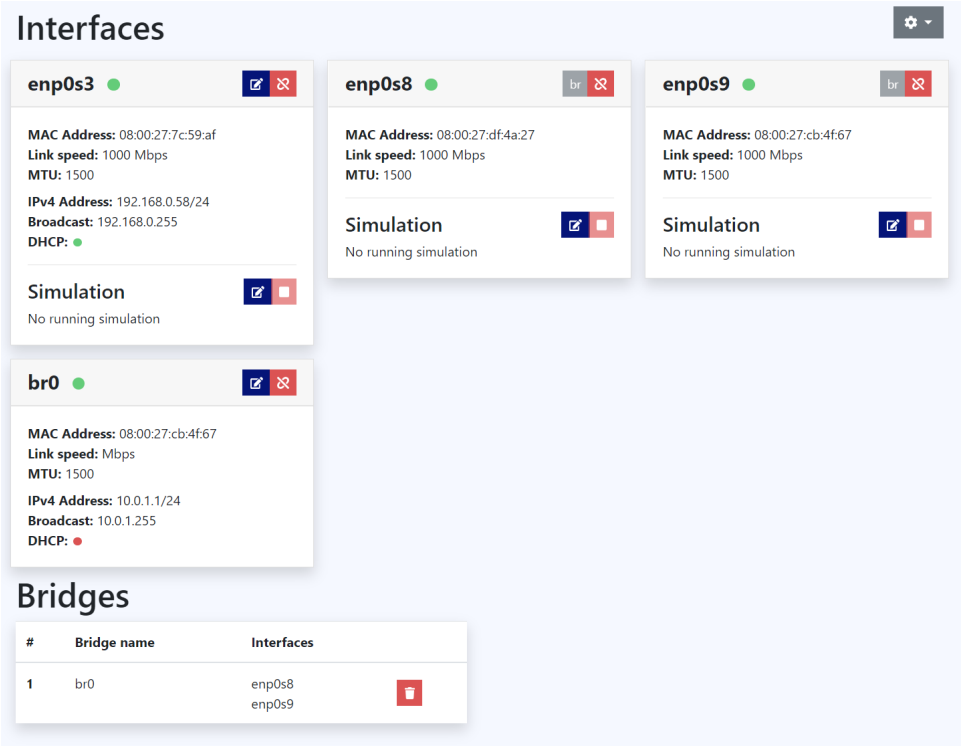
Il·lustració 43: Proves triple-arm - Esquema de xarxa

Al gràfic podem veure com el segment inferior està dividit en dos. Cadascun d'aquests subsegments està connectat a una de les interfícies de xarxa del *emuWAN*. A continuació, establim una interfície pont que uneixi aquests dos subsegments, creant-ne un de sol i permetent la comunicació en nivell 2 entre els equips *System A* i *System B*.

A més, hem establert una interfície específica per a la gestió, que rep la seva adreça IP per DHCP. Addicionalment, hem permès que l'equip *emuWAN* pugui encaminar el tràfic entre el segment 192.168.0.0/24 i el segment 10.0.1.0/24 per donar accés a Internet als equips *System A* i *System B*, encara que no és un requisit per a aquesta simulació.

Finalment, hem establert l'adreça IP 10.0.1.1 a la interfície pont. Aquesta adreça és la porta d'enllaç dels equips del segment inferior.

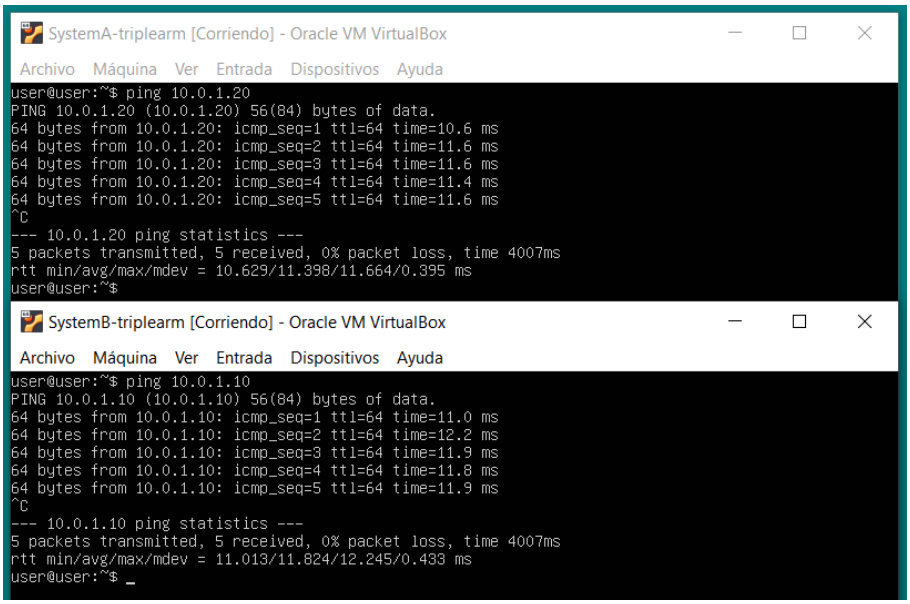
Amb tota aquesta configuració, veiem com queda la interfície gràfica de l'emulador d'enllaços WAN:



Il·lustració 44: Proves triple-arm - Interfície de l'emulador

Podem visualitzar tots els components descrits anteriorment: tres interfícies de xarxa, una interfície pont i el corresponent bloc de gestió.

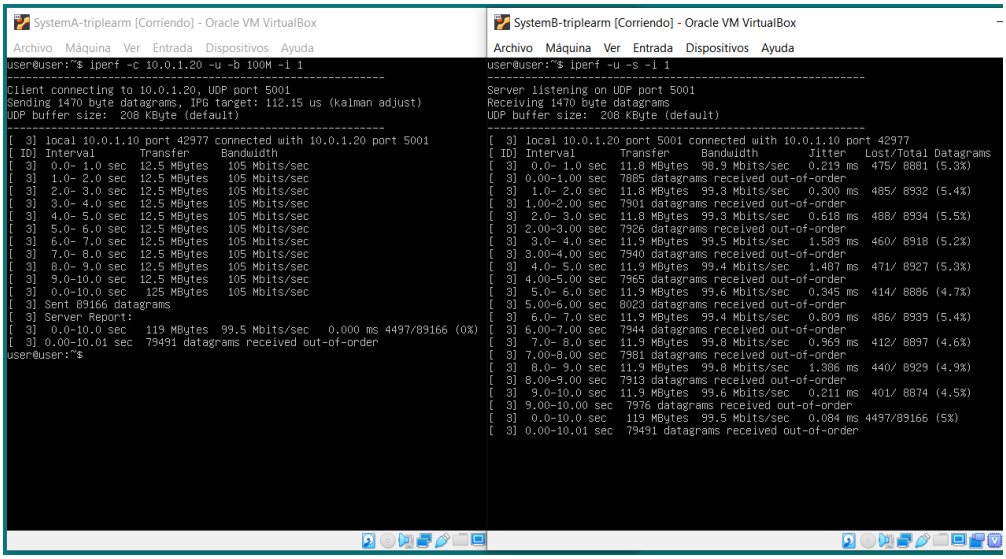
Amb això, establim els mateixos paràmetres d'emulació de les proves anteriors a la interfície `enp0s9`, i comprovem que la latència s'incrementa:



Il·lustració 45: Proves triple-arm - Prova retard sobre ICMP

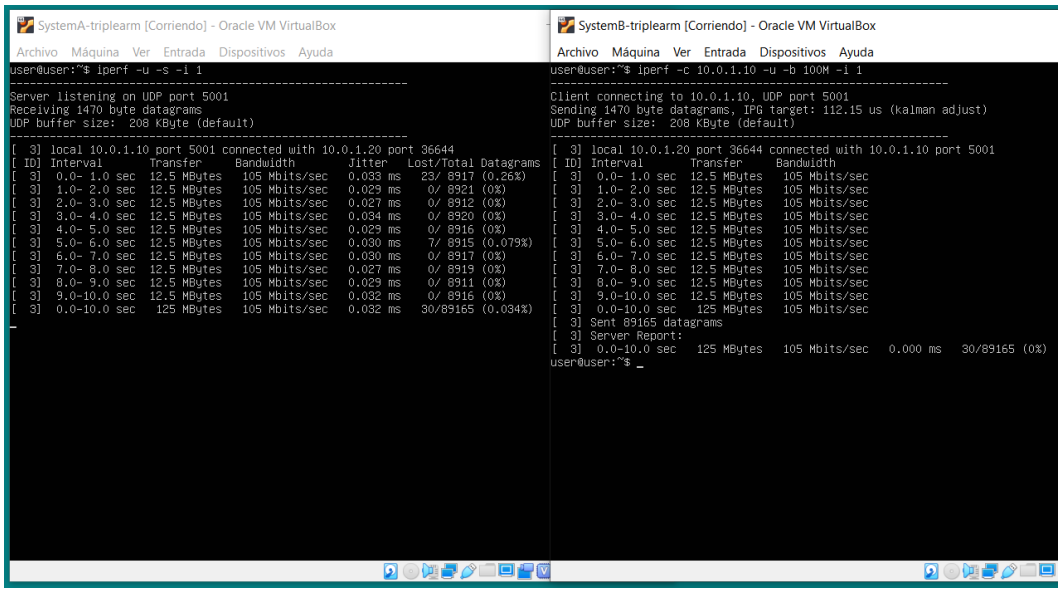
El resultat d'aquesta prova és el mateix que hem vist en el bloc de proves anterior en què el retard s'aplica un sol cop: o en l'enviament del paquet ICMP, o en el paquet de resposta.

Veiem a continuació la prova IPERF amb el *System B* com a servidor i el *System A* com a client:



Il·lustració 46: Proves triple-arm - Proves de reordenament i pèrdua de paquets 1

I finalment, veiem la mateixa prova amb el *System A* com a servidor i el *System B* com a client:



Il·lustració 47: Proves triple-arm - Proves de reordenament i pèrdua de paquets 2

Igual que en el bloc de proves anteriors, veiem que l'afectació d'aquesta emulació només succeeix en una sola direcció. A diferència de l'arquitectura anterior, però, podem corregir aquest comportament aplicant els paràmetres desitjats a la interfície germana dins el pont, sense provocar problemes de connectivitat en la interfície de gestió.

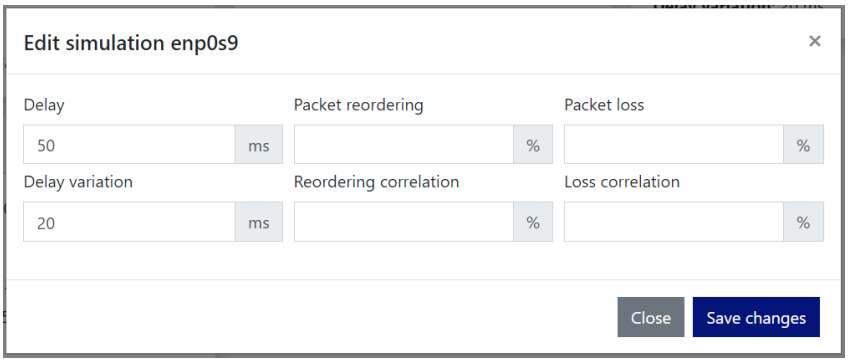
Com ja s'ha explicat, un altre avantatge d'aquest format és la possibilitat d'afegir més interfícies i emulacions en la mateixa instància del dispositiu d'emulació. Malauradament, VirtualBox, la plataforma utilitzada per fer aquestes proves, limita el nombre d'interfícies de xarxa a 4, de forma que no es poden fer les corresponents proves ja que requeririen d'un mínim de 5 interfícies.

8.5. Proves de variació i correlació

En els blocs de proves anteriors s'han executat emulacions sense variació ni correlació per tal de poder observar els resultats d'aquestes proves d'una forma més senzilla. En aquest apartat, comprovarem el correcte funcionament dels paràmetres de variació de latència i correlació de pèrdua i reordenament de paquets.

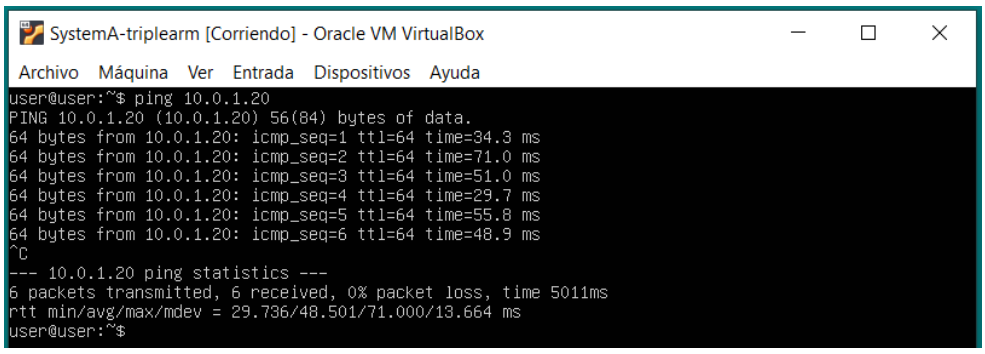
Farem un conjunt de proves aïllades per tal de veure l'afectació que representen aquestes variacions i correlacions. Per fer-ho, utilitzarem l'escenari de tres interfícies de xarxa que s'ha usat anteriorment.

Primerament, comprovarem l'efecte de la variació de temps en la latència. D'aquesta manera, establirem una nova emulació amb un retard de 50 mil·lisegons i una variació de 20 mil·lisegons:



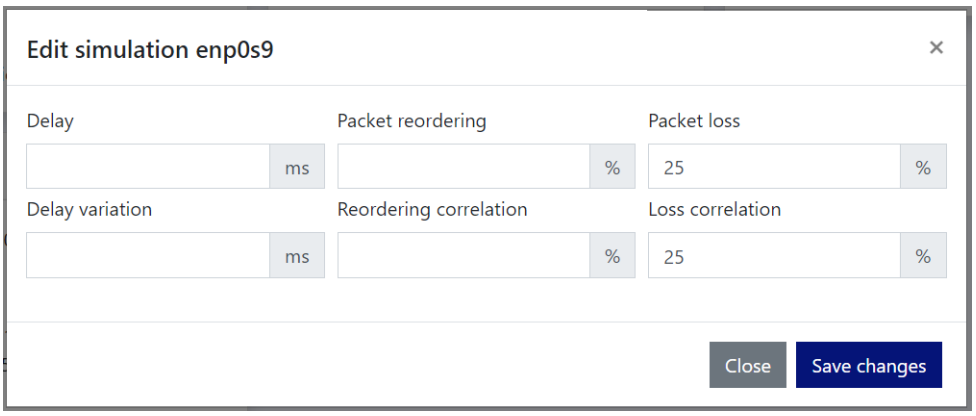
Il·lustració 48: Proves variació i correlació - Emulació de variació de retard

Com podem observar en la següent il·lustració, executem una prova PING des del *System A* cap al *System B*, i esperem visualitzar uns temps de resposta que oscil·lin entre els 30 mil·lisegons i una mica més de 70 mil·lisegons:



Il·lustració 49: Proves variació i correlació - Resultat de variació de retard

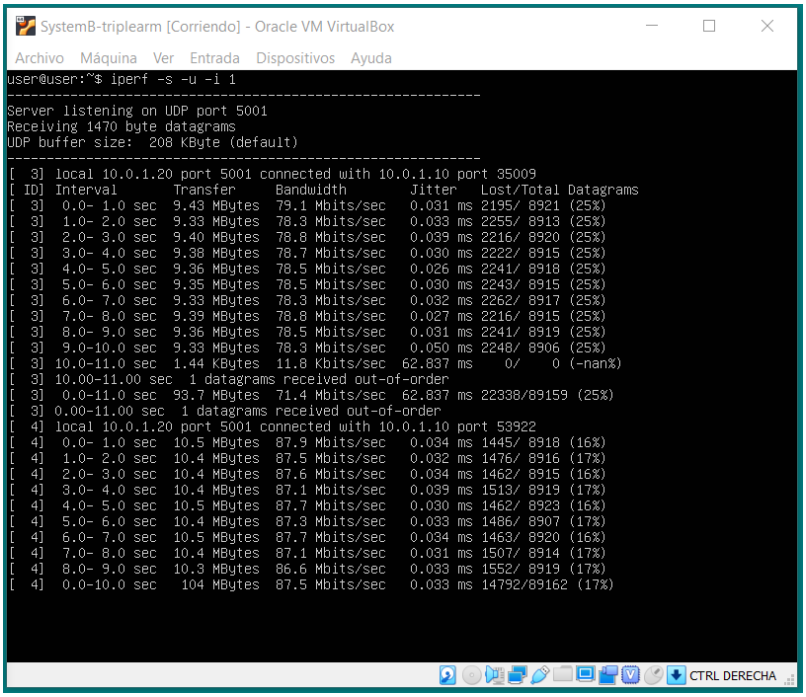
A continuació, comprovem l'efecte de la correlació en la pèrdua de paquets. Per a fer-ho, establirem una emulació de mostra amb una pèrdua del 25% dels paquets, i, a continuació, establirem una correlació del 25% sobre la pèrdua anterior, del 25%. Veiem els paràmetres de la prova:



Il·lustració 50: Proves variació i correlació - Correlació de variació de pèrdua

A continuació llancem les proves corresponents amb IPERF per visualitzar els resultats. Fem el llançament d'IPERF amb el *System A* com a client i el *System B* com a servidor; utilitzant el protocol UDP, com en totes les proves anteriors.

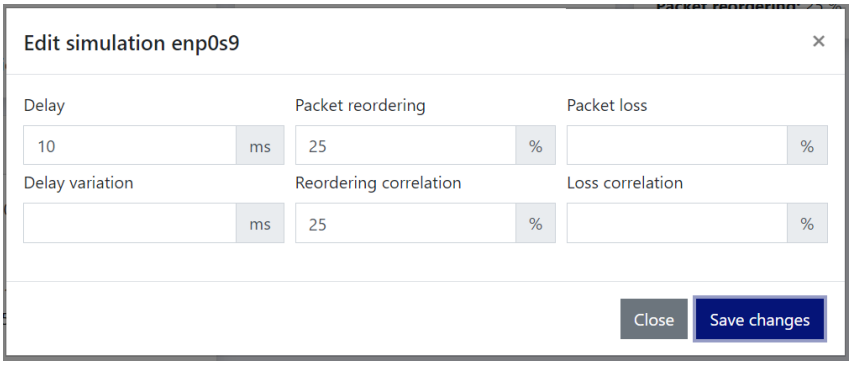
Es fan les dues proves de forma consecutiva, de forma que podem visualitzar els resultats de forma conjunta directament en el terminal del sistema servidor:



Il·lustració 51: Proves variació i correlació - Resultat de correlació de pèrdua

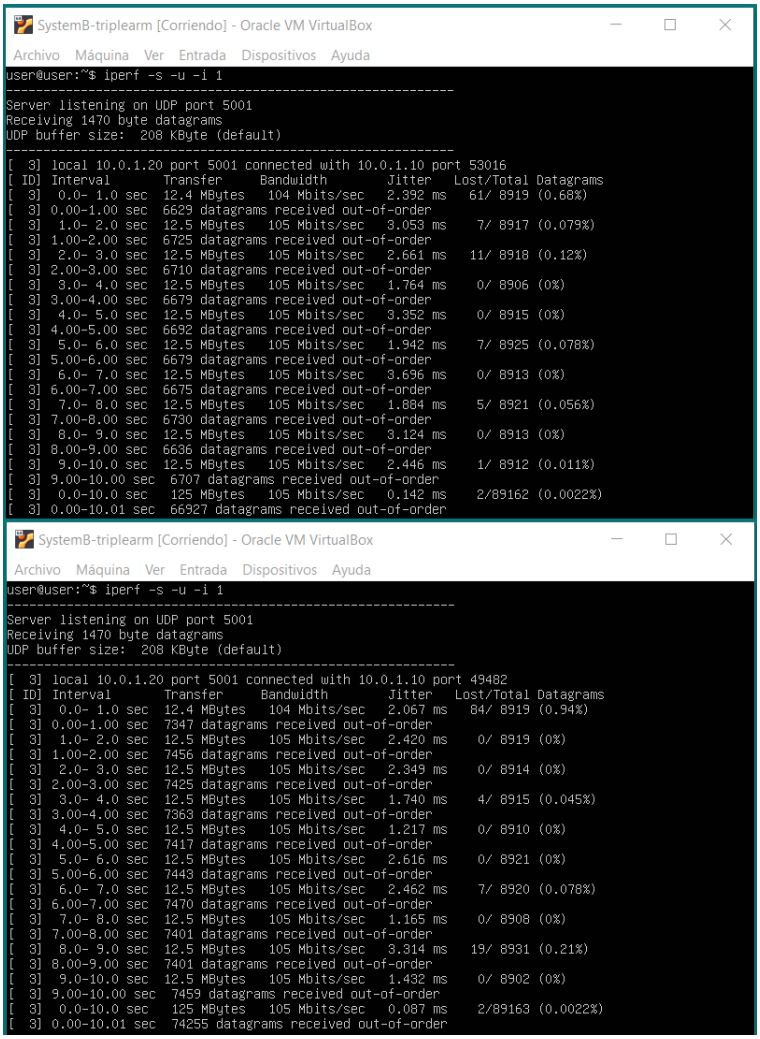
Es pot observar, que en la primera prova, la pèrdua total ha estat d'exactament un 25% dels paquets, tal com s'ha establert. Per contra, l'efecte de la correlació del 25% ha produït una modificació en la segona prova, que presenta una pèrdua mitjana del 17% dels paquets.

Finalment, repetim la prova anterior, però amb el reordenament de paquets. Establirem una mostra amb el 25% del reordenament, tenint en compte que aquest ens afegirà 10 mil·lisegons de retard. A continuació afegirem un 25% de correlació sobre els resultats anteriors. Els paràmetres de la prova, seran:



Il·lustració 52: Proves variació i correlació - Correlació de variació de reordenament

Observem a continuació els resultats de les dues proves i comprovem que es produeix una variació en el nombre de datagrames rebuts amb un ordre diferent a l'esperat:



Il·lustració 53: Proves variació i correlació - Resultat de correlació de reordenament

8.6. Proves amb aplicacions

Per tal de poder demostrar la utilitat de l'emulador WAN amb exemples més gràfics, es mostren en aquest apartat dues proves realitzades amb aplicacions d'anàlisi de xarxes.

En totes les proves d'aquest apartat s'utilitza l'esquema de xarxa mostrat en la il·lustració 42 d'aquest treball, corresponent a l'arquitectura *triple-arm*. Els equips dels extrems de la xarxa, *System A* i *System B*, són equips amb el sistema operatiu Windows 10.

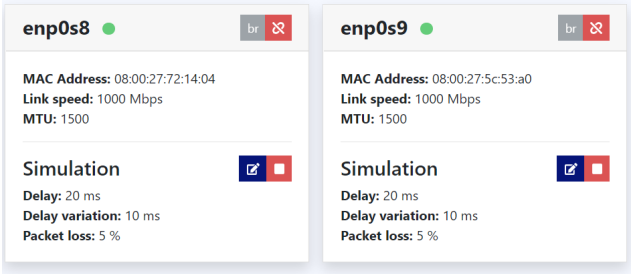
No és objecte d'aquest treball entrar a detallar ni el sistema de trucades SIP ni el programa StarTrinity, així com tampoc la transferència local de fitxers o el programa HELIOS, de forma que s'omet deliberadament aquesta informació.

8.6.1. Trucada SIP – StarTrinity

La primera prova la farem amb StarTrinity[5], un software de simulació de trucades VoIP que permet, veure i analitzar el comportament d'una xarxa durant una trucada VoIP. Ens interessa veure quina afectació produeix a una trucada una emulació pel que fa al *jitter* i a la pèrdua de paquets.

Un cop instal·lat i configurat el programa StarTrinity en els dos equips *System A* i *System B*; considerem que el sistema A serà qui faci la trucada i el sistema B el que la rebi. Se segueixen les següents passes per a l'emulació:

1. Es configura al sistema B perquè respongui a la trucada als 2 segons i que la finalitzi als 35 segons d'haver despenjat.
2. Llancem, sense cap emulació, una trucada des del sistema A, que ens servirà com a referència.
3. Introduïm els següents paràmetres a l'emulació: 20ms de retard a les dues interfícies, amb una variació de 10ms, a fi de produir *jitter*. Una pèrdua del 5% dels paquets. El resultat d'aquesta emulació, a la interfície de l'emulador és:



Il·lustració 54: Proves amb StarTrinity - Configuració de l'emulació

4. Repetim la trucada que hem fer en el punt 2 per obtenir el resultat i poder comparar-les.

Amb aquestes dues trucades ja tenim prou per comprovar els efectes de l'emulació sobre l'enllaç, com podem comprovar a la taula:

Answer delay	Answered duration	Caller lost packets	Caller max RFC3550 jitter	Called lost packets	Called max RFC3550 jitter	RTCP RTT (min/avg/max)	RTCP caller packet loss	RTCP caller jitter	RTCP called packet loss	RTCP called jitter
2.062,00ms	35.155,00ms	0/(1757+0) (0.00%)	14,70ms	93/(1664+93) (5,29%)	22,47ms	29,27/270,55/732,00ms	17,93% (315/1757)	0,00/16,86/37,00ms	18,70% (311/1663)	0,57/13,19/23,55ms
2.048,00ms	35.150,00ms	0/(1757+0) (0.00%)	13,48ms	0/(1759+0) (0.00%)	14,00ms	0,60/2,24/6,47ms	0,00% (0/1757)	0,00/13,31/26,00ms	0,00% (0/1759)	0,80/10,71/17,44ms

Il·lustració 55: Proves amb StarTrinity - Resultats de les proves[6]

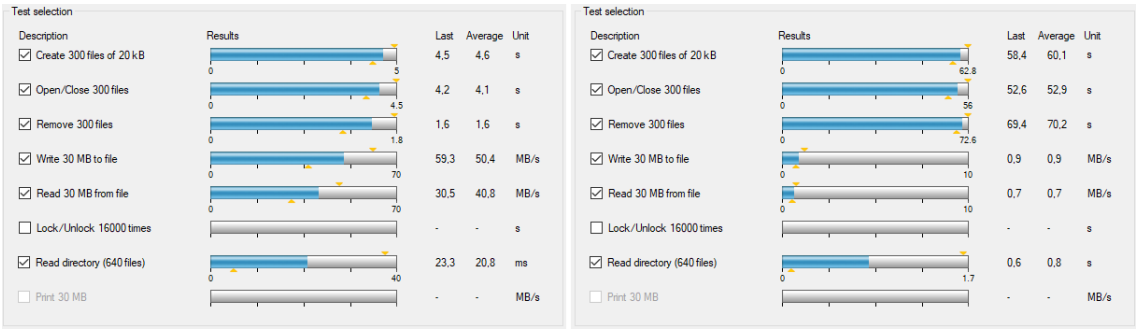
[5] StarTrinity - http://startrinity.com/
[6] Aquesta imatge està disponible amb major resolució a l'ANNEX III – Imatges (figura 1)

Les trucades es mostren en ordre cronològic invers. Podem veure en la columna *"Called lost packets"*, com s'ha produït una pèrdua del 5,29% dels paquets. A més, podem veure una afectació significativa al *jitter*, que augmenta dels 14,70 mil·lisegons als 22,47 mil·lisegons.

8.6.2. Transferència de fitxers local – HELIOS lantest

La darrera prova d'aquest capítol és la transferència de fitxers locals entre els sistemes A i B en el mateix escenari anterior. D'aquesta forma, al Sistema B s'hi ha configurat una carpeta compartida accessible des de la xarxa local a la qual ens connectarem mitjançant el programa HELIOS lantest[7] i sobre el qual executarem les corresponents proves.

Un cop feta una prova de rendiment sense emulació, establim l'emulació amb els següents paràmetres a les dues interfícies del bridge: 10 ms de latència amb una variació de 5 ms i un 1% de pèrdua de paquets. Repetim la prova i comparem els resultats:



Il·lustració 56: Proves amb HELIOS lantest - Resultats de les proves

A la imatge de l'esquerra veiem els resultats de les proves prèvies a activar l'emulador, i, a la de la dreta, els resultats posteriors a l'activació.

Com es pot observar, l'emulació establerta provoca una gran quantitat de retransmissions i alenteix la xarxa fins al punt que l'escriptura de 30 MB de dades passa de fer-se a 50,4 MB/s a 0,9 MB/s. Això demostra com, clarament, l'emulador d'enllaços WAN provoca l'efecte desitjat.

[7] HELIOS lantest - https://www.helios.de/web/EN/products/LanTest.html

9. Casos d'ús

Un cop explicat i demostrat el funcionament de l'aplicació d'emulació d'enllaços WAN, un es pot demanar quines utilitats pot tenir un dispositiu que provoca alentiments i problemes de connectivitat a la xarxa, característiques aquestes mai considerades desitjables.

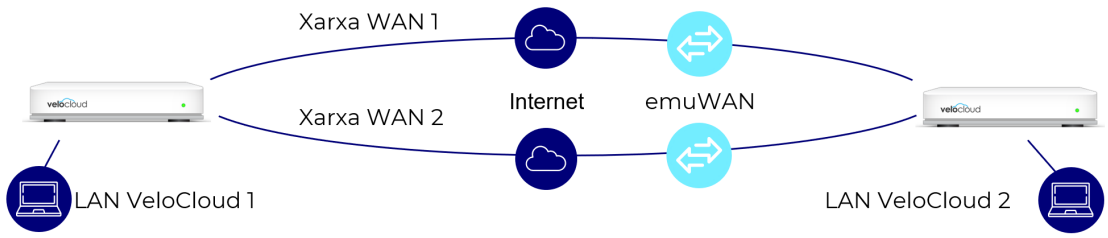
Com ja s'ha explicat al principi d'aquest document, l'aplicació d'emulació d'enllaços WAN s'ha d'entendre com a una eina de laboratori. Aquesta eina permet fer proves dins un entorn controlat i segur, com pot esser una xarxa local, per a provar diferents escenaris.

Un primer escenari que podem esmentar a tall d'exemple és el desenvolupament d'aplicacions. En moltes ocasions els desenvolupadors d'aplicacions en xarxa fan el desenvolupament i proves en una màquina local, fet que, en molts casos, no contempla dificultats de connexió.

Fent ús d'aquest emulador de xarxes WAN, els desenvolupadors d'altres aplicacions poden provar les seves aplicacions en entorns de xarxa no ideals, però sense abandonar l'entorn local, i segur, del seu laboratori.

Un altre escenari, es contempla en la demostració i prova d'entorns d'acceleració WAN, com les solucions SD-WAN. Tal i com s'ha esmentat en la introducció, el projecte naix de la necessitat de demostrar la funcionalitat del producte VeloCloud, de VMware.

Per poder fer aquest tipus de demostracions i proves, es necessita emular un entorn en què dues seus, amb els corresponents equips de VeloCloud, s'interconnecten amb dos enllaços diferents. Es vol comprovar que quan la qualitat dels enllaços es deteriora, o canvia, la solució SD-WAN actua segons les directrius que s'han establert.



Il·lustració 57: Esquema exemple SD-WAN

10. Conclusions

El Treball de Fi de Grau del qual aquest document n'és la memòria, ha suposat el desenvolupament complet d'un emulador d'enllaços WAN amb una interfície gràfica servida a través d'una plana web de gestió tot fent ús de diferents eines i paquets inclosos o instal·lables en els sistemes operatius Linux.

El desenvolupament d'aquesta aplicació s'ha fet parant especial esment en la seva modularitat, per garantir, d'aquesta manera, la possibilitat de continuar de forma senzilla el desenvolupament de nous mòduls i funcionalitats. Aquesta modularitat, com s'ha pogut llegir en aquesta memòria, resideix, fonamentalment, en la separació dels diferents components del *frontend* i del *backend*.

L'aplicació desenvolupada, incorpora dos mètodes de gestió: per una banda, incorpora una API REST amb la qual es pot interactuar des d'scripts i eines complementàries; per una altra banda, s'incorpora una detallada interfície gràfica servida mitjançant una interfície web sense navegació que permet una interacció molt senzilla per a l'usuari final.

A més, el dispositiu d'emulació WAN permet el seu desplegament en diverses arquitectures d'interfícies de xarxa, de forma que poden funcionar a partir d'una sola interfície de xarxa, fins a un nombre il·limitat d'interfícies. Aquesta flexibilitat és transparent per a l'usuari, que pot veure com la interfície d'usuari s'adapta a les diferents arquitectures que s'hi estableixin.

En el darrer punt d'aquesta memòria, hem pogut comprovar com el dispositiu d'emulació WAN, batejat com emuWAN, efectivament, permet aplicar els tres efectes desitjats en els enllaços: retard de paquets amb la seva corresponent variació, i pèrdua i reordenament de paquets, amb la seva corresponent correlació.

Amb tot això, aleshores, es pot considerar que els objectius establerts i explicats al principi d'aquest document s'han complert dins el termini planificat, i, en conjunt, consideram reeixit aquest Treball de Fi de Grau.

Tot i considerar el reeiximent dels objetius, es vol fer constar que, tal i com s'ha explicat anteriorment, romanen elements diversos que estan pendents de esser desenvolupats i afegits a les funcionalitats de l'aplicació.

Entre d'altres, un dels desenvolupaments pendents, per exemple, és l'addició de més efectes a l'emulació. Si bé els efectes incorporats són importants, existeixen altres efectes que poden afectar els paquets durant el trànsit en enllaços WAN, com puguin esser la corrupció de paquets, o la duplicació de paquets.

Un altre desenvolupament pendent és la comprovació de l'elevació de permisos en la inicialització de l'aplicació, o la creació d'un mòdul que permeti la gestió de rutes des de la pròpia interfície gràfica de l'aplicació.

En qualsevol cas, però, es considera que el treball que s'ha proposat i desenvolupat és un bon resum de molts dels continguts que s'han après i assimilat durant l'estudi del Grau en Enginyeria de Tecnologies i Serveis de Telecomunicació amb menció d'Enginyeria Telemàtica.

ANNEX I - Guia de referència API REST

1. NetworkInterface

URL	{*URL*}/networkinterface/
Descripció	Obté un llistat de totes les interfícies de xarxa presents al dispositiu amb els detalls de cadascuna.
Mètode	GET

Consulta HTTP	**Resposta HTTP** (exemple)

No es requereixen dades	{
	"success": true,
	"response": [
	{
	"id": string,
	"status": bool,
	"mtu": int,
	"MAC": string,
	"speed": int,
	"IP4": {
	"status": bool,
	"CIDR": string,
	"address": string,
	"mask": string,
	"broadcast": string,
	"dynamic": bool
	},
	"bridge": null
	},
	...,
	]
	}

URL	{*URL*}/networkinterface/{*ID*}/
Descripció	Obté el detall de la interfície especificada en el paràmetre {*ID*}
Mètode	GET

Consulta HTTP	**Resposta HTTP** (exemple)

No es requereixen dades	{
	"success": true,
	"response": {
	"id": string,
	"status": bool,
	"mtu": int,
	"MAC": string,
	"speed": int,
	"IP4": {
	"status": bool,
	"CIDR": string,
	"address": string,
	"mask": string,
	"broadcast": string,
	"dynamic": bool
	},
	"bridge": null
	}
	}

<table>
<tr><td>URL</td><td colspan="2">{URL}/networkinterface/{ID}/</td></tr>
<tr><td>Descripció</td><td colspan="2">Modifica la interfície especificada en el paràmetre {ID}</td></tr>
<tr><td>Mètode</td><td colspan="2">POST</td></tr>
<tr><td colspan="2">Consulta HTTP (exemple)</td><td>Resposta HTTP (exemple)</td></tr>
<tr><td colspan="2">

```
{
    "CIDR": string,
    "DHCP": bool
}
```

S'ha d'especificar un dels dos camps.

</td><td>

```
{
  "success": true,
  "response": {
    "id": string,
    "status": bool,
    "mtu": int,
    "MAC": string,
    "speed": int,
    "IP4": {
      "status": bool,
      "CIDR": string,
      "address": string,
      "mask": string,
      "broadcast": string,
      "dynamic": bool
    },
    "bridge": null
  }
}
```

</td></tr>
</table>

<table>
<tr><td>URL</td><td colspan="2">{URL}/networkinterface/{ID}/status/</td></tr>
<tr><td>Descripció</td><td colspan="2">Connecta o desconnecta la interfície especificada en el paràmetre {ID}</td></tr>
<tr><td>Mètode</td><td colspan="2">POST</td></tr>
<tr><td colspan="2">Consulta HTTP (exemple)</td><td>Resposta HTTP (exemple)</td></tr>
<tr><td colspan="2">

```
{
    "status": string
}
```

El valor que prendrà la cadena de text serà "up" o "down"

</td><td>

```
{
  "success": true,
  "response": {
    "id": string,
    "status": bool,
    "mtu": int,
    "MAC": string,
    "speed": int,
    "IP4": (optional) {
      "status": bool,
      "CIDR": string,
      "address": string,
      "mask": string,
      "broadcast": string,
      "dynamic": bool
    },
    "bridge": null
  }
}
```

</td></tr>
</table>

2. Simulation

URL	{*URL*}/simulation/{*ID*}/
Descripció	Obté el detall de la simulació existent sobre la interfície indicada en el paràmetre {*ID*}.
Mètode	GET

Consulta HTTP	**Resposta HTTP** (exemple)

No es requereixen dades	`{`
	`  "success": true,`
	`  "response": {`
	`  "id": string,`
	`  "status": bool,`
	`  "delay": int,`
	`  "delayVariation": int,`
	`  "loss": int,`
	`  "lossCorrelation": int,`
	`  "reorder": int,`
	`  "reorderCorrelation": int`
	`  }`
	`}`

URL	{*URL*}/simulation/{*ID*}/
Descripció	Estableix els paràmetres de la simulació sobre la interfície especificada en el paràmetre {*ID*}.
Mètode	POST

Consulta HTTP (exemple)	**Resposta HTTP** (exemple)

`{`
`  "delay": int,`
`  "delayVariation": int,`
`  "loss": int,`
`  "lossCorrelation": int,`
`  "reorder": int,`
`  "reorderCorrelation": int`
`}`
	`{`
	`  "success": true,`
	`  "response": {`
	`  "id": string,`
	`  "status": bool,`
	`  "delay": int,`
	`  "delayVariation": int,`
	`  "loss": int,`
	`  "lossCorrelation": int,`
	`  "reorder": int,`
	`  "reorderCorrelation": int`
	`  }`
	`}`

URL	{*URL*}/simulation/{*ID*}/reset/
Descripció	Estableix els paràmetres de la simulació sobre la interfície especificada en el paràmetre {*ID*}.
Mètode	POST

Consulta HTTP (exemple)	**Resposta HTTP** (exemple)

No es requereixen dades	`{`
	`  "success": true,`
	`  "response": {`
	`  "id": string,`
	`  "status": bool,`
	`  "delay": int,`
	`  "delayVariation": int,`
	`  "loss": int,`
	`  "lossCorrelation": int,`
	`  "reorder": int,`
	`  "reorderCorrelation": int`
	`  }`
	`}`

3. Bridge

URL	{*URL*}/bridge/
Descripció	Obté un llistat d'interfícies pont amb les interfícies de xarxa vinculades
Mètode	GET

Consulta HTTP (exemple)	**Resposta HTTP** (exemple)

No es requereixen dades	`{`
	`  "success": true,`
	`  "response": [`
	`  {`
	`  "id": string,`
	`  "interfaces": {`
	`  "1": string,`
	`  "2": string`
	`  }`
	`  },`
	`  ...,`
	`  ]`
	`}`

URL	{*URL*}/bridge/
Descripció	Estableix una nova interfície pont
Mètode	PUT

Consulta HTTP	**Resposta HTTP** (exemple)

`{`
`  "name": string,`
`  "interfaces": [`
`  string,`
`  string`
`  ]`
`}`
 	
Es requereix un nom per a la nova interfície i els IDs de les interfícies que es volen interconnectar.
	`{`
	`  "success": true,`
	`  "response": {`
	`  "id": string,`
	`  "interfaces": {`
	`  "1": string,`
	`  "2": string`
	`  }`
	`  }`
	`}`

4. Errors

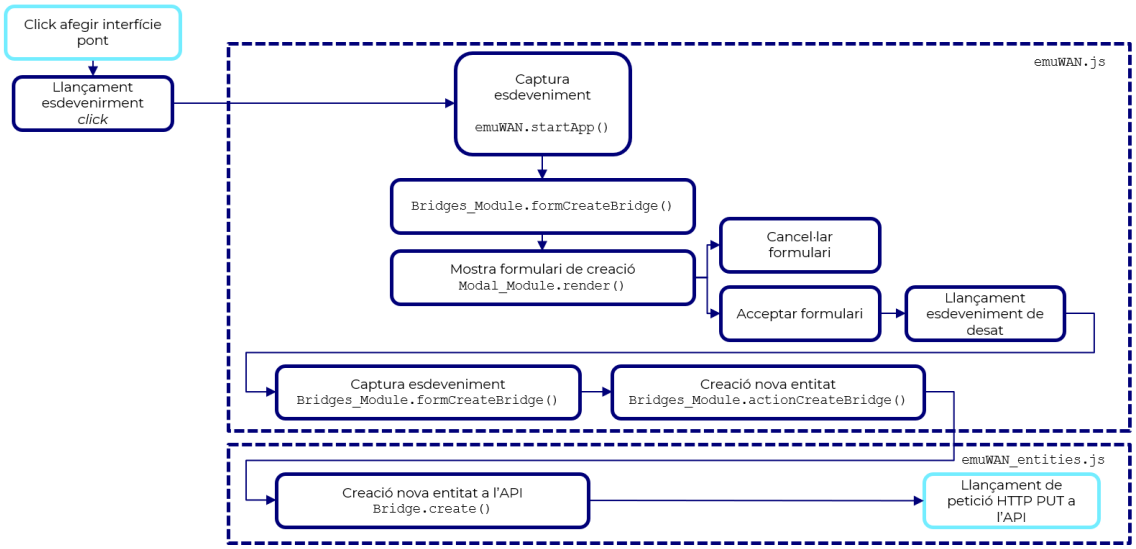
Tots els errors de l'aplicació es retornen en el mateix format, fet que permet una gestió més àgil dels mateixos. El format és el següent:

```
{
  "success": false,
  "errors": [
    {
      "key": string,
      "error": string
    },
    ...,
  ]
}
```

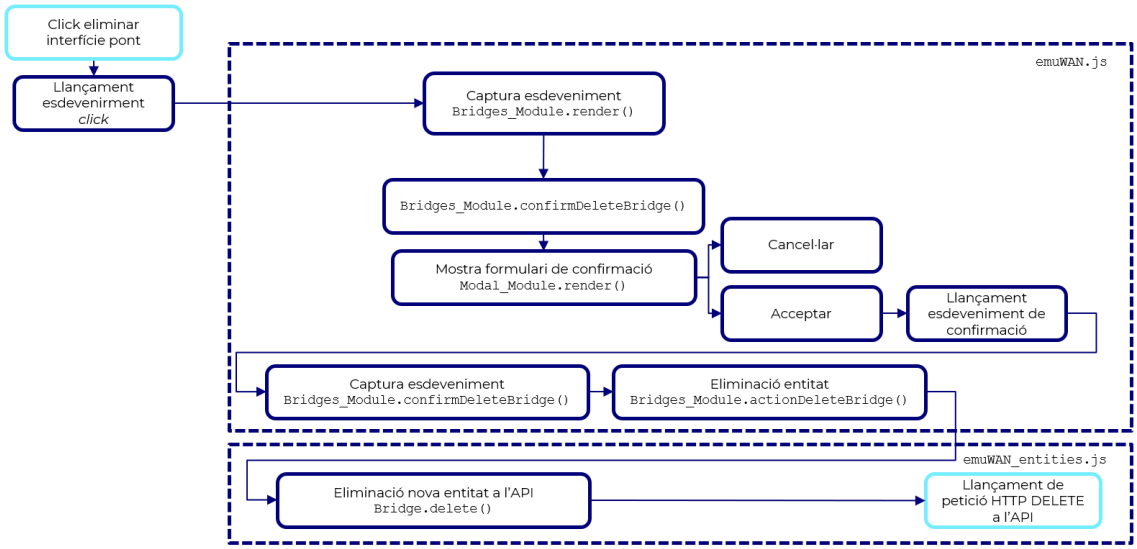
ANNEX II – Diagrames de flux

1. Frontend

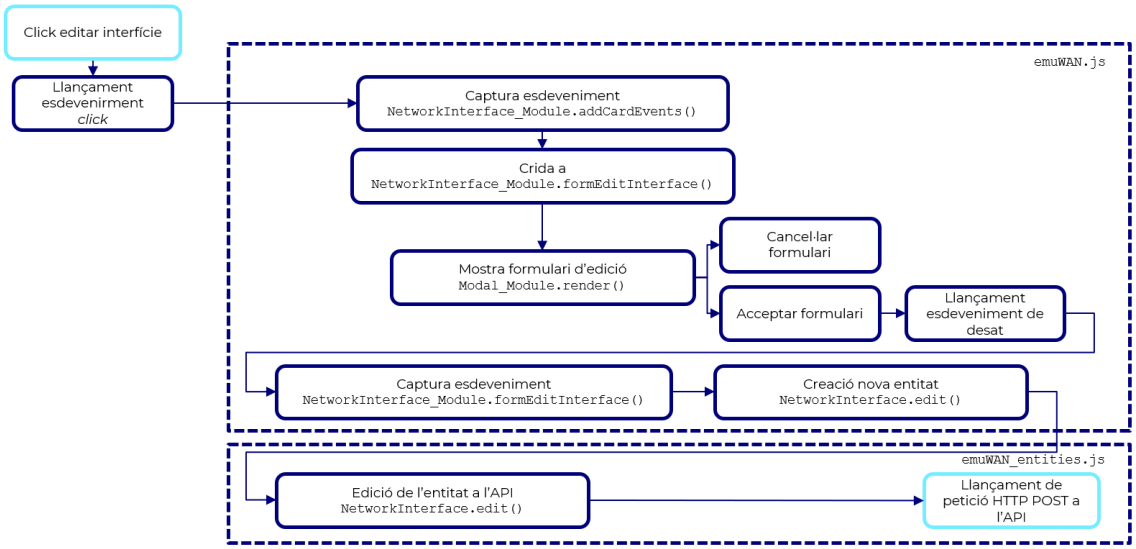
1.1. Creació d'interfícies pont



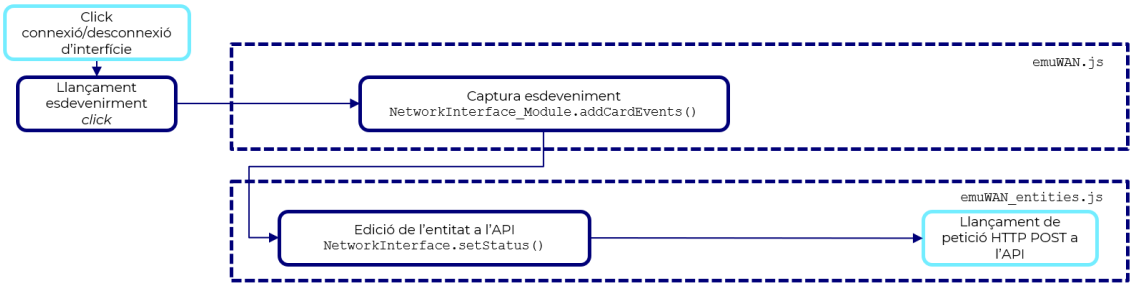
1.2. Eliminació d'interfícies pont



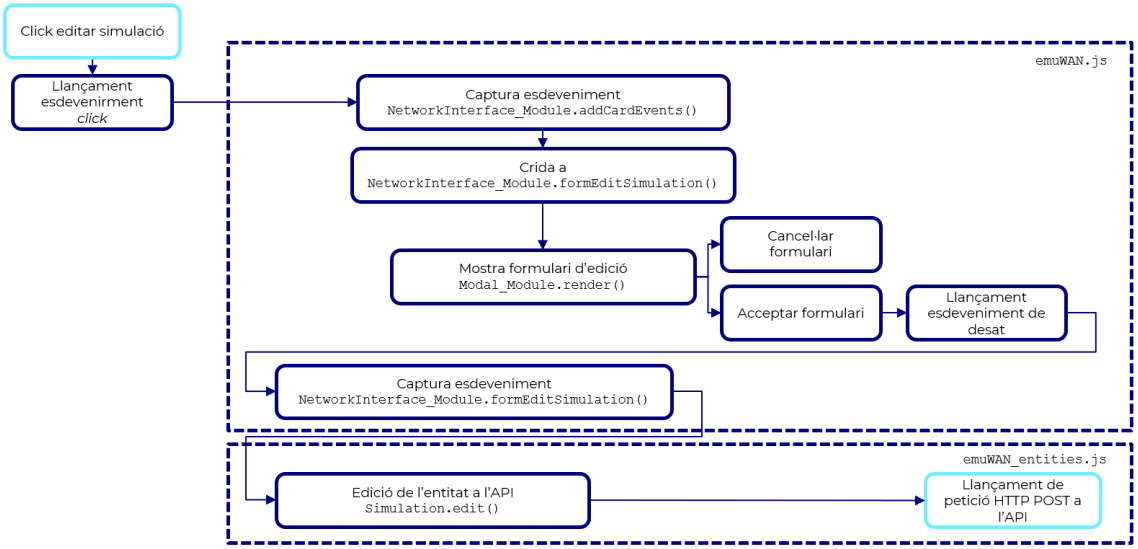
1.3. Edició d'interfícies de xarxa



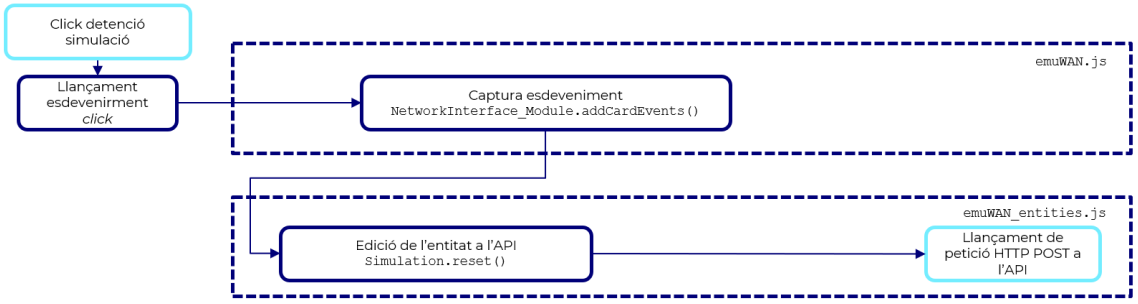
1.4. Connexió i desconnexió d'interfícies de xarxa



1.5. Edició de l'emulació

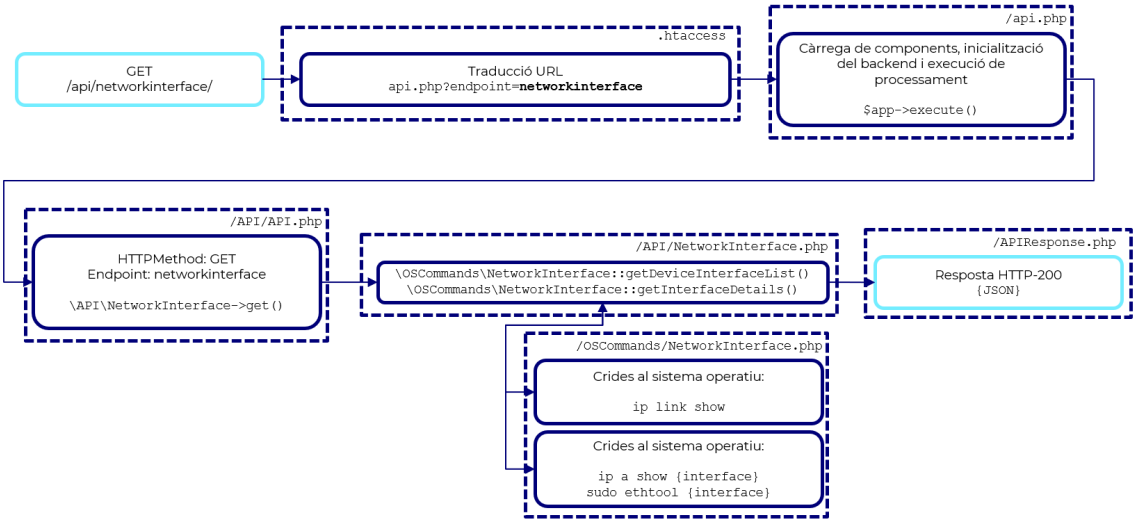


1.6. Detenció de l'emulació

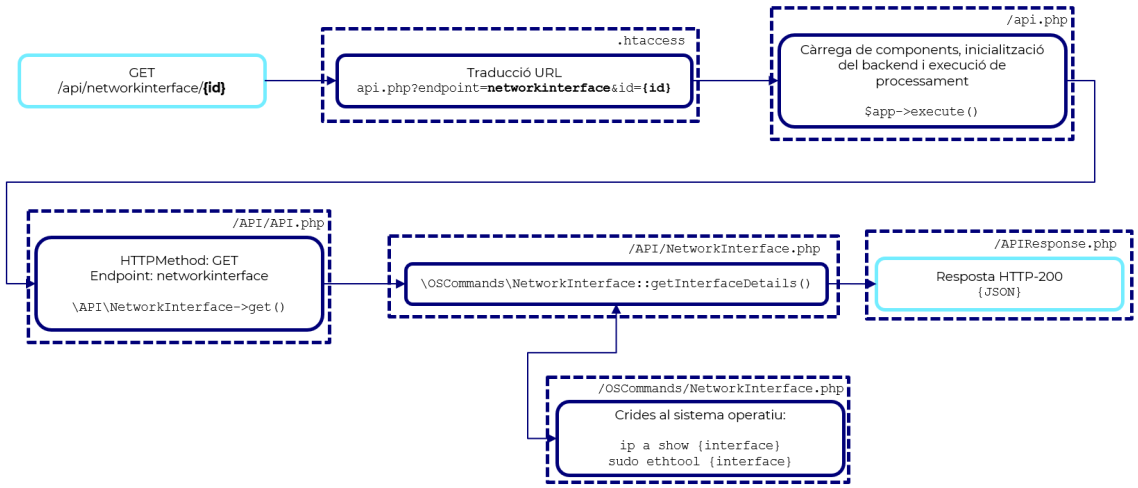


2. Backend

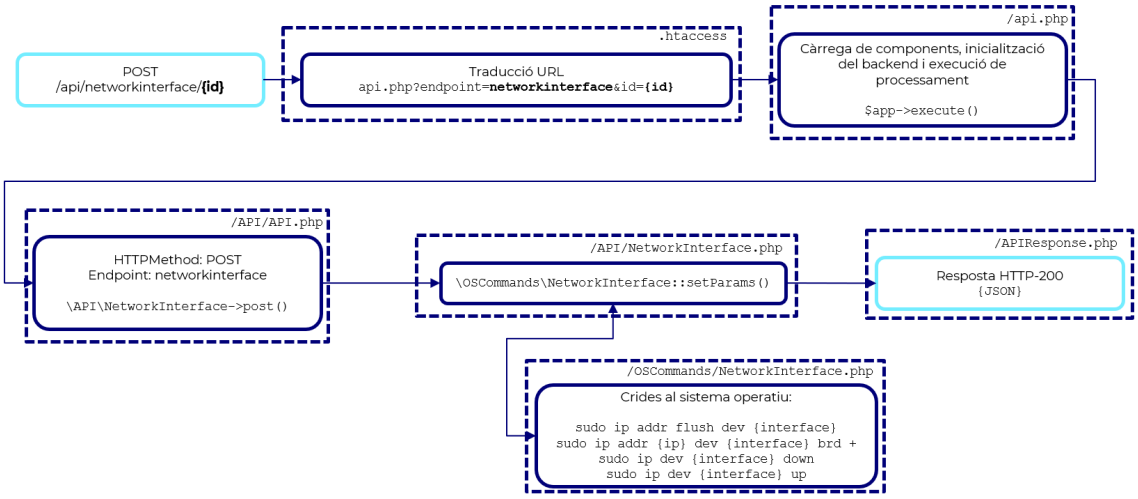
2.1. Obtenció del llistat detallat d'interfícies de xarxa



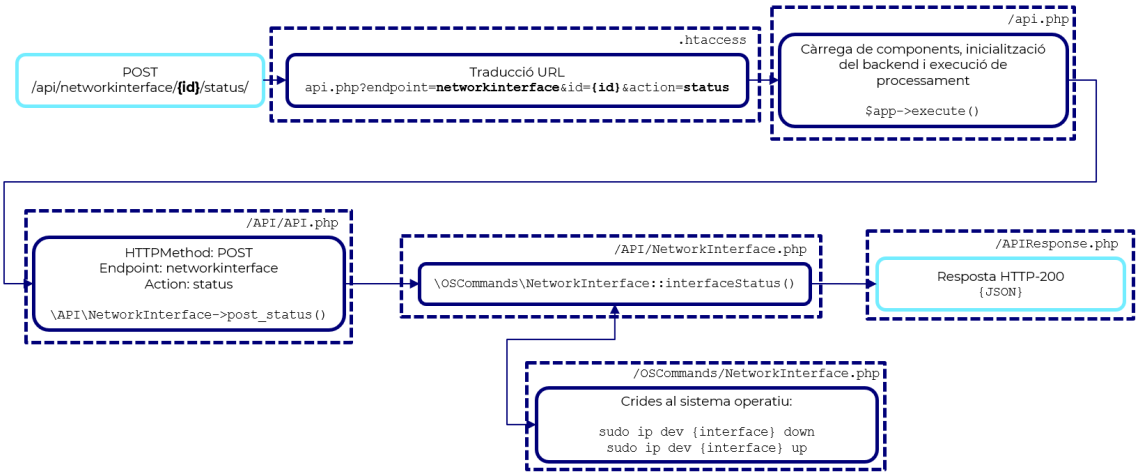
2.2. Obtenció del detall d'una interfície de xarxa



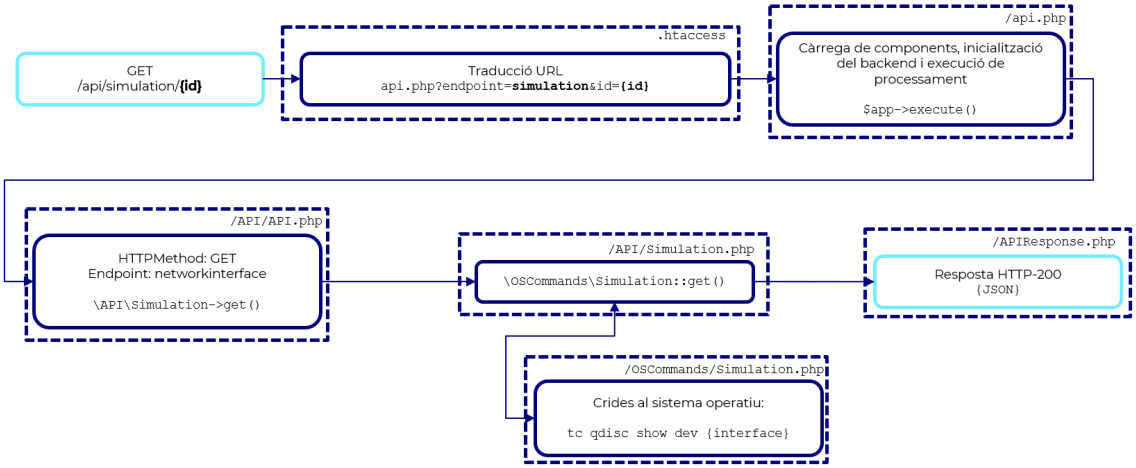
2.3. Edició d'una interfície de xarxa



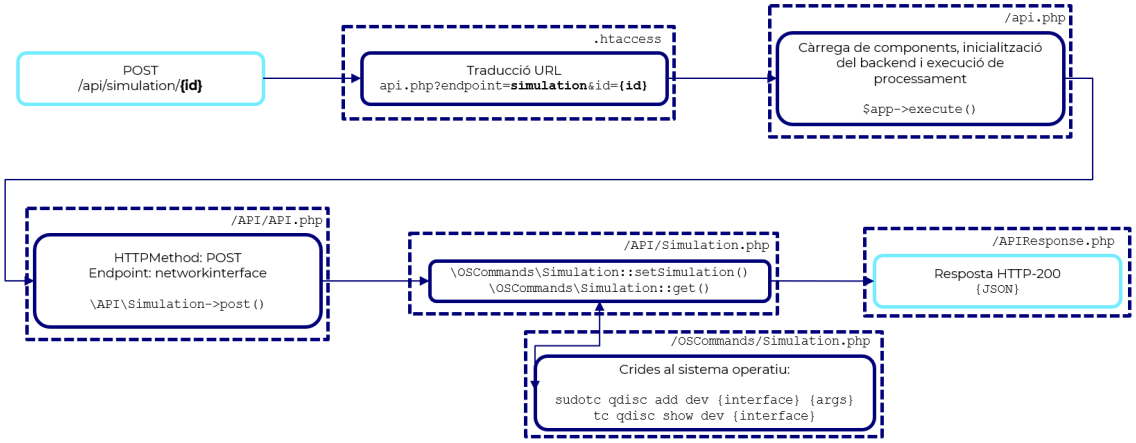
2.4. Connexió i desconnexió d'una interfície de xarxa



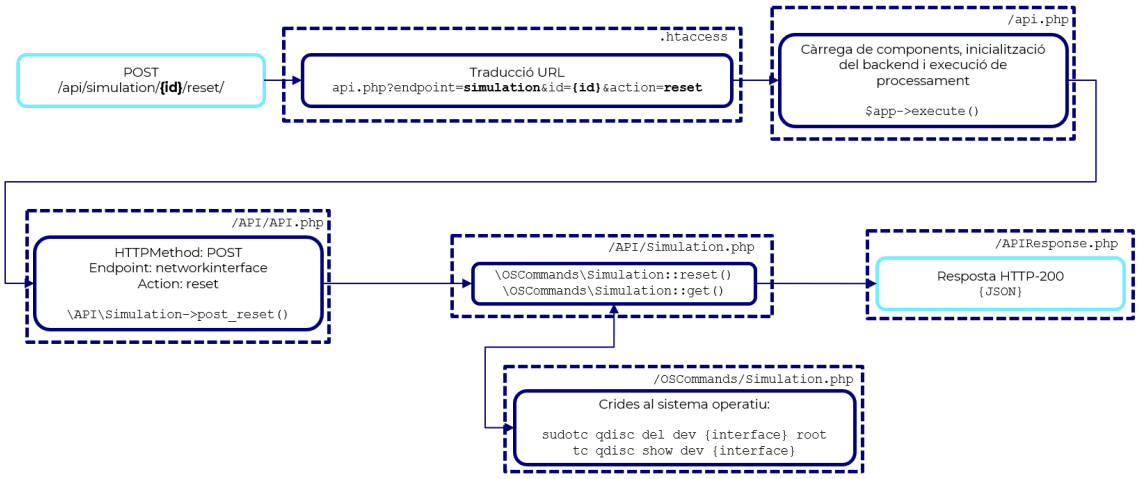
2.5. Obtenció dels detalls d'una emulació



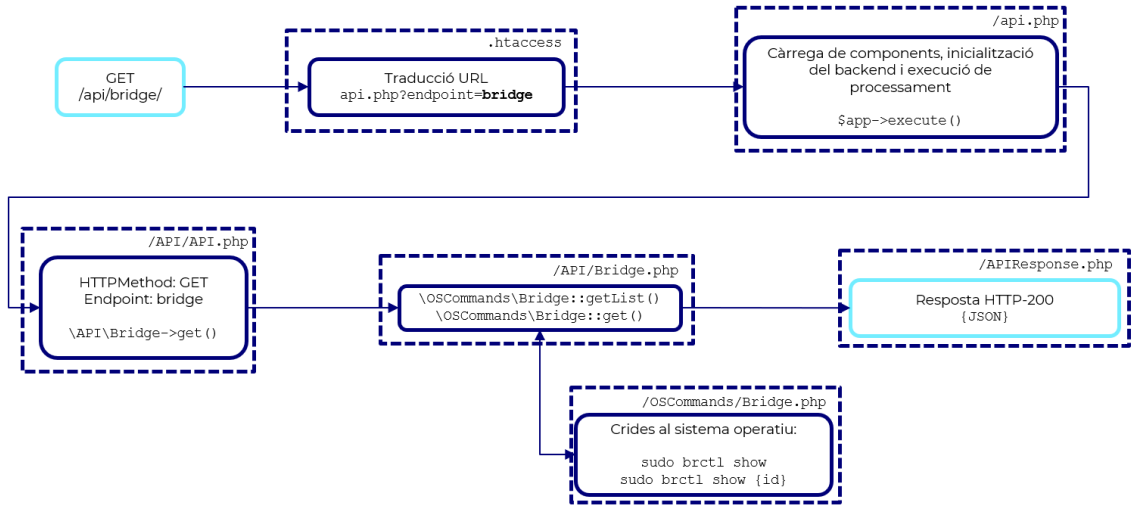
2.6. Edició d'una emulació



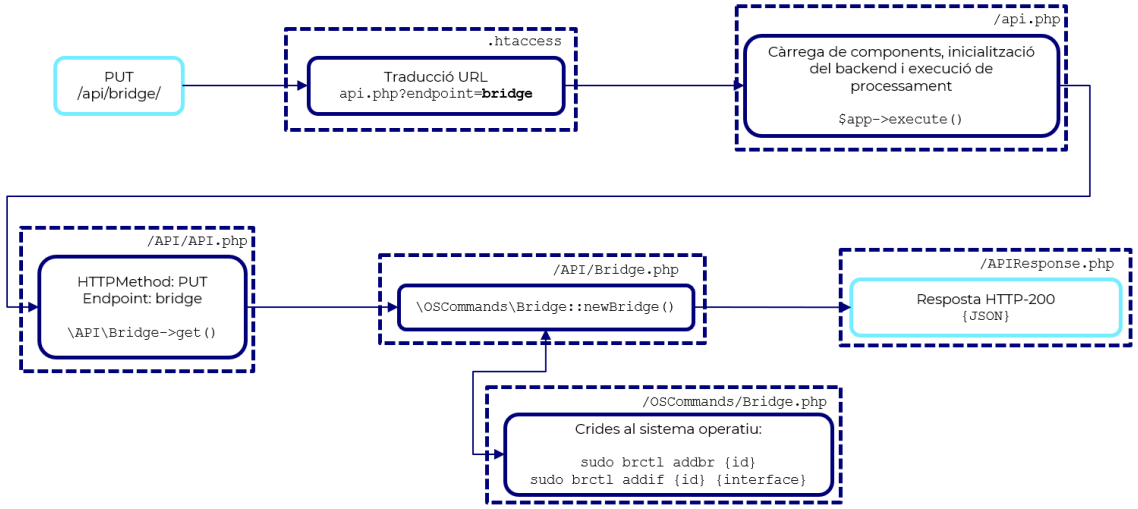
2.7. Restabliment d'una emulació



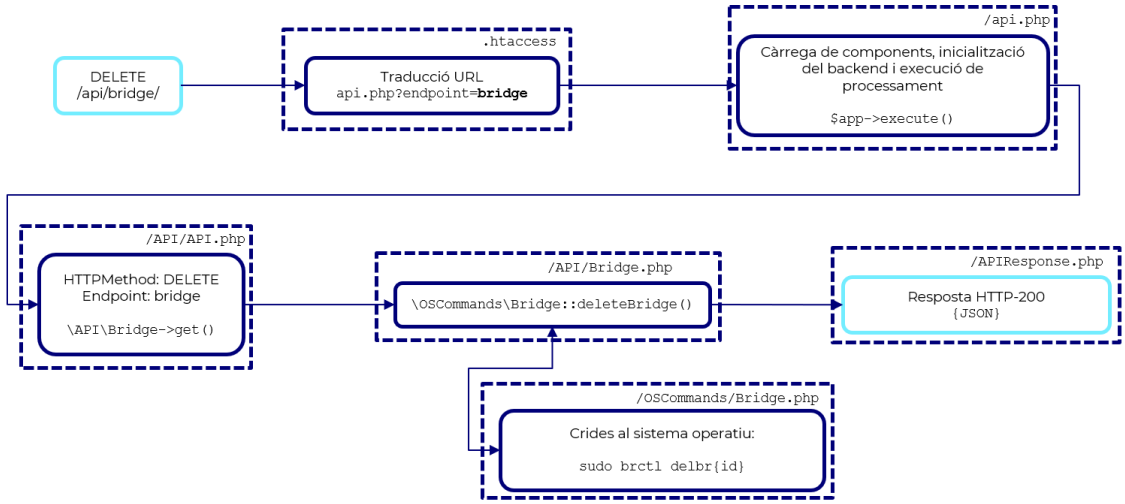
2.8. Obtenció del llistat detallat d'interfícies pont



2.9. Creació d'una nova interfície pont



2.10. Eliminació d'una interfície pont



ANNEX III – Imatges

Figura 1:

Answer delay	Answered duration	Caller lost packets	Caller max RFC3550 jitter	Called lost packets	Called max RFC3550 jitter	RTCP RTT (min/avg/max)	RTCP caller packet loss	RTCP caller jitter	RTCP called packet loss	RTCP called jitter
2.062,00ms	35.155,00ms	0/(1757+0) (0,00%)	14,70ms	93/(1664+93) (5,29%)	22,47ms	29,27/270,55/732,00ms	17,93% (315/1757)	0,00/16,86/37,00ms	18,70% (311/1663)	0,57/13,19/23,55ms
2.048,00ms	35.150,00ms	0/(1757+0) (0,00%)	13,48ms	0/(1759+0) (0,00%)	14,00ms	0,60/2,24/6,47ms	0,00% (0/1757)	0,00/13,31/26,00ms	0,00% (0/1759)	0,80/10,71/17,44ms