Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADUA: TELEKOMUNIKAZIO TEKNOLOGIEN INGENIARITZA

GRADU AMAIERAKO LANA

SOFTWARE DEFINED NETWORKING SAREETAKO SEGURTASUNA INDUSTRIA 4.0-N

Ikaslea: Atutxa Imatz, Asier

Zuzendaria (1): Huarte Arrayago, Maider

Zuzendaria (2): Jacob Taquet, Eduardo

Ikasturtea: 2017-2018

Data: Bilbo, 2018ko uztailaren 13an

Aurkibidea

Irudien aurkibidea

Taulen aurkibidea

Abstract

The aim of this project is to introduce the reader into SDN (Software Defined Networking) and to remark everything that involves cybersecurity in that area, specially regarding Industry 4.0. After showing the importance of these networks nowadays, next step is to examine the problems that can occur in services and infrastructure around it. When it comes to the "controller", which is one of the most crucial parts of these kind of systems, the goal is to try different software and to identify their suitability in terms of security. Furthermore, simulating various situations will help to document the outcome, which will define if the project is successful and the value of the process.

Resumen

El objetivo de este proyecto es introducir al lector en SDN (Software Defined Networking) y profundizar en todo lo que afecta a la ciberseguridad en esa área, especialmente en lo que respecta a Industria 4.0. Después de presentar la importancia de estas redes hoy en día, el siguiente paso será examinar diferentes problemas que puedan surgir en los servicios e infraestructuras alrededor de estas redes. En lo que respecta al controlador, que es una de las partes más importantes de estos sistemas, el objetivo es probar diferentes softwares e identificar su idoneidad en cuanto a seguridad. Además, se simularán varias situaciones reales para documentar las decisiones, lo que definirá el éxito del proyecto.

Laburpena

Proiektu honen helburua irakurlea SDN (Software Defined Networking) arloan sartzea da eta horri dagokion zibersegurtasunean sakontzea, bereziki industria 4.0-ri dagokionez. Gaur egun sare hauek duten garrantzia aurkeztu ondoren, hurrengo pausoa sare hauen inguruko zerbitzu eta azpiegituretan sor daitezkeen arazoak aztertzea izango da. Kontroladoreari dagokionez, helburua software desberdinak ikertzea, frogatzea eta segurtasunaren arloan bakoitzaren egokitasuna ikustea izango da, sistema hauetako atal garrantzitsuena baita. Gainera, egoera erreal batzuk simulatuko dira ondorioak justifikatu eta dokumentatzeko, horrek definituko baitu proiektuaren arrakasta eta prozesuaren balioa.

1. Sarrera

Telekomunikazioen arloa teknologiaren adar garrantzitsuenetako bat da, gizakiaren urruneko komunikazioaren beharra betetzen baitu. Ondorioz, aintzinatik existitzen den ideia da, baina ez da izan orain dela hamarkada batzuetara arte indarra hartu duela, izan diren ikerkuntza eta garapenei esker.

Garapen historikoan zehar bi bide nagusi ikus ditzakegu, hardware-arena eta software-arena. Lehenengoak zerikusi zuzena du elektronikarekin eta horretan izan diren garapenei esker lortu dira aurrerapauso handienak, baina hauek kontrolatzeko eta komunikazioa arautzeko erabiltzen den software-a horrekin batera joan behar da. Izan ere, lan honetan komunikazioa kontrolatzeko beharrezkoak diren protokoloak aztertuko dira sakontasunean.

Informatikan eta telekomunikazioetan, protokoloak bi entitate edo gehiagoren arteko komunikazioan informazioa transmititzea ahalbidetzen duten arauen multzoak dira. Estandarizatutako arau hauei esker edozein medio fisikotik gauzatzen den komunikazioaren sintaxia, semantika eta sinkronizazioa definitzen dira. Beste hitz batzuekin esanez, protokoloen multzoa bi konputagailuren arteko lotura fisikoaz aparte, informazioa trukatzeko sistema da.

Gaur egun, komunikazioen egoerak eta etorkizunean izango diren joerak kontuan izanik, protokoloen berrikuspen bat beharrezkoa da. Ikuspuntu hori garatuko da lan honetan zehar, lehenik testuinguruan sartuz, teknologiaren arazoak eta beharrak sakon aztertuz eta etorkizunari begira indarra hartzen doan soluzio bat aurkeztuz.

Gehiago sakonduz, lan honetan software bidez definitutako sareak (Software Defined Networking[1], SDN) aztertuko dira, batez ere hauen segurtasunaren inguruko ikerketa eginez. Sare hauek industria 4.0[2] arloan duten garrantzia eta etorkizunean izango duten erabilera hedatua ikusita, zibersegurtasuna zuzen bermatzeko tresnak ikertzea izango da garapen honen helburua. Izan ere, SDN sareak industria-lantegietako komunikazio-sareetarako oso egokiak dira hurrengo ataletan ikusiko den moduan, makina guztiak lotzeko eta sentsore eta ekipoen arteko komunikazioa zuzen bermatzeko balio dutelako.

2. Testuingurua

Teknologia aurrera doan ahala, erabiltzaileek eskatzen dituzten zerbitzuak eta hauek inplementatzeko metodoak aldatzen doaz. Sozialki onartuta dago informazioaren garaia dela, eta uneoro transmititzen diren datu fluxuak kudeatzea beharrezkoa da, *big data* kontzeptuaren atzean dauden datu baseak edozein aplikaziotan erabiltzen baitira. Mugikortasuna ere berebiziko ezaugarria bilakatu da, dispositiboak edozein lekutatik eta edozein unetan konektatu daitezkeelako sarera.

2.1. Gaur egungo sareen mugak

Oraingo sareak kontsideratzen baditugu ezinezkoa da munduko populazio osoaren beharrak asetzea, eta beraz, telekomunikazioaren mundua jadanik dauden sareak aprobetxatzen eta hobetzen doa. Etorkizun hurbil batean gertatuko den laugarren industria iraultzaren munduak, enpresa eta hornitzaileak ere soluzio berrien bila daude zerbitzuak modu eraginkorragoan eman ahal izateko. Gaur egungo sareen mugen artean hurrengoak[3] dira nabarmenenak:

· Konplexutasuna: sareak erabiltzaileen beharretara hobeto moldatzeko, protokoloak hobetu dira seguruagoak eta eraginkorragoak izateko. Hala ere, protokolo hauek arazo espezifikoak konpontzeko prestatzen dira (isolatuta definiturik), horregatik abstrakzioaren arazoa sortzen da. Beste modu batera esanda, guztiek baterako elkarlana falta da.

· Politika inkoherenteak: sare osoarentzat politika bat inplementatzeko, sareen administrariek milaka mekanismo eta gailu konfiguratu behar dituzte. Adibidez, sarera makina birtual[4] berri bat gehitzen den bakoitzean denbora luzea pasa daiteke administrariak sarbide zerrendak konfiguratu arte.

· Eskalagarritasun eza: datu baseen hazkundearekin batera, sareen gaitasuna ere handitu behar da. Sarera sartzen diren milaka ekipo berrien ondorioz, sarea konplexuagoa bilakatzen da gailu guzti horien konfigurazioaren beharragatik.

· Enpresen beharrizana: gizarteak eta hornitzaileek nahi dituzten zerbitzuak eta gaitasunak lortzeko ekipoen garapena beharrezkoa da, hauek produzitzen dituzten enpresen lana askotan besteena baino geldoago doalarik.

· Estandarizazioa: 2013an Alemanian egin zen ikerketa baten arabera[5], parte hartu zuten 278 enpresen artean gehienak arazo nagusitzat estandarizazioa eta enpresako mailen arteko “hizkuntza” komun baten beharra ikusten zuten.

· Prozesuen eta lanen kudeaketa: ikerketa horretan agertu zen beste arazo nabarmen bat kudeaketari buruzkoa izan zen, makina guztiak eta hauen prozesuak ondo monitorizatzeko eta antolatzeko beharra egonik.

2.2. Sare arkitektura berri baten beharra

Arazoak aurkeztu ondoren, orain indarra hartzen ari diren zerbitzuek eskatzen dituzten gaitasunak zeintzuk diren ikusiko ditugu:

· Trafiko patroien heterogeneotasuna: orain arte nagusi ziren bezero-zerbitzari komunikazioetaz aparte aplikazio berrienak makinen arteko informazio fluxuak sortzen dituzte, batez ere datu baseen eraginagatik. Honen ondorioz, trafikoa zein ekipo

motaren artean transmititzen den aldatu da. Gainera, orain erabiltzaileak edozein lekutatik sar daitezke sarera mugikorrei esker, trafikoaren patroietan eraginez.

· Sare administrarien lan-karga gehikuntza: administrariek erabiltzaile guztien datuak gorde eta babestu behar dituzte bezero berriak sarera gehitzen jarraitzen duten bitartean, presiopean lan egitera behartuz.

· "Hodeian" oinarritzen diren zerbitzuen hazkundea: biltegiratze fisikoak indarra galtzen joan diren bitartean, hodeiaren kontzeptua gorantz joan da. Horregatik, erabiltzaileek datuak atzitzeko sarbide azkar eta eraginkorrak eskatzen dituzte. Honek eragin zuzena du sarearen eskalagarritasunean, segurtasunean, gaitasunean eta eraginkortasunean.

· *Big data* eta banda zabalera gehiagoren beharra: datu base izugarriak agertzearen ondorioz, sareen hornitzaileei eskatzen zaien banda zabalera ere handitu da. Izan ere, banku hauek kudeatzeko milaka zerbitzariren prozesatzea behar da.

Aurreko eskakizunak kontsideratuz, SDN (Software Defined Networking)[6] motako sareak oso ondo moldatzen dira etorkizunean espero diren hobekuntzetara eta beraz, hauen inguruko hainbat garapen espero dira hurrengo urteetan. Hurrengo ataletan, sare hauen onurak azalduko dira, aurrean azaldutako arazoen soluzioa dela justifikatzeko.

3. Artearen egoera

3.1. Software Defined Networking - SDN

3.1.1. Definizioa

Software bidez definitutako sareak (ingelesez *Software Defined Networking*, SDN) sare konputazionalekin erlazionatutako teknika batzuk dira, modu determinista, dinamiko eta eskalagarri batean sare zerbitzuen inplementazioa errazagoa egitea ahalbidetzen dutenak. Hauei esker sarearen administrariak ez ditu zerbitzu horiek maila baxuan inplementatu behar, sistemaren kudeaketa maila batera arte automatizatuz.

3.1.2. Historia

SDN teknologia azkenaldiko teknologia bada ere, bere jatorria duela urte askotatik dator. Internet jaio zenean izan zuen arrakastaren ondorioz, sareen kopurua eta tamaina esponentzialki hazi zen, sare hauen programagarritasunaren beharra agertuz. Ordutik aurrera helburu horren bila ibili izan dira ikertzaileak, historia hiru etapa nabarmenetan azaldu daitekeelarik:

3.1.2.1. Sare aktiboak

90. hamarkadaren hasieran aplikazio berriak agertu ziren eta honen ondorioz ikertzaileek sare protokolo berriak diseinatu zituzten. Hala ere, protokolo horiek IETF erakundeak estandarizatu behar zituenez, oso prozesu luzea bilakatu zen.

Ikertzaile batzuek orduan beste bide batzuk hartu zituzten, sareen kontrola helburu gisa zutelarik. Aurretik zeuden sareak ez zirenez programagarriak kontrolera orientatutako sare aktiboak sortu zituzten, API-en kontzeptua bultzatuz. Honi esker, nodoen prozesamendua, biltegiratzea, pakete ilarak… kudeatzea lortu zen.

Hala ere, sareen sinpletasuna defendatzen zuen beste korronte bat zegoen ikertzaileen artean. Beraz, sare aktiboek ez zuten arrakasta totala izan, baina bai ireki zuten bidea mota honetako sareentzat etorkizunari begira.

3.1.2.2. Kontrol eta datu planoen banaketa

Hurrengo hamarkadan sarearen tamainak eta informazio fluxu kopuruak sareen bilakaera bat eskatzen zuen. Ikertzaileek ziurtasuna eta moldagarritasuna bermatzen zituzten sareak lortu behar zituzten, ordukoek ez zituztelako eskakizunak betetzen.

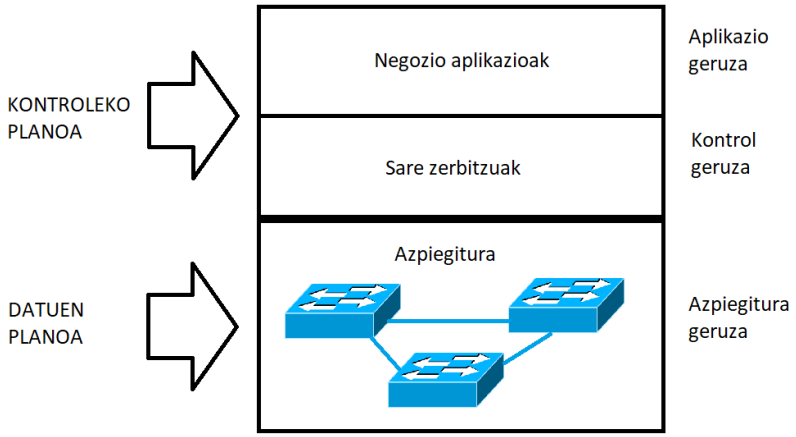
Jadanik zeuden ekipoek arazoen arazketari eta bideraketaren jokabideari aurre egin behar zioten, baina aurretik azaldu den moduan lan gogorregia zen hauentzat. Horregatik, hardware ekipoen enpresak paketeen berbidalketaren logika datu planoan inplementatzea garatzen hasi ziren eta ISP-ek[7] (Internet Service Provider) zerbitzu seguruen ikerketa (sare birtual pribatuak adibidez, VPN) landu zuten.

3.1.2.3. Openflow API-a

API honi esker, kontroladoreak goiko eta beheko geruzekin dituen interfazeak definitu eta kontrolatu daitezke. Beraz, API-a bi azpiataletan banatzen da: *northbound* eta *southbound*. Lehenengoak kontroladorearen goiko zerbitzuekiko harremanak kudeatzen ditu, egin behar dituen atazak automatizatzeko eta sarea kudeatzeko tresnak emanez. Bigarrenak ostera, kontroladorearen beheko azpiegitura osorako interfazea administratzen du, ekipo fisikoekiko loturak ezarriz eta mantenduz.

3.1.3. Arkitektura

SDN sareen muina arkitektura berezian dago, kontroleko planoa datuen planotik banatzen baita. Gainera, guztia kontroladore batekin kudeatzea lortzen da, aurreko ataletan azaldutako arazoak konponduz.



1. irudia: SDN sare arkitektura orokorra

3.1.3.1. Kontroleko planoa

Kontroleko planoa sarearen burua da, sarearen beste atal guztiak kontrolatuko dituen entitateekin osatua. Bere eginkizuna sareko funtzionalitateak automatizatzea da. Hurrengo funtzioak bete ditzake:

· Arazoak identifikatu eta konpondu

· Bideak gehitu edo ezabatu

· Segurtasun erasoak detektatu

· Seinalizazioa egin

· Baliabideak erreserbatu

· Bideraketa kudeatu

Funtzio guzti hauek modu zentralizatuan betetzen dira kontroladorearen bitartez, baina guzti honen inguruan dagokion atalean hitz egingo da.

3.1.3.2. Datuen planoa

Datuen planoa (ingelesez *data plane, forwarding plane* edo *user plane*) erabiltzaileen datuak errepresentatzen dituen entitatea, bideratze ekipo baten sarrera interfazera heltzen diren paketeekin zer egin erabakitzen duena. Modu orokorrean esanda, sartzen den pakete batentzat switch-ak helmugako helbidea zein den ikusteko begiratzen duen taulari egiten dio erreferentzia geruza honek.

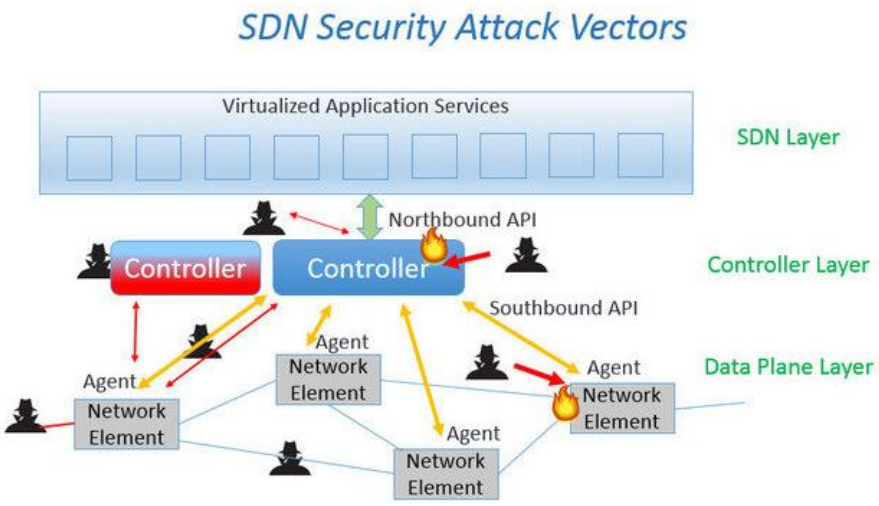
3.1.3.3. Azpiegitura

Azpiegitura ekipo fisiko guztiek eta hauek konektatzeko loturek osatzen duten multzoa da. Ekipo horiek host-etan eta beraien arteko komunikazioa ahalbidetzen duten transmisio gailuetan bereiz daiteke.

3.1.4. Segurtasuna

Orain arte SDN sareen testuingurua eta arkitektura azaldu dira, gaur egun indarra hartzen ari den teknika multzo bat delako. Hala ere, SDN sareetan alderdi garrantzitsuenetakoa segurtasuna da eta dagokion arreta jarri behar zaio.

Izan ere, ingurune zentralizatua denez (kontroladorearen ikuspuntutik) ondo prestatuta ez dauden sareetan erasoak gauzatzea erraza izan daiteke eta oso eragin txarrak izan ditzake. Horretarako, kontroladorearen segurtasuna eta bertarako sarbidea zuzen konfiguratu behar dira, edozein motako erasoak (switch-etara edo kontroladorera zuzenean) kudeatu ahal izateko.



2. irudia: SDN geruzak eta eraso bektoreak[8]

3.1.4.1. Oinarrizko ezaugarriak

Sare seguru bateko oinarrizko ezaugarriak ondorengoak dira:

· Konfidentzialtasuna

· Osotasuna

· Informazioaren eskuragarritasuna

· Autentikazioa

Bestalde, segurtasunaren arlotik babestu behar diren alderdi garrantzitsuenak hurrengoak dira:

· Datuen babesa

· Sareko aktiboen babesa

· Komunikazioen transakzioak

3.1.4.2. Eraso motak

Aurretik azaldu den bezala, bi plano nagusi daude sare mota hauetan: kontroladore planoa eta datu planoa. Gerta daitezkeen arazoak gertatzen diren geruzaren arabera saila ditzakegu ere:

3.1.4.2.1. Datu planoko erasoak

Erasotzaileak askotan sarera sarbidea lortzen dute fisikoki edo birtualki. Hau orokorrean sarera sarbide baimendua duten ekipoetara sartuz lortzen dute, jarraian sareko beste ekipoei eraso eginez. Hemen sartzen dira DoS[9] (Denial of Service) motako erasoak, sareko dispositiboen erabilgarritasuna deuseztatuz.

3.1.4.2.2. Kontrol planoko erasoak

Beste arazo mota bat erasotzaileak sarerako sarbidea lortzea izan daiteke, sareko kontroleko loturetara eraso eginez. Modu honetan, sareko switch baten kontrola hartzeak bestelako hainbat arazo sor ditzake: kontroleko loturara indar erasoak (hainbat mezu lotura apurtu arte), kontroladoreak sortu ez dituen kontrol mezuen bidaltzea edo honek benetan sortu dituenak aldatzea…

3.1.5. SDN industria 4.0-n

Ordenagailu sare tradizionalak ez daude prestatuta sare fluxuen aldaketa konstanteak jasateko eta ekipo guztien artean transmititzen diren datuak kudeatzeko. Izan ere, ezaugarri horiek IoT (Internet of Things[10]) mugimenduaren oinarria dira, eta industria 4.0-rekin oso estu lotuta daude.

Termino berri horrek Internet bidez hainbat objektu desberdinen arteko elkarloturari egiten dio erreferentzia, orain arte konektatuta ez zeudenak. Adibidez, etorkizun hurbil batean etxeko gailu guztiak egongo dira sarera lotuta, garbigailutik hasiz eta lorategiko ur-sentsoreetara arte.

Beraz, arlo honetan oso erabilgarriak izango dira SDN sareak, mugimendu berri horrek eskatzen dituen ezaugarriak eskaintzen dituztelako.

3.2. OpenFlow

OpenFlow[11] Stanford-eko unibertsitatean garatutako *switching* teknologia bat da. Software zerbitzari batek switch-en sare batean pakete batek helburura heltzeko hartu behar duen bidea kalkulatzea ahalbidetzen duen protokolo irekia da.

Sare konbentzional batean, konmutadore bakoitzak software propioa du, ekintza guztiak kudeatzen dituena. OpenFlow-rekin, paketeen transferentzien aukeraketak zentralizatu egiten dira. Horregatik, sarea eta konmutadoreak modu desberdinean eta bakarka programatu daitezke.

Konmutadore tradizionaletan, paketeen transmisioa (bideraketa) eta goi mailako bideraketa (kontrol mezuen transferentzia) gailu berdinean gertatzen dira. OpenFlow konmutadore batek, aurretik aztertu den moduan, bi maila horiek banandu egiten ditu, goi mailako trafikoaren bideraketa kontroladore banatu batek eginez.

4. Lanaren helburuak eta irismena

Lan honen helburu nagusia industria 4.0 inguruneetan software bidez definitutako sareak erabiltzean segurtasuna bermatzeko dauden tresnak aztertzea eta inplementatzea izango da, hauen eraginkortasuna konprobatuz eta frogak eginez.

Bigarren mailako helburuak inplementatu eta frogatuko diren tresnak nola erabiltzen diren dokumentatzea izango da, prozesua ahalik eta zehatzen deskribatuz.

Proiektuaren irismena frogak eta inplementazio birtualak egitea izango da, aurrerago azalduko diren tresnak eta softwareak erabiliz. Izan ere, bestelako inplementazioak eta gailu fisikoak lan honetan alde batera utziko dira, lan teorikoa izango baita.

5. Alternatiben analisia

Garapen honetan zehar industria inguruneetan hedatzen diren SDN sareen ahultasunak bilatu dira. Horretarako, *ethical hacking*-ean[12] erabiltzen diren tresnez baliatuz, hainbat eraso gauzatuko dira. Frogen egiazkotasuna eta garrantzia bermatzeko eraso arkitektura desberdinak garatuko dira, hauen puntu nagusiak hurrengoak izanik: maketaren arkitektura, erabilitako simulazio softwarea, simulatutako sarearen topologia, kontroladorearen aukeraketa eta erasoak gauzatzeko esparrua.

Hasteko, maketaren arkitekturaren garrantzia azpimarratu behar da. Simulazioak egiteko erabiltzen den sare arkitektura eta muntaketa modu desberdinetan egin daitezke, eta honen arabera emaitza desberdinak lortzeaz aparte zailtasun maila ere alda daiteke. Erabiltzen diren ekipo eta sare interfaze fisiko kopurua, hauen kokapena eta interkonexioa… guztiak zehaztu behar dira erasoa gauzatu aurretik.

Simulazio softwareak maketa prestatzea ahalbidetzen du, eta horretarako hainbat tresna erabilgarri daude: VirtualBox[13], VMware Workstation[14], Mininet[15], GNS3[16]… Bakoitzak ezaugarri desberdinak ditu, eta erabilera aldetik ere oso desberdinak izan daitezke. Beraz, egokia hautatzea oso garrantzitsua da.

Simulatutako topologiari dagokionez, hainbat aukera egon daitezke. Atal hau esanguratsua bada ere, simulatu nahi dugun sarea errealitateko sare batera egokitzea oso desberdina izan daiteke kasuan kasu. Hortaz, lan honetan erabiliko diren topologiak objektiboki aukeratu badira ere (industria inguruneetarako esanguratsuenak), beste hainbat froga egin daitezke.

Kontroladorea hautatzeak beste aparteko atal bat behar du. Izan ere, aurretik azaldu den moduan aukera asko daude, bakoitzak ezaugarri propioak izanik. Lan honetan ez dira guztiak garatuko, interesgarriak izan daitezkeen batzuk bakarrik.

Azkenik, erasotzeko softwarea zein izango den eta gauzatuko den *exploit*-a[17] zehaztu behar dira. Eraso hauen sakontasuna eta zailtasuna nahi adina altuak izan daitezke, lortu nahi den emaitzaren arabera.

Aurreko atalean azaldu diren arloetan dauden aukerak aurkeztea eta arrazoiak emanez egokienak aukeratzea izango da atal honen helburua. Erabiliko diren irizpideak lan honen helburu eta irismenarekin bat etorriko dira, aukera batzuk interesgarriak badira ere alde batera utziz (lan espezifiko eta sakonagoetarako).

5.1. Arkitektura

Lehenik eta behin, garapenean erabiliko diren simulazio arkitekturak aztertuko dira, batez ere makina fisikoen loturak eta edukiak zainduz. Distribuzioan dauden aukerak identifikatu ahal izateko, desberdindu behar diren hiru entitate bereizi behar dira: SDN sarea, kontroladorea eta erasotzeko gailua. Hauen kokapena eta interkonexioa izango dira aukerak desberdintzeko balioko dutenak.

Aukera posible bat guztia ekipo fisiko berdinean izatea da. Honen abantaila nagusia sinpletasuna da, simulatuko diren ekipoen loturak errazagoak bihurtuz. Hala ere, arkitekturaren *host*-a izango den ekipoak gaitasun handiak izan behar ditu, birtualizazioek eta simulazioek baliabide asko hartzen baitituzte. Kasu honetan bi aukera bereiz daiteke aparte: sarea eta kontroladorea ekipo birtual batean eta zibererasoen softwarea beste ekipo birtual batean egotea edo guztia berdinean izatea. Bigarrena sinpleena bada ere, ez da errealitatera asko hurbiltzen, normalena saretik kanpoko ekipo batek erasotzea baita. Ekipo birtual bitan banatzen badugu, erasoko duen ekipoa SDN sarearen parte izan daiteke ere, erasotzaile teoriko batek sareko ekipo batera sarbidea lortzea bezala. Hurrengo diagraman ikus daitezke barne-aukera posibleak:

- *Ekipo fisiko bakar batean (SDN sarea + kontroladorea + erasotzailea)*
  - *Makina birtual bakarra **(1)***
  - *Bi makina birtual*
    - *· Erasotzailea SDN sarearen parte **(2)***
    - *· Erasotzailea SDN saretik kanpo **(3)***

Beste aukera bat bi ekipo erabiltzea izango litzateke. Kasu honek baliabideen banaketa eraginkorragoa lortzen du, konfiguratzeko zailtasun handiagoak izanik. Gainera, erasotzailea eta sarea ekipo fisiko bitan banatzea errealitatera gehiago hurbiltzen da, birtualizazio maila* gutxiago izanik. Lehengo bi aukerak bereizten dira hemen ere: erasotzailea SDN sarearen parte izatea edo ez.

- Ekipo fisiko bi erabiltzea (SDN sarea + kontroladorea =|= erasotzailea)

- *Erasotzailea SDN sarearen parte **(4)***
- *Erasotzailea SDN saretik kanpo **(5)***

	1	**2**	**3**	**4**	**5**
Sinpletasuna	9	6	8	5	6
Errealitatearekin antzekotasuna	5	6	7	8	9
Baliabideen erabilpena	7	6	6	8	8

1. taula: Arkitektura desberdinen konparazioa

**Oharra:*

Birtualizazio mailak gehitzeak frogen esangura gutxitzen du, garapenaren zailtasuna handitzeaz aparte. Beraz, birtualizazio maila bi baino gehiago erabiltzea saihestuko da, konfigurazioa errazagoa izateko eta errealitatera gehiago hurbiltzeko.

5.2. Simulazio softwarea

Hainbat simulazio software daude eskuragarri sarean, bakoitzak ezaugarri desberdinak izanik. Kasu honetan hainbat aukeraketa irizpide daude, garrantzitsuenak baliabideen erabilera, informazioaren ugaritasuna eta sinpletasuna izanik.

VirtualBox software oso ezaguna da birtualizaziorako, hainbat konfigurazio desberdin onartuz. Gainera, asko erabiltzen da mundu osoan zehar, eta ondorioz, sarean dagoen informazioa eta laguntza handiak dira. Gainera, aurrerago azalduko diren beste birtualizazio softwareak baino errazagoa da konfiguratzeko, martxan jartzeko eta erabiltzeko. Hurrengo ezaugarriak[18] ditu:

- Windows, macOS, Linux eta Solaris-ekin bateragarria den tresna multi-plataforma
- Terminal bidez kontrolatu daiteke
- Makinen artean fitxategiak partekatzeko tresnak
- Makina birtualak berrezartzeko irudiak sortzeko ahalmena
- 3D grafikoentzako soporte mugatua
- VMware softwarearen makina birtualekin bateragarritasuna
- Bideoak grabatzeko tresnak
- USB 2.0 eta USB 3.0 (luzapen batekin) bateragarritasuna

VMware beste programa ezagun bat da arlo honetan, EMC Corporation enpresak diseinatutakoa. Enpresa honek banatutako softwareen artean esanguratsuena VMware Workstation da, ordaindu behar dena, baina dohaineko beste batzuk ere baditu (VMware Server, VMware Player…). Ezaugarri garrantzitsuenak hurrengoak dira:

- Host-aren eta sistema birtualaren artean fitxategiak partekatzeko erraztasuna
- Txartel adimendunen irakurleekin bateragarritasuna
- USB 3.0-rekin bateragarritasuna
- Makina birtualak berrezartzeko irudiak sortzeko ahalmena
- Makina birtualak partekatzeko tresnak
- vSphener/ESXi eta vCloud Air-ekin integrazioa
- 3D-ko grafikoak DirectX10 eta OpenGL 3.3-rekin bateragarritasuna
- “Nested virtualization”[19] teknika erabiltzeko ahalmena

Sistema birtualak simulatzeko programez aparte, sare osoak simulatzeko softwareak ere aukeratu behar dira. SDN sareen arloan garrantzitsuena *Mininet* da, milaka ikerkuntza lanetan erabili baita. Mininet-ek SDN garapen eta frogentzako ingurune birtual bat eskaintzen du, bateragarritasun maila altuak lortuz beste programa lagungarri askorekin. Hurrengo ezaugarriak ditu:

- OpenFlow aplikazioak garatzeko ingurune sinplea eskaini
- Topologia desberdinean hainbat garatzailek modu independentean lan egitea ahalbidetu
- Topologia konplexuak simulatzeko ahalmena
- CLI[20] batekin kontrolagarria
- Programazio gabeko erabiliera posiblea
- Hainbat Python API sareak sortzeko eta lantzeko

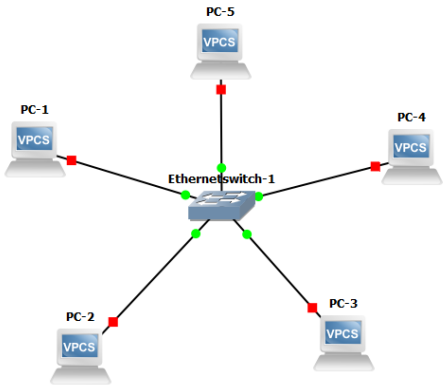
GNS3 sare topologiak diseinatzeko simuladore grafiko bat da, sareko elementu bakoitza konfiguratzeko eta ezartzeko aukera ematen duena. Topologiak modu zehatzean ezartzeko aukera ematen duenez, sare txikiak diseinatzeko oso erabilgarria da, baina eskalagarritasun baxua du (hainbat nodo eta bideragailu simulatzeko ahalmen baxua). OpenFlow-rekin bateragarria da, Open vSwitch-ak simula baitaitezke.

5.3. Simulatutako topologia

Sareak diseinatzerako orduan ekipoen topologia oso garrantzitsua da, honen araberakoa izango baita hauen arteko konexioa eta datuen transmisioa. Hainbat topologia bereiz daitezke, baina funtsean 5 dira nagusiak:

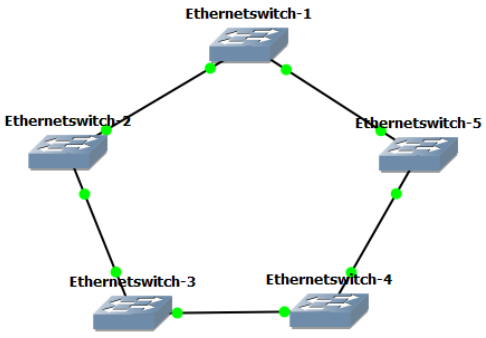
- Bus: kasu honetan nodo guztiak lotzen dira kable batera, kable hori sarearen *backbone*-a bihurtuz. Iturritik bidaltzen den seinaleak kablearen bi noranzkotan bidaiatzen du, bertara lotutako nodo guztietara helduz. SDN sare batean honetako topologia ez da oso eraginkorra, kontroladorearen lana ez delako esanguratsua.

- Izarra: topologia honetan nodo zentral bat dago, sareko beste nodo guztiak bertara lotuta egonik. Nodo zentral horrek seinale errepikatzaile modura lan egin dezake, OSI ereduan[21] duen sakontasunaren arabera seinale hori maila batera arte prozesatuz. Kasu hau interesgarria izan daiteke lan honetan frogatzeko.



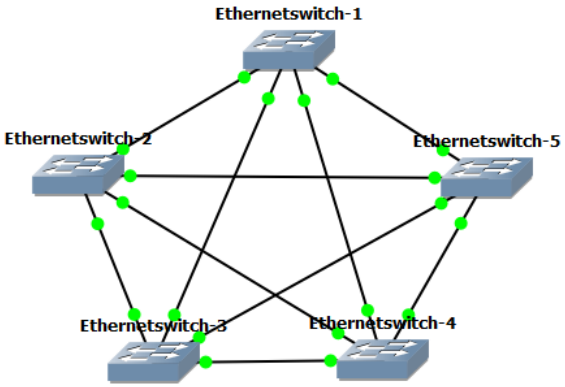
3. irudia: Izar topologiaren adibidea

- Eraztuna: nodoak bata bestearekin punturik puntu lotzean datza, azken nodoa lehenarekin lotuz. Topologia hau kasu berezietan erabiltzen da, bideraketa aldetik ez baita oso eraginkorra (eta beraz, ez da esanguratsua lan honen helbururako).



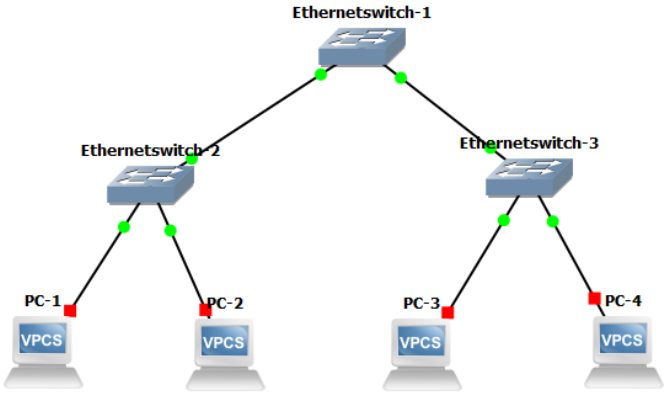
4. irudia: Eraztun topologiaren adibidea

- Guztiak guztiekin: nodo guztiak elkarrekin lotzean lortzen den topologia da. Guztiak guztiekin konektatuta egotearen abantaila bide erredundanteak izatea da, erroreen eta matxuren ondorioz datu transmisioak ez galtzeko. Lan honetan ez da sakonago ikusiko.



5. irudia: Guztiak guztiekin topologiaren adibidea

- Zuhaitza: topologia honetan nodoak punturik punturako loturekin lotzen dira, nodo bakoitza maila batean egonik. Nodo bakoitzak "gorantz" beste nodo bat du, eta "beherantz" bat edo gehiago izan ditzake. Sareak eta azpisareak sortzeko oso baliagarria da, topologia banatu bat izateko. Lan honetan topologia honen inguruan egingo da lan batez ere.



6. irudia: Zuhaitz topologiaren adibidea

5.4. Kontroladorea

Aurretik aipatu den moduan, kontroladorea sarearen atal garrantzitsuena izango da, honek kontrolatuko baitu sare osoa. Segurtasunaren arloan ere ekipo sentsibleena da, kontroladoreari eraso eginez sare osoa bertan behera gera daitekeelako. Ondorioz, zati hau landu behar da sakonen.

Lan honen garapenean, sarea kudeatzeko hainbat kontroladore desberdin aukera daitezke, funtsean antzekoak izanik baina bakoitza bere berezitasunekin.

5.4.1. NOX

SDN sareen hasieran izaniko lehen kontroladorea. Honen inguruko informazio badago sarean, lehen lan guztiak honekin egin baitziren. Hala ere, gaur egun zaharkituta dago eta garapena ez da honekin egingo (batez ere etorkizunean ez dituelako berrikuntzak izango). C++ean idatzia.

5.4.2. POX

Aukera lehenetsia, kontroladore ohikoena da. Izan ere, NOX-en ahizpa txikia eta hobetua da, honen inguruan ere informazioa dagoelarik sarean. Hau Python-en programatuta dago.

5.4.3. Beacon

Kode trinko eta adierazgarria erabiltzen duen kontroladorea, emaitza onak ematen dituena. Hala ere, izar topologiak bakarrik ahalbidetzen ditu.

5.4.4. Floodlight

Kontroladore honen berezitasuna eskalagarritasunean eta interfaze grafikoan dago. Hala ere, konfigurazio minimoarekin martxan jartzea erraza da eta oso interoperatibitate ona du OpenFlow switch birtual eta fisikoekin.

5.4.5. Ryu

Ryu osagaietara orientatutako programaziodun SDN erreferentzia-markoa da. Python-en programatuta dago eta API luze bat eskaintzen du edozein motatako sare kudeaketa edo kontroleko aplikazio gauzatzeko. Gainera, OpenFlow protokoloarekin guztiz bateragarria da bere bertsio denetan eta guztiz irekia da.

5.4.6. Faucet

Python-en oinarritutako beste kontroladore bat, OpenFlow 1.3 switchekin bateragarria dena. 2. mailako bideraketa, VLAN-ak eta 3. mailako IPv4/IPv6 bideraketa inplementatzen ditu. Instalazio erraza izateaz aparte, kontrol planorako segurtasuneko tresnak eskaintzen ditu: TLS edo 802.1AE MACSec erabiliz babesten du switch eta kontroladorearen arteko lotura.

5.4.7. ONOS

Kontroladore konplexuena, Java lengoaian programatuta dagoena. Aukera honek eskuragarritasun, sendotasun eta eskalagarritasun handiak eskaintzen ditu, interfaze grafikoa inplementatzeaz aparte.

Hurrengo orrialdean aukeraketa taula bat dago, aurretik aipatutako kontroladorez gain beste askorekin:

	Lengoaia	GUI	Dokumentazioa	Modulartasuna	Banatua/Zentralizatua	Plataformak	Produktibitatea	Southbound APIak	Northbound APIak	Bazkideak	Multithreading	Openstack
ONOS	Java	Web	Ona	Altua	B	Linux, MAC OS, Windows	Nahikoa	OF1.0, 1.3, NETCONF	REST API	ON.LAB, At&T, Ciena, Cisco, Ericsson, Fujitsu, Huawei, Intel, Nec, Nsf.Ntt Communication, Sk Telecom	Bai	Ez
Open-Day-Light	Java	Web	Oso ona	Altua	B	Linux, MAC OS, Windows	Nahikoa	OF1.0, 1.3, 1.4, NETCONF/YANG, OVSDB, PCEP, BGP/LS, LISP, SNMP	REST API	Linux Foundation beste 40 erakunde baino gehiagorekin (Cisco, IBM, NEC…)	Bai	Bai
NOX	C++	Python + QT4	Eskasa	Baxua	Z	Gehien bat Linux	Nahikoa	OF 1.0	REST API	Nicira	NOX_MT	Ez
POX	Python	Python + QT4	Eskasa	Baxua	Z	Linux, MAC OS, Windows	Altua	OF 1.0	REST API	Nicira	Ez	Ez
RYU	Python	Bai	Nahikoa	Nahikoa	Z	Gehien bat Linux	Altua	OF 1.0, 1.2, 1.3, 1.4, NETCONF, OFCONFIG	REST for Southbound	Nippo Telegraph And Telephone Corporation	Bai	Bai
Beacon	Java	Web	Nahikoa	Nahikoa	Z	Linux, MAC OS, Windows	Nahikoa	OF 1.0	REST API	Standford University	Bai	Ez
Maestro	Java	-	Eskasa	Nahikoa	Z	Linux, MAC OS, Windows	Nahikoa	OF 1.0	REST API	RICE, NSF	Bai	Ez
Flood-Light	Java	Web/Java	Ona	Nahikoa	Z	Linux, MAC OS, Windows	Nahikoa	OF 1.0, 1.3	REST API	Big Switch Networks	Bai	Ez
Iris	Java	Web	Nahikoa	Nahikoa	Z	Linux, MAC OS, Windows	Nahikoa	OF 1.0, 1.3, OVSDB	REST API	ETRI	Bai	Ez
MUL	C	Web	Nahikoa	Nahikoa	Z	Gehien bat Linux	Nahikoa	OF 1.4, 1.3, 1.0, OVSDB, OFCONFIG	REST API	Kulcloud	Bai	Bai
Runos	C++	Web	Nahikoa	Nahikoa	B	Gehien bat Linux	Nahikoa	OF 1.3	REST API	ARCCN	Bai	Ez
Lib-Fluid	C++	-	Nahikoa	Nahikoa	-	Gehien bat Linux	Nahikoa	OF 1.0, 1.3	-	ONF	Bai	Ez

2. taula: Kontroladoreen konparazioa[22]

5.5. Erasotzeko softwarea

Proiektuan erasoak gauzatzeko erabiliko den softwarea zein izango den aurkeztuko da atal honetan. Aurretik aipatutako *ethical hacking* tresna ugari daude sarean, erabiltzailearen baliabide eta ezagutzen arabera edo egin nahi den erasoaren sakontasunaren arabera. Oso ezaguna den Kali Linux[23] sistema eragilea Debian GNU/Linux-en oinarritutako sistema bat da, eta zibersegurtasun frogetara oso bideratuta dago. Izan ere, defektuz hainbat programa instalatuta ditu, arlo honetan erabiltzen direnak: Armitage, Burpsuite, Faraday IDE, Metasploit… Proiektuaren garapenean azken hau erabiliko da.

Metasploit[24] informatikaren arloan segurtasun ahulezien inguruko informazioa lortzeko kode irekiko proiektua da. Ruby lengoaian idatzita dago (lehen Perl-en) eta *pentesting*[25] deritzen sare sarbide frogetan oinarritzen da batez ere. Proiektu honen software ezagunena Metasploit Framework izenekoa da, eta urrutiko ekipoetarako *exploit*-ak sortzeko eta exekutatzeko tresna bat da.

6. Lanaren garapena

6.1 Prozesuaren deskribapena

Lanean zehar hainbat froga desberdin garatuko dira, bakoitzak aukera eta parametro desberdinak dituelarik. Horiek aukeratu ondoren, guztiak prestatu behar dira dagokien moduan. Horretarako erabilitako prozedura azalduko da orain.

Lehenik eta behin, sareak simulatzeko softwarea (Mininet) nola erabiliko den azalduko da. Mininet programaren bidez hainbat parametro desberdinekin simulatu daitezke sareak, aukera idealetatik topologia zehatz eta errealistetara. Hurrengoak dira lan honetan zehar gehien erabiliko direnak:

· "topo": Aukera honen bidez simulatuko den topologia zehazten da. Komando bidez aukera batzuk sar daitezke, topologia normalenak erabili ahal izateko (zuhaitza, lineala, sinplea…). Gainera, topologia pertsonalizatuak ere egin daitezke script bidez forma bereziak erabili ahal izateko.

· "link": Komando honek sareko nodoen arteko loturak zehazteko balio du. Ikerketa honetan loturen banda zabalera adierazi ahal izateko erabiliko da, batez ere erasoak egiten direnean konexioek jasan dezaketen pakete fluxu maximoak zehaztu ahal izateko.

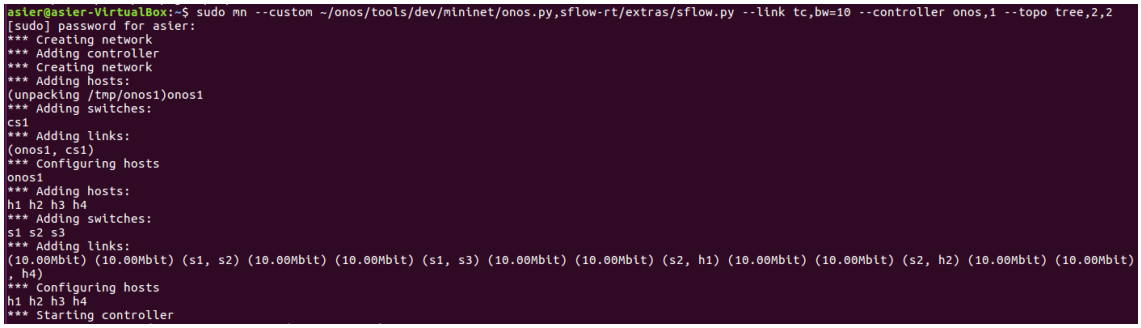
· "controller": SDN sareetan fluxuak kontrolatzeko kanpoko softwarea zehazteko aukera da hau. Gehienetan "remote" aukera sartuko da, Mininet-ek ezer ez bada adierazten POX erabiltzen duelako.

· "switch": Sareko switch ekipoak nolakoak izan behar diren adierazteko erabiltzen da. SDN sareak simulatu nahi direnez, "ovsk" izango da komando honetan adieraziko den aukera.

Eranskinetan sakonago aztertuko da software honen erabilera eta eskaintzen dituen aukera guztiak, baina amaitzeko hasieraketa adibide bi ikus daitezke hemen:

```
asier@asier-VirtualBox:~$ sudo mn --controller=remote --link tc,bw=5 --switch=ovsk --mac
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(5.00Mbit) (5.00Mbit) (h1, s1) (5.00Mbit) (5.00Mbit) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...(5.00Mbit) (5.00Mbit)
*** Starting CLI:
mininet>
```

7. irudia: Mininet-en hasieraketa: kanpoko kontroladorea, 5MB/s-ko banda zabaleradun loturak, "Open vSwitch" motako switch-ak eta ekipoen MAC helbideak sinpleak



8. irudia: Mininet-en hasieraketa: kontroladore berezia, topologia bereziarekin, zuhaitz topologia (sakonera 2, zabalera 2)…

Beste atal batekin jarraituz, aurretik esan den moduan hainbat aukera desberdin daude erasoak gauzatzeko. Garrantzituenak lan honetarako Metasploit eta hping3 izango dira.

Metasploit-ek hainbat aukera desberdin eskaintzen ditu erasoak egiteko. Izan ere, helburua eta eraso motaz aparte erasotzaile teorikoaren identitatea eta bestelako parametroak ere zehaztu daitezke. Erabilera nahi adina konplikatu daiteke, baina lan honetan egingo diren erasoak ulertzeko hurrengo metodologia azalduko da:

· Lehenik, Metasploit hasieratu ondoren, erasoak bideratzeko interfaze fisiko edo birtuala aukeratu behar da, hortik irtengo baitira eraso-paketeak.

· Ondoren, erabiliko den exploit-a hautatu behar da. Hainbat aukera desberdin eskaintzen ditu defektuz.

· Azkenik, exploit-a aukeratu ostean eraso-parametroak zehaztu behar dira: helburua zein izango den, erasoa gauzatuko duen erasotzailearen IP helbidea zein izango den (ausazkoa izan daiteke), bestelako parametroak (paketeen kantitatea, fluxuaren eraso denbora…),…

Hping3 ordea askoz sinpleagoa da, baina eraginkorra izaten jarraituz. Beraz, froga gehienetan hau erabiliko da. Tresna honekin flooding erasoak gauzatu daitezke komando bakar baten bidez, parametro gehiarriak adierazi daitezkeelarik (erabiliko den protokoloa adibidez).

Kontroladorea, hasieran aipatu den moduan, SDN sareen atal garrantzitsuenetarikoa da, hau baita datu-fluxuak kudeatu, erasoak detektatu eta eten behar dituen elementua. Beraz, honen konfigurazioa zuzena izan behar da sistema osoaren funtzionamendu egokia bermatzeko. Lehen adierazi den moduan, hainbat aukera desberdin daude kontroladorea inplementatzeko, eta bakoitzarekin prozesu desberdinak jarraitu behar dira parametroak zehazteko.

ONOS kontroladorea hasieratzeko errazenetakoa da, komando bakar baten bidez egiten baita. Gainera, CLI (komando bidezko interfazea) eskaintzen du jarraian, parametroak aldatu ahal izateko edota datu-fluxuak kontrolatu ahal izateko. Web bidezko interfaze grafikoa ere eskaintzen du, uneoroko trafikoaren informazioa ikusi ahal izateko. Printzipioz ez du behar konfigurazio berezirik sare normal bat martxan jartzeko, baina lan honetan aztertuko den segurtasuna automatizatzeko ez duenez tresnarik ematen, kanpoko software bereziak behar dira.

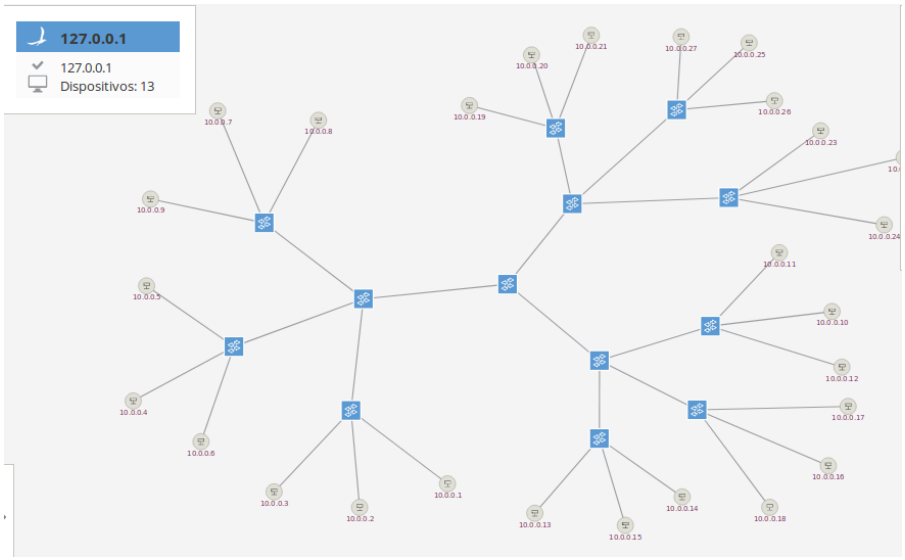
RYU kontroladorea (lan honetan aztertuko den bestea) konplexuagoa da, kontroladoreak script-etan idazten direlako eta normalean kasu bakoitzerako bereziki prestatzen direlako. Horrez gain, script abiarazle bat du ("ryu-manager" izenekoa) eta horrekin nahi den kontroladorea hasten da. Aurrerago azalduko den moduan, segurtasuna inplementatzeko mekanismoak kontroladorean bertan inplementatzen dira, baina horretarako topologiaren araberako kontrol bat zehaztu behar dugu, switch-en eta portuen arteko mapaketa adieraziz. Hurrengo harrapaketan ikus daiteke horren adibide bat:

```
# Sustained no attack count for switch/port combinations
self.noAttackCounts = {"s1":  [0] * 3,
                       "s11": [0] * 3,
                       "s12": [0] * 3,
                       "s21": [0] * 3,
                       "s22": [0] * 3,
                       "s2":  [0] * 3}

# Mapping from switch/port/destination MAC combinations to flow rates
self.rates = {"s1": [{}, {}, {}],
              "s11": [{}, {}, {}],
              "s12": [{}, {}, {}],
              "s2": [{}, {}, {}],
              "s21": [{}, {}, {}],
              "s22": [{}, {}, {}]}

# Mapping from switches and ports to
# attached switchtes/hosts
self.portMaps = {"s1": ["s11", "s12", "s2"],
                 "s11": ["AAh1", "AAh2", "s1"],
                 "s12": ["ABh1", "ABh2", "s1"],
                 "s21": ["BAh1", "BAh2", "s2"],
                 "s22": ["BBh1", "BBh2", "s2"],
                 "s2": ["s21", "s22", "s1"]}

# Mapping from datapath ids to switch names
self.dpids = {0x1: "s1",
        0xb: "s11",
        0xc: "s12",
        0x2: "s2",
        0x15: "s21",
        0x16: "s22"}
```

9. irudia: RYU kontroladorearen script-ean switch eta portuen mapaketa

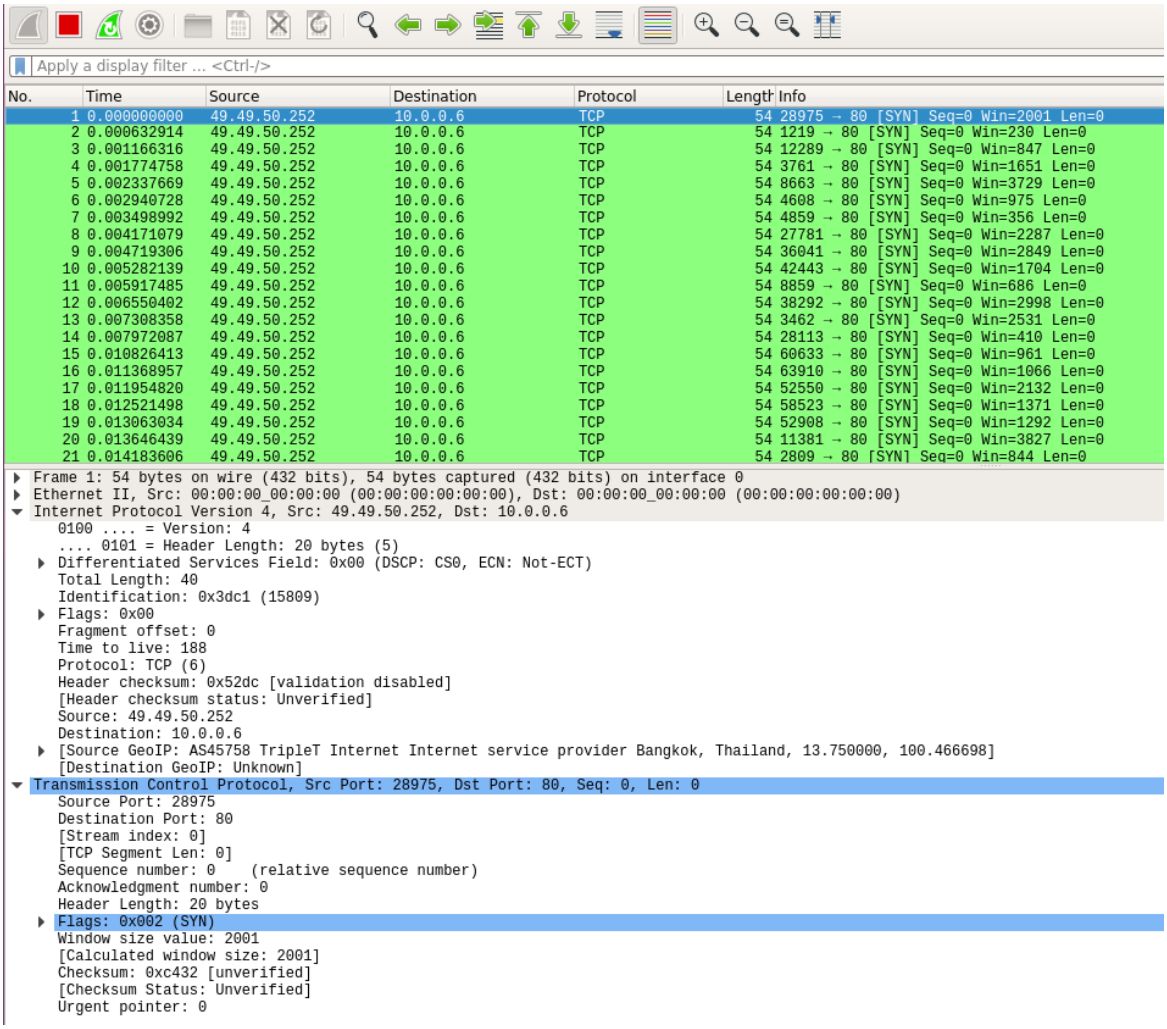
6.1. Lehen froga

Proiektu honen lehen froga egiteko sistema sinple bat prestatu da, aurreko aukeren artean sinple baina esanguratsuak hautatuz. Beraz, azaldutako arkitektura sinpleena erabili da, guztia ekipo fisiko eta makina birtual berean simulatuz (SDN sarea, kontroladorea eta erasotzailea). Gainera, VitualBox birtualizazio softwarea erabili da makina birtual hori martxan jartzeko, Ubuntu 16 sistema eragilearekin. Sistema eragile hori aukeratu da zerbitzarietarako Debian-en oinarritutakoak asko erabiltzen direlako, eta kasu honetan kontroladorea simulatzeko erabili da. Mininet softwarea erabiliz hiru mailako zuhaitz topologia bat simulatu da ONOS kontroladorearekin, nodo nahiko dituelako baina oraindik sinplea izaten jarraitzen duelako. Atakeak egiteko softwarea Metasploit Framework izan da, kontroladorearen kontrako TCP SYN mezuen *flooding*-a gauzatuz. Hurrengo harrapaketan ikus daiteke erabiliko den topologiaren grafikoa:



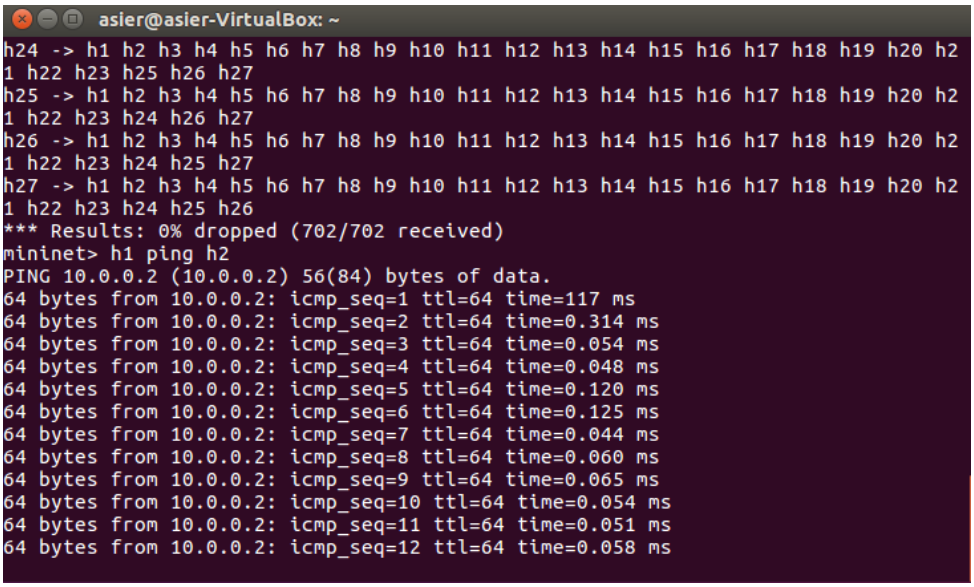
10. irudia: Lehen frogaren topologia

Aurrean azaldutako prozesuan orduan SYN flooding erasoa gauzati da. “enp0s9” interfazea (kontroladorea sare lokalera ez baina Internet-era lotzen duen interfazea) hurbilagotik behatzen bada, jasotzen diren mezuak TCP hasieraketa elkarrizketa batean egiten diren SYN motakoak direla ikus daiteke:



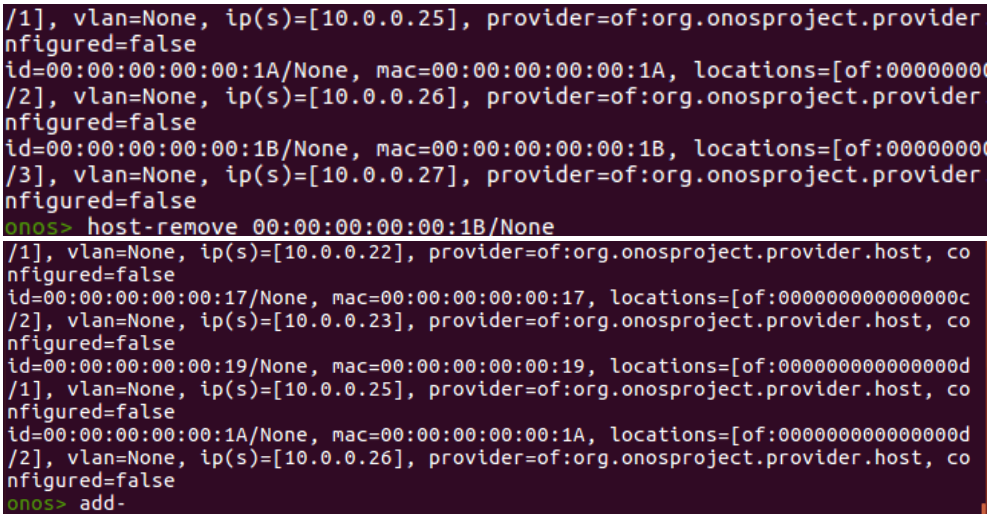
11. irudia: Erasotako interfazea Wireshark-en

Ikusten denez, mezuen iturri teorikoak 49.49.50.252 helbidea du, baina errealitatean IP hori Metasploit-ek ausaz asmatu du. Erasoa martxan badago ere, ping-ak egitea oraindik posiblea da:



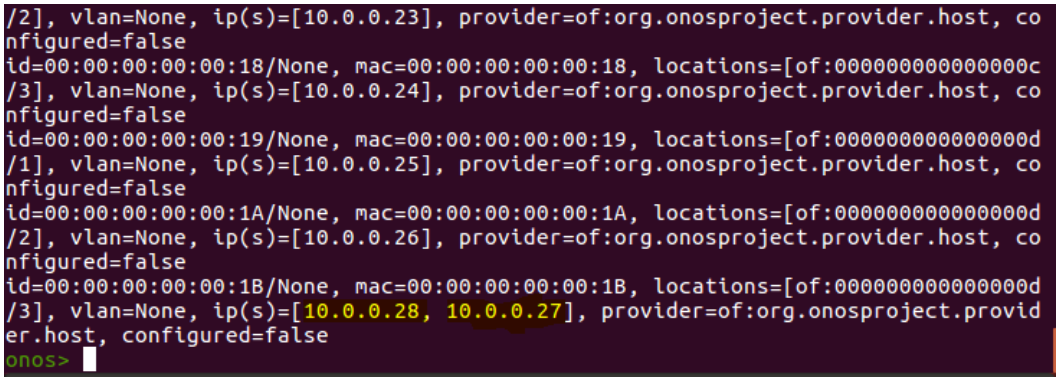
12. irudia: Ping-en konprobazioa

Hala ere, konektibitatea ondo dago datu-fluxuak jadanik ezarrita daudelako, baina berriak sortu edo lehengoak aldatu daitezkeen ikusi behar da. Horretarako, ONOSen terminaletik *host* bat ezabatuko da tauletatik, *remove-host* komandoarekin.



13. irudia: Host bat kontroladorearen tauletatik ezabatzea

Berriro guztien arteko ping-ak egiten badira, kontroladoreak *host* hori berriro detektatu beharko luke, tauletan berrezarriz. Gainera, eskuz *host* horri IP helbidea aldatzen bazaio, hori ere detektatu eta gorde beharko luke. Erasoa martxan dagoen bitartean baliteke behar bezala lan ez egitea, baina SYN mezuen eraso batekin ez da ezer arrarorik gertatu:



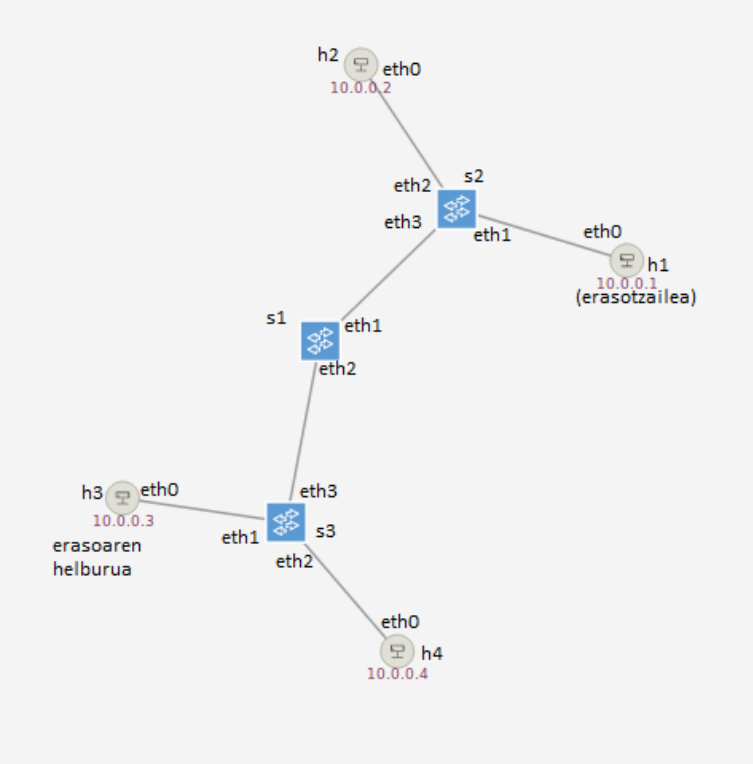
14. irudia: Host-a berrezartzea kontroladorearen tauletan

6.2. Bigarren froga

Kasu honetan, aurreko frogaren antzeko arkitektura erabiliko da, ohikoena delako ikerketa hauetan. Hala ere, oraingoan erasoak Mininet tresnarekin sortutako sareko ekipo batetik egingo dira, balizko erasotzaile batek sarbidea lortuko balu bezala. Modu honetan, industria 4.0 inguruneetan sor daitekeen egoera bat simulatuko dugu, sare pribatu bateko ekipo bat kutsatuko balitz bezala. Kasu hauetan, ekipo horretatik erasoak sare barruko beste nodo batzuetara bideratzen dira, sarea barrutik "apurtu" ahal izateko. Beraz, froga honetan garatuko den tresna baliagarria izango da egoera hauetarako.

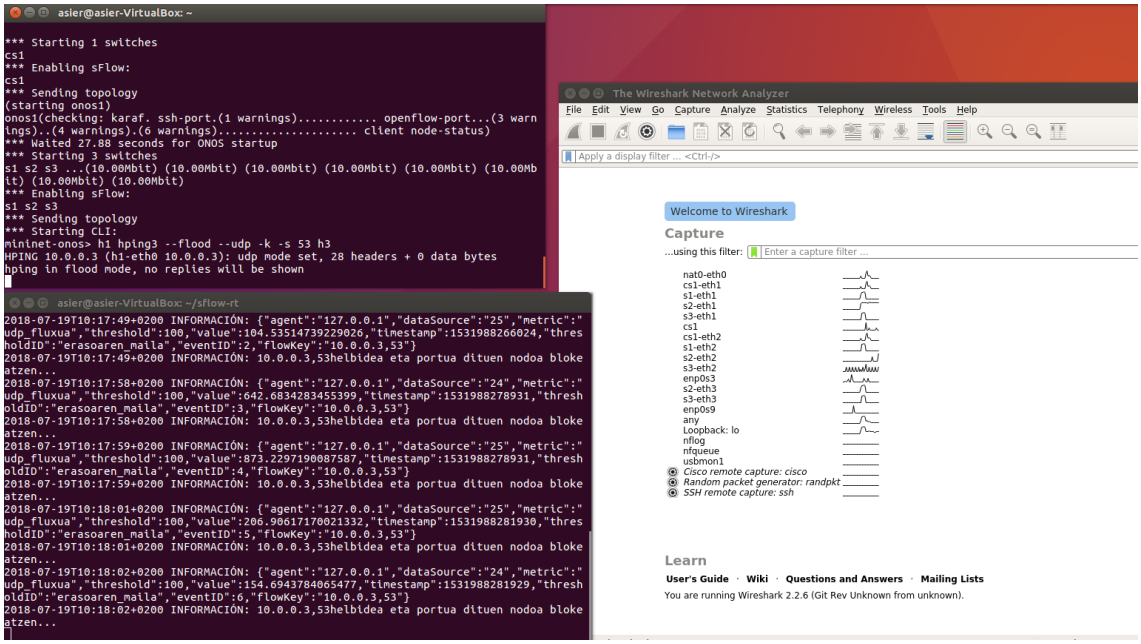
Hping3[26] softwarea erabiliko da, protokolo bidezko erasoak[27] gauzatzeko asko erabiltzen den tresna baita. Sare topologia aurrekoa bezala zuhaitz motakoa izango da, baina kasu honetan sinpleagoa (sakonera 2 eta zabalera ere 2). Erabili den kontroladorea ONOS izan da, baina oraingoan sFlow-RT[28] izeneko OpenFlow fluxu kontroladore bat ere erabili da, ONOS-ekin batera script bat abiaraziz. Fluxu kontroladore horrekin sarean trukatutako datuen analisiak eta kudeaketak gauzatu daitezke uneoro, informazio guztia web interfaze batean izanik. Script horrekin (antiflooding.js) garatzaileak adierazitako *threshold*[29] batetik gorako trafikoa duten fluxuak detektatu eta blokeatu egin daitezke, DDoS motako erasoak mitigatuz.

Hurrengo irudian ikus daiteke erabili den sarea nolakoa den:



15. irudia: Bigarren frogaren topologiaren irudia

Jarraian, hping3-rekin eraso bat gauzatu da h1 eta h3 host-en artean, sare osoa gurutzatzen duena. Ikusten denez, erasoa segituan detektatu du sFlow-ek, iturria blokeatuz.

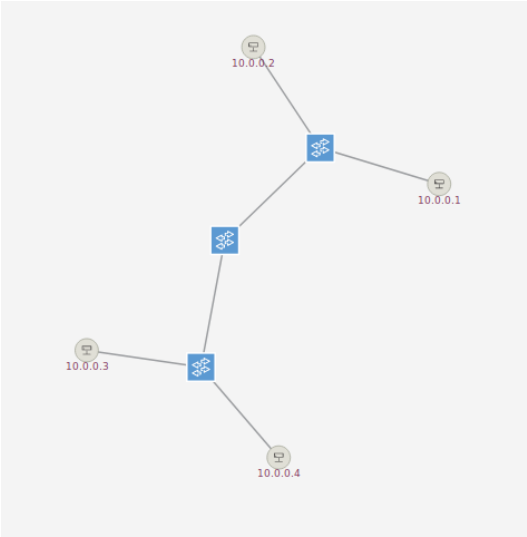


16. irudia: Erasoa hasi eta blokeatu

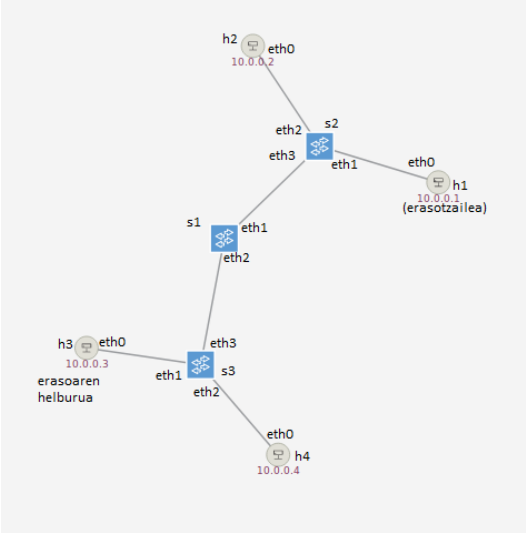
sFlow-ren interfazean badirudi erasoa aurrera doala, baina hor interfaze guztiak agertzen dira eta beraz, h1 eta s2 artekoa ere. Hori ezin da blokeatu, h1-ek berak sortzen duelako eta s2 switch-ak blokeatu. Beraz, Wireshark trafiko arakatzailea erabiliz, beste loturetan erasoa blokeatu dela ikus daiteke. Hor ondo ikusten da s2-eth1 interfazean (s2 switch-a eta h1 host-aren artekoa) erasoak jarraitzen duela, aurretik esan den moduan hori ezin delako blokeatu. Hala ere, h3-ranzko beste interfazeetan erasoa hasi dela baina blokeatu egin dela ikus daiteke, adibidez s1-eth1 interfazean (s1 eta s2 switch-en artekoa).

17. irudia: Erasoa h1 ekipoaren interfazean soilik martxan, beste interfazeetan blokeatuta (s2 switch-etik aurrera)

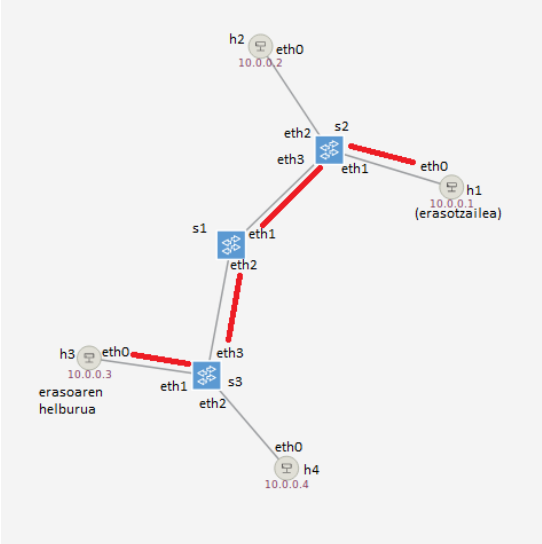
Laburbilduz, hurrengo grafikoetan ikus daiteke erasoaren eta honen blokeoaren eboluzioa, aurretik aipatutako tresnei esker:



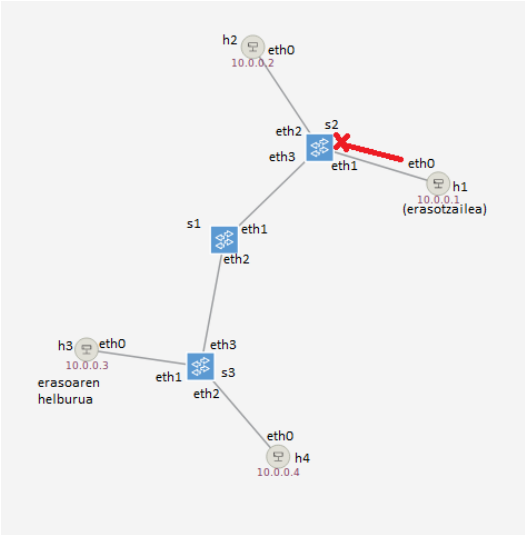
18. irudia: Sarearen zuhaitz topologia



19. irudia: Sareko ekipo eta interfazeak



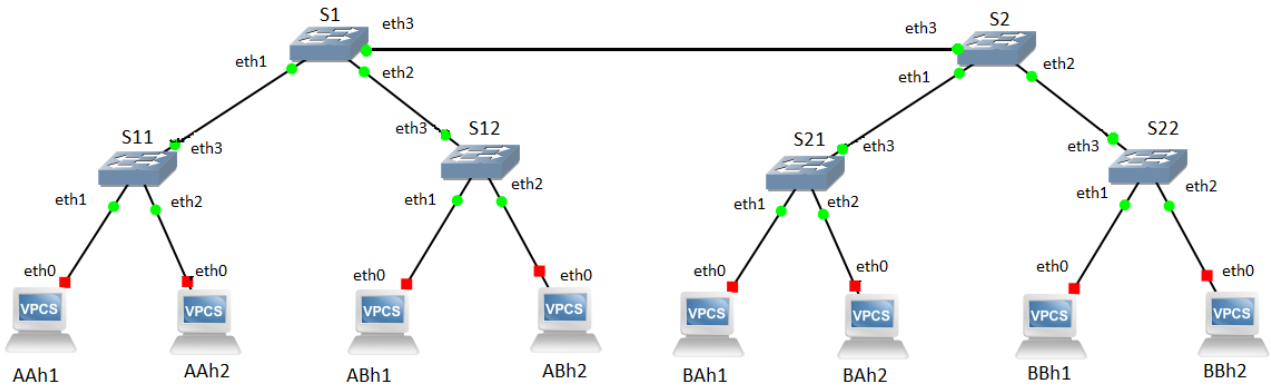
20. irudia: Erasoa sarean



21. irudia: Erasoa blokeatuta, h1-etik s1-erarte soilik

6.3. Hirugarren froga

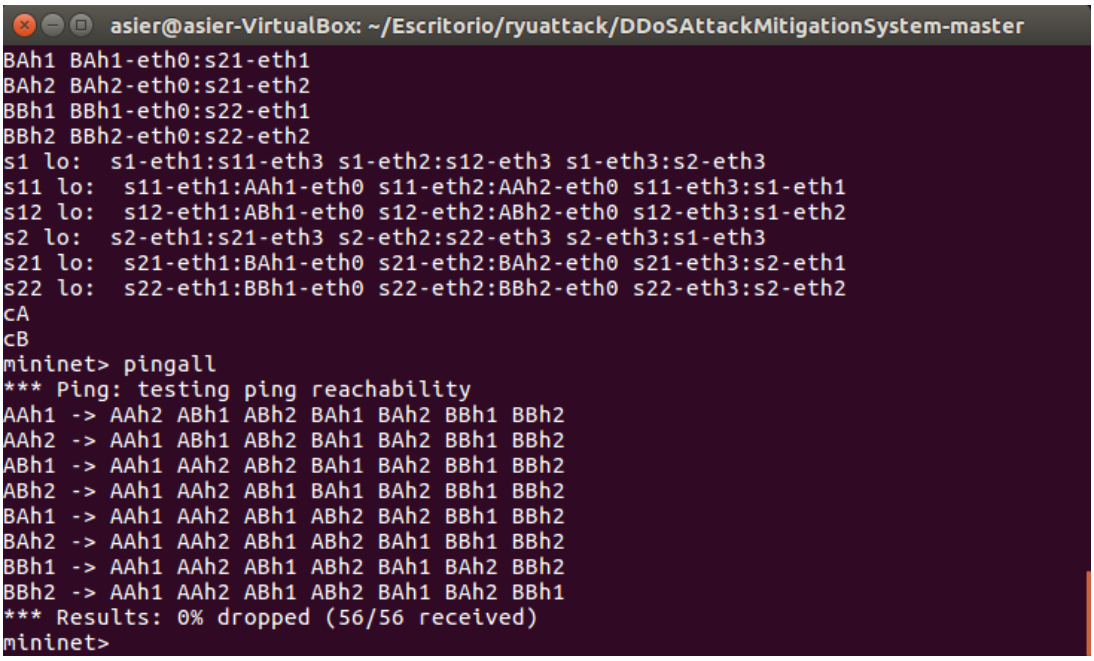
Froga honetan arkitektura berdina erabiliko da, baina beste atalak aldatu egingo dira. Izan ere, kasu honetan Python-en programatutako RYU kontroladorea erabiliko da, hainbat aukera ematen baititu. Gainera, topologia konplexuagoa erabiliko da, antzeko host kopurua erabiliz baina bi domeinutan banatuz (bakoitzak kontroladore bat izanik). Hurrengo harrapaketan ikus daiteke grafikoki:



22. irudia: Hirugarren frogaren topologiaren irudia

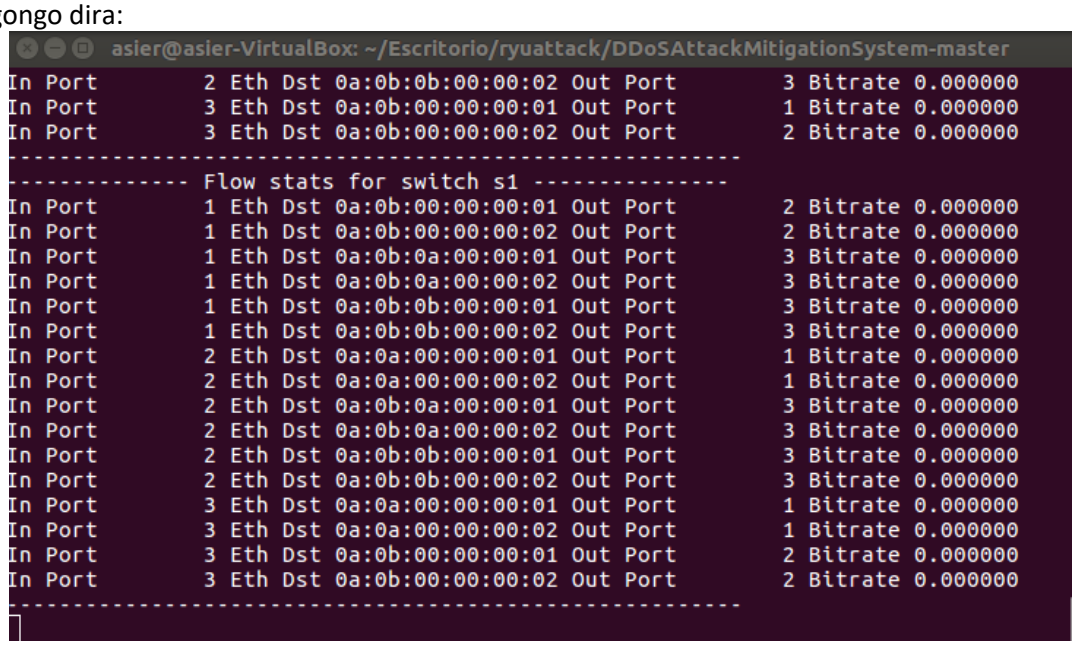
Erasotzeko sistema berdina izango da, sareko host batetik hping3-rekin *flooding* bat gauzatuz. Kasu honetan ordea, erasoaren aurkako tresna RYU kontroladoreak inplementatuko du, Python-en idatzitako script bati esker (eranskinetan kodea). Eraso bat gertatzen ari dela ikusten badu, kontroladoreak bere domeinuko ekipo bat bada erasotzailea domeinura konektatzen duen loturan politika bat ezarriko du, transmisioak ebakiz. Beste domeinukoa bada, beste kontroladoreari abisatuko dio, honek bertan berdina eginez. Ezarritako politika 10 segundora kenduko da, berriro lotura abian jarriz (erasoa gelditu den ikusteko). Benetako inplementazio batean, berrabiarazte denbora hori luzeagoa izan daiteke, kasu honetan froga hobeto ikusteko egiten delarik.

Lehenik kontroladore biak eta sarea hasieratuko dira, simulatutako host-en arteko konektibitatea bermatuz:

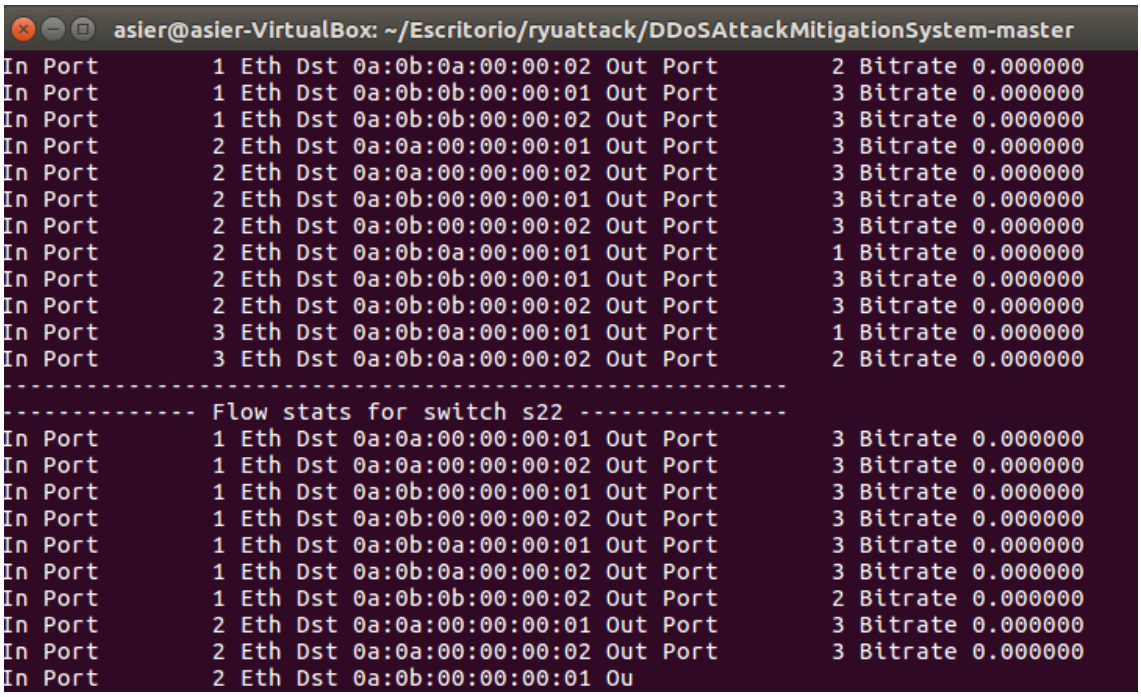


23. irudia: Sarea hasieratu eta konektibitatea bermatu

Ikusten denez, ping guztiak ondo heldu dira, eta beraz kontroladoreetan fluxuak gordeta egongo dira:

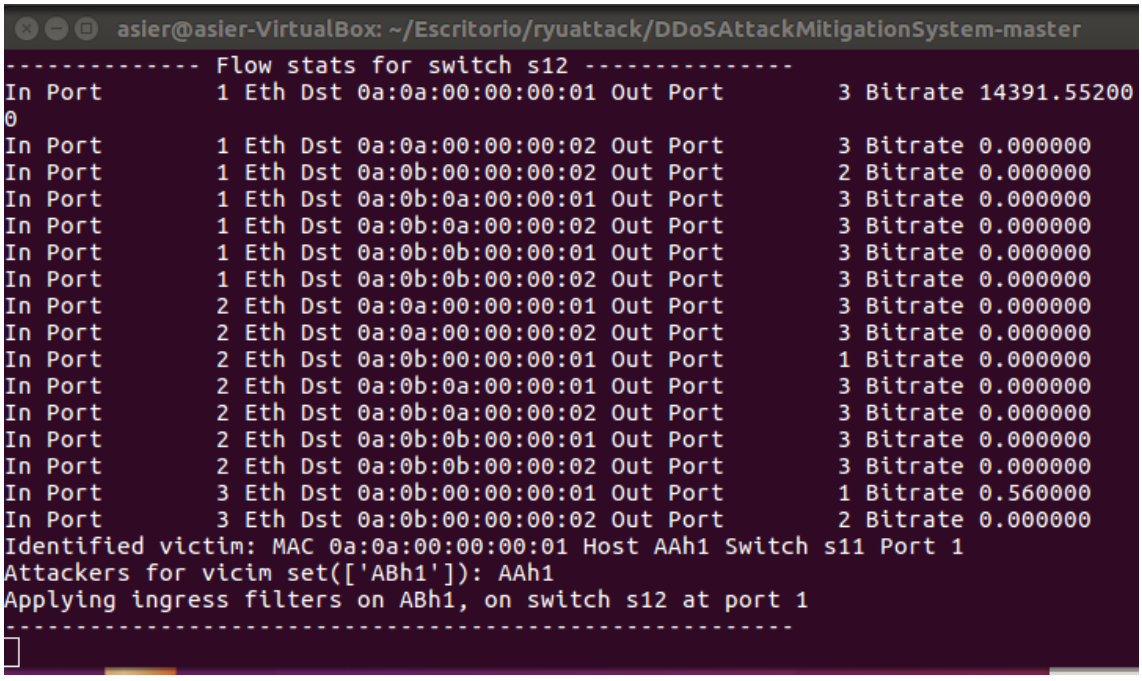


24. irudia: A domeinuko kontroladorea

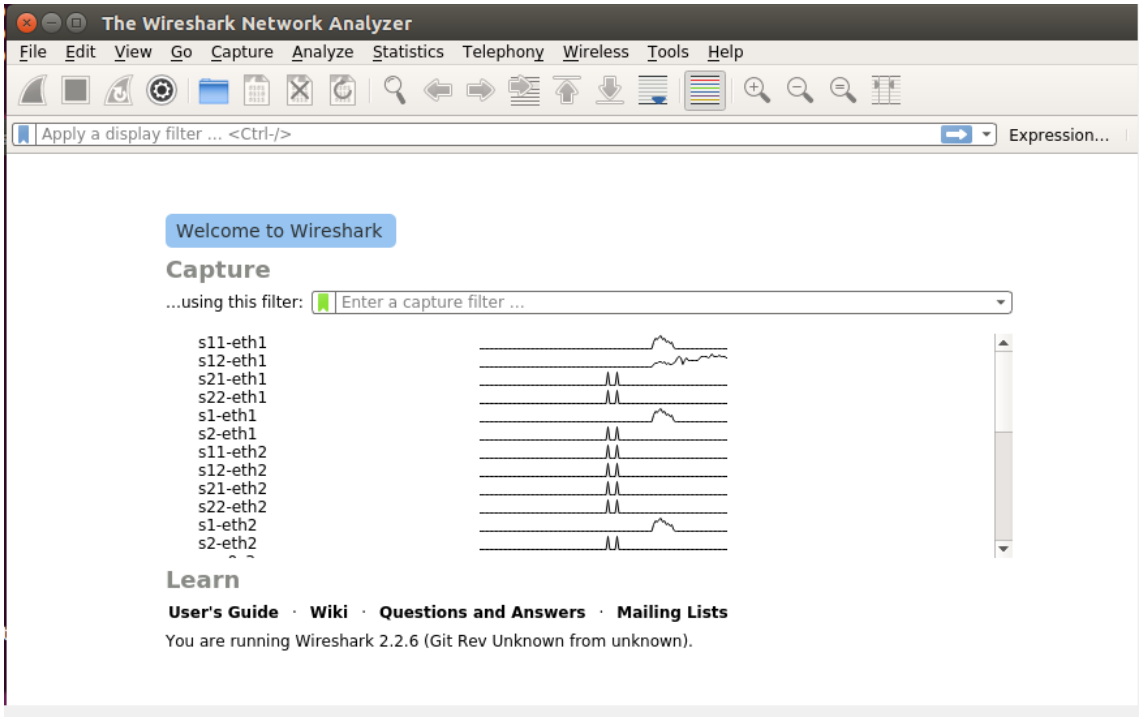


25. irudia: B domeinuko kontroladorea

Erasoa gauzatzen denean, aurretik esan denez bere domeinuko kontroladoreak detektatu egingo ditu (erasotzailea eta biktima). Jarraian, erasotzen ari den host-a blokeatu egingo du, Wireshark trafiko arakatzailean ikusten den moduan:



26. irudia: A domeinuko kontroladoreak erasoa detektatu eta blokeatu



27. irudia: Erasoa lotura guztietan blokeatuta (erasotzailearenan izan ezik)

Beraz, tresna hau oso egokia dela ikus daiteke, erasoen aurkako segurtasun trensa kontroladorean bertan inplementatzen delako. Honi esker, ez dira sarearen kontrol softwareak modulutan banatu behar, guztia zentralizatuta geratuz (bideraketa eta trafiko kudeaketaz aparte, segurtasuna ere bai).

Segurtasuna kontroladorean bertan (RYU-ko python fitxategietan) inplementatzeak kontroladorea sarera guztiz molda daitekeela esan nahi du, defektuz dakarren kontroladorea oinarritzat hartuz baina beharrezkoak diren konfigurazioak eginez.

7. Ondorioak

Lan honen heburu nagusia industria 4.0 inguruneetan software bidez definitutako sareak erabiltzean segurtasuna bermatzeko dauden tresnak aztertzea eta inplementatzea izan da, eta horretarako eskuragarri dauden tresna batzuk frogatzeaz aparte, kasu konkretu baterako froga bat ere prestatu da (bigarren froga).

Ikusi denez, hainbat segurtasun mekanismo desberdin erabil daitezke SDN sareen osotasuna bermatzeko, nahi diren konfigurazio aukeren arabera hautatu baitaitezke. Alde batetik, RYU kontroladorea python lengoaian programatuta dago, eta kontroladorearen kodean bertan inplementatu daiteke segurtasuna. Ondorioz, hainbat aukera ematen dituen lengoaia batean programatzeak onura handiak ekar ditzake, eta gainera kontroladorea eta segurtasuna modulu berean inplementatzeak zentralizazioa inplikatzen du (eta horrek kudeatzeko erraztasuna). Hala ere, topologiaren arabera programazioa desberdina izan behar da, switch eta nodo guztiak “erregistratu” eta kudeatu behar direlako kodean. Beraz, kasuan kasu kontroladore desberdina abiarazi behar da eta topologia aldaketa asko jasango dituen sare baterako ez da guztiz erabilgarria.

Bestalde, ONOS kontroladorea eta Sflow-rt trafiko kudeatzailea ere erabili dira. Hauen berezitasuna sarearen ezaugarriak moduluka banatzean datza, nodo hauetarako erasoren bat egotekotan erraz kontrola daitekelarik. Horretaz gain, eskalagarritasun hobea ere lortzen da inplementazio honekin, alde batetik ONOS konfiguratzen delako eta Sflow bestetik. Gainera, segurtasun script-ak ez du topologiarekin zerikusirik, eta tresna berdina erabil daiteke kasu bakoitzerako.

Ezaugarri nagusietaz gain, hurrengo taularekin informazio sakonagoa azter daiteke:

	ONOS + Sflow-rt	**RYU**
Erantzun denbora	~3,35 s	~ 2,21 s
*Fluxu kopuru maximoa**	4000	8000
*Latentzia egoera kritikoan**	0,4 ms	0,22 – 0,3 ms
Eskalagarritasuna	Handia	Baxua
Interfaze grafikoa	Bai (web)	Ez
Interfazearen erabilera	Erraza	Zaila
Topologiaren araberako dependentzia	Ez	Bai

3. taula: Erabilitako tresnen konparazioa

** Datu teorikoak*

Errendimenduari dagokionez, taulan erasoarekiko erantzun denbora agertzen da, eta ikusten denez RYU sendoagoa da arlo horretan. Sakonago aztertuz, hainbat ikerketetan ikusi den moduan[30], abiadura eta fluxu-kopuru handienak RYU, ONOS eta OpenDayLight kontroladoreek eskaintzen dituzte. Beraz, segurtasuna modu eraginkor batean inplementatu ahal izateaz aparte, lan honetan frogatu diren kontroladore biak oso gaitasun altuak erakusten dituzte atal horretan. Hala ere, bien artean RYU-k ezaugarri hobeak ditu orokorrean.

Ondorioz, tresna biak oso erabilgarriak dira eta erantzun denbora zuzenak dituzte. Antzeko aukerak ematen badituzte ere, bakoitza helburu zehatz baterako baliagarria da: RYU kontroladoreak sinpletasuna eta zentraltasuna eskaintzen ditu, sare txiki eta ez oso aldakor batean erasoen aurkako defentsa bat inplementatu ahal izateko; ONOS kontroladoreak, Sflow-rt-ren laguntzarekin, modulartasuna eta interfaze grafikoen erabileraz gain eskalagarritasun handia ematen ditu, tamaina handiko enpresa batek behar duen aldagarritasuna hain zuzen. Beraz, kasu hau oso erabilgarria da industria 4.0 arloan, segurtasun mekanismo automatizatuak inplementatzea guztiz beharrezkoa baita.

8. Eranskinak

8.1. Lan ingurunea - Mininet

Lan honetan Mininet programarekin abiatzea aukeratu da. Izan ere, honek hainbat aukera ematen ditu topologia karakterizatzeko eta ekipoen zehaztasunak erabakitzeko. Gainera, OpenFlow protokoloarekin oso lotuta dago eta informazio asko dago sarean honi buruz (ikerketa lanak, tutorialak, foroak…). Arazo bakarra interfaze grafikoaren gabezia da (*MiniEdit* izeneko tresna gehigarri bat badago ere), baina topologiak terminaletik sortu ondoren grafikoak editore batekin egin daitezke beharrezko trafikoa ilustratzeko.

Mininet-ekin hasteko web orrialde ofizialean sartuz, bertan eskuragarri dagoen makina birtual bat deskarga daiteke. Bertan Ubuntu Server 14.04 LTS bat dago instalatuta, Mininet eta beste hainbat aplikazio erabilgarri dituelarik. Zerbitzari bertsioa izatean ez du interfaze grafikorik (deskargatu eta instalatu daiteke), baina hori ez da arazoa izango Mininet-ek ere ez duelako eta makina birtualera *ssh* bidez konektatu daitekeelako host-etik.

Arazoak:

Atal honetan arazo batzuk sor daitezke. Mininet erabiltzean Xterm bidez ekipoen terminalak simula daitezke, baina programa hori erabiltzeko interfaze grafikoa behar da (beste leiho batean agertzen delako terminal hori, aparte kudeatu ahal izateko). Horretarako, aurretik aipatu den bezala host-etik ssh lotura bat behar da, eta horretarako lehenik VirtualBox-eko sare birtual bat sortu behar da. Izan ere, hori barik ezinezkoa da ekipo berdineko makinen artean konektatzea, sare txartelak eta interfazeak ez daudelako lotuta.

8.1.1. Mininet sarrera

Lehenik, software honen sarrera egingo da, hasieraketa eta oinarrizko komandoak aurkeztuz.

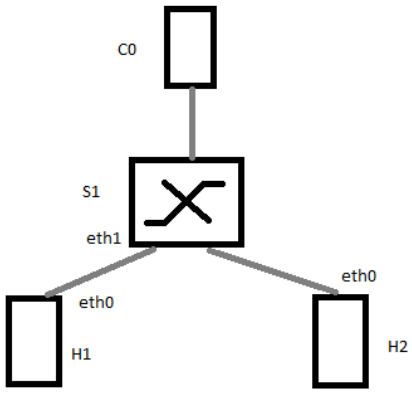
Konexio ezarpena

Aurretik esan bezala, makina birtuala hastean *ifconfig* komandoarekin bere interfazeak ikusiko dira. Gure arreta *eth0* interfazean jarriko dugu, hau baita lotura egiteko balio duena. Aipagarria da beste interfaze batzuk ere ikus daitezkeela: eth0, lo eta ovs-system. *eth0* eta *eth1* ordenagailuaren sare txartel fisikoen interfazeei dagokie eta lo *loopback* interfazeari. Ovs-system ostera, VirtualBox-ean sortutako sare birtualari dagokio.

Orduan, *eth0* interfazearen IP helbidea ikusi eta host-eko terminaletik ssh lotura bat ezarriko da bertara hurrengo komandoarekin: *ssh -X mininet@ipaddress*. "-X" aukera geroago Xterm terminalaren display-a atera ahal izateko da. Jarraian makina birtualaren login eta

pasahitza eskatuko ditu lotura gauzatzeko (web orritik deskargatutako makina birtualaren kasuan biak dira “mininet”).

Konexio ezarpenarekin amaitzeko, terminal berri honetan *sudo mn* komandoa erabiliko da Mininet softwarea hasieratzeko (oinarrizko topologiarekin) eta honen komando lerroa agerrarazteko.



28. irudia: Oinarrizko topologia

Aipagarria da lan honetan agertzen ez diren froga batzuetarako konexio ezarpen metodo hau erabili dela, baina beste kasu guztietan ez da prestatuta dagoen makina birtuala erabili. Horren abantailak bistakoak dira, interfaze grafikoa izateaz aparte (Ubuntu 16.04 Desktop) konfigurazio aukera hobeak eta errazagoak lortzen direlako. Hala ere, software honekin hasteko makina birtuala erabil daiteke.

Oinarrizko komandoak

Orain software honek eskaintzen dituen komando erraz batzuk aipatuko dira, azalpen labur bat esleituz bakoitzari:

help: Beste hainbat testuingurutan bezala, kasu honetan ere komando honi esker Mininet-i buruzko laguntza lor daiteke. Lerro hori sartzean lortzen diren emaitzak aurrerago azalduko diren komandoak agerrarazten ditu. Gainera, nodoei komandoak bidaltzeko nola egin behar den ere erakusten du (*<node> command {args}*). Hala ere, *xterm h1* erabiltzera gonbidatzen gaitu, aurretik aipatu den bezala nodo bakoitzaren terminalak simulatzeko eta sistema modularizatzeko.

nodes: Honi esker, uneko topologian dauden ekipo/host, switch eta kontroladore guztiak aurkezten ditu terminalak. Oso erabilgarria da hasieraketa ostean topologia zuzen sortu bada konprobatzeko.

dump: Komando honek topologiari buruzko informazio gehigarria ematen du. Bertan, ekipo guztien interfazeak, IP helbideak, pid-ak… lor daitezke.

net: Ekipoen arteko loturak ikusteko komandoa. Ekipo bakoitzaren interfaze bakoitza zein ekipo/interfazerekin konektatuta dagoen erakusten du, oso erabilgarria hau ere.

ping: Komando tipiko hau testuinguru honetan ere badago, ekipoen arteko loturen egoera ikusteko baliagarria izanik. Beraz, ekipo batek beste bati Echo Request motako ICMP mezu bat bidaltzea simulatzen du.

> *Oharra:*
>
> *SDN sareekin gabiltzanez, ping mezuen erantzunak jasotzeko behar den denbora desberdina izango da lehen mezuaren eta beste guztien artean (lehen mezuak denbora gehiago behar du). OVSwitch-aren lotura batetik beste baterako jasotako paketeak taula batean begiratzen dira. Lehen paketea bada, taulan oraindik ez da sarrera bat egongo bide horrentzat eta beraz, lehenik kontroladorera bidali behar da. Hurrengo pakete guztiek ordea ez dute bide hori egin behar, taulan jadanik sarrera bat dagoelako eta bideraketa zuzenean egiten delako.*

xterm: Komando honekin sareko edozein ekipo kontrola daiteke, ekipo horren terminalea simulatuko duen leiho bat irekiz. Hau izango da aukera onena host-ekin ekintzak gauzatzeko. Aipatzekoa da "*ssh -X mininet@ipaddress*" egitean -X aukerari esker erabil daitekeela Xterm, X11 forwarding-a baimentzen duelako.

tcpdump: Komando ezagun hau sareko aztertzaile bat irekitzeko erabiltzen da. Wireshark programak egiten duen bezala, honekin ere ping batean sortutako paketeak ikus daitezke adibidez.

pingall: Honi esker host ekipo guztien arteko ping-ak sortzen dira, beraien arteko loturak ondo dauden konprobatu ahal izateko komando batekin bakarrik. Oso erabilgarria da sarea hasieratzean edota konfigurazioan aldaketak egitean guztia zuzen dagoen ikusteko.

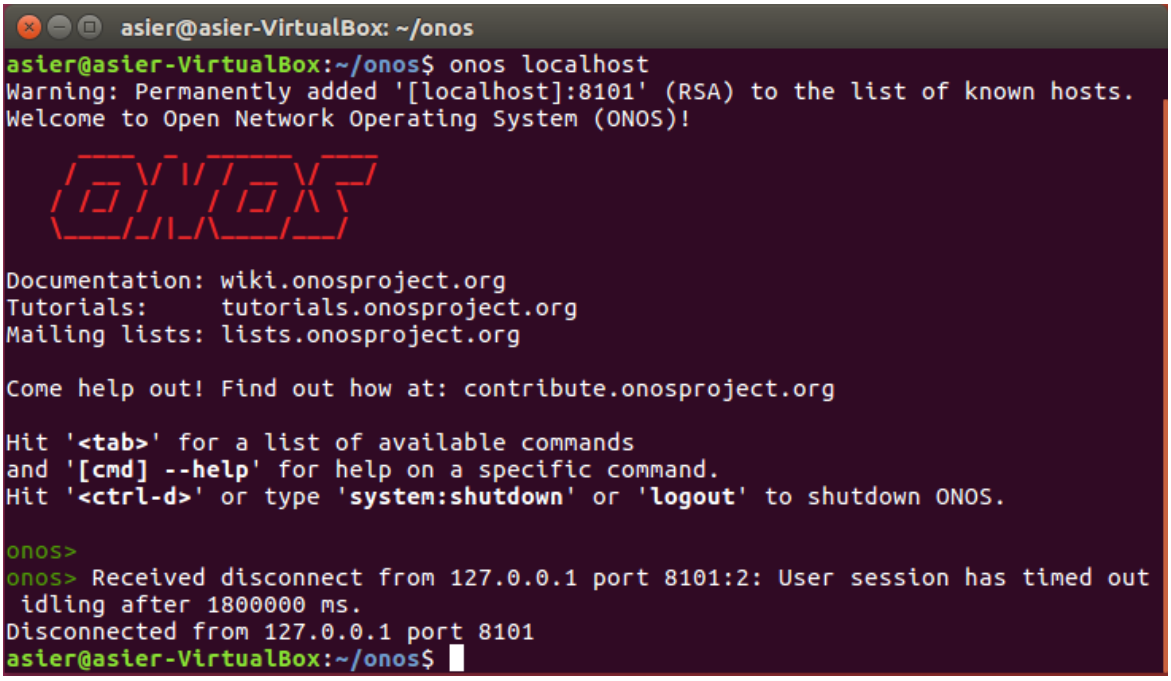
iperf: TCP banda zabalera kalkulatzeko komandoa, host-en artean dagoen gaitasuna ikusteko.

iperfudp: Aurrekoaren berdina baina UDP protokoloa erabiliz.

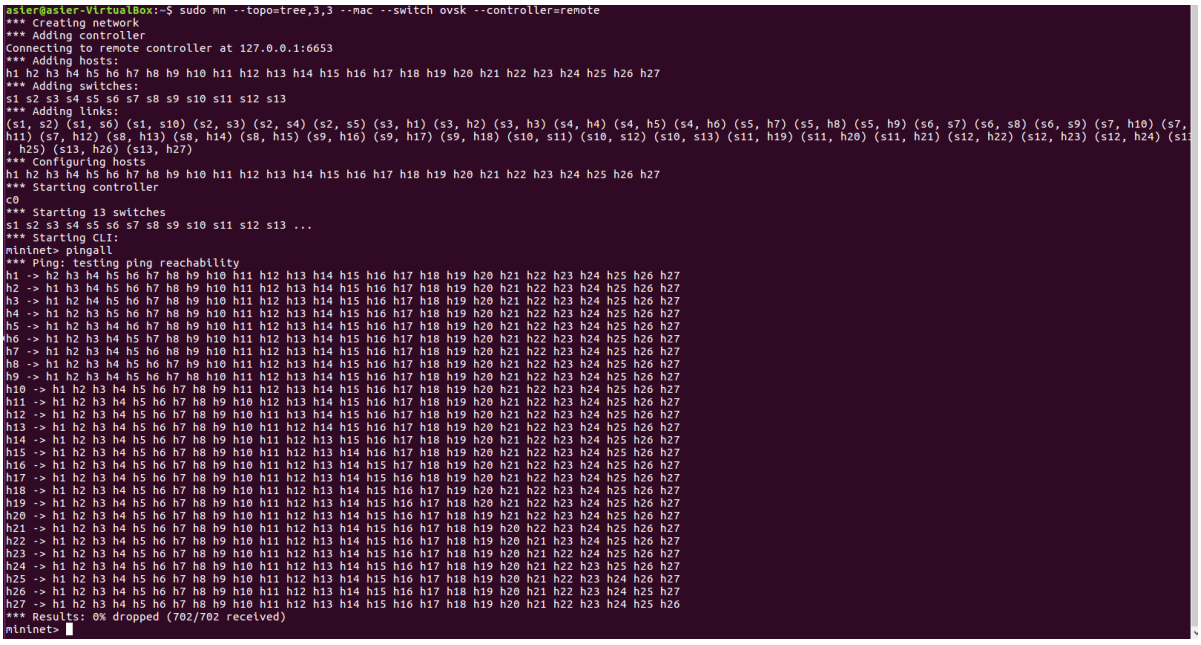
> *Oharra:*
>
> *Oso gomendagarria da mininet-eko sesio bat amaitzean "sudo mn -c" komandoa exekutatzea, sesio horretan ezarritako aukera eta konfigurazio guztiak garbitu eta hasieratzen dituelako. Zombie prozesuak ere ezabatzen ditu.*

8.2. Lehen froga gauzatzeko gida

Prozesua hasteko lehenik eta behin ONOS hasieratu behar da. Hasieratzen denean, komando lerroko interfazea (CLI) abiaraziko da, printzipioz ezer aldatzeko erabiliko ez bada ere. Behin kontroladorea guztik martxan dagoela, SDN sarea hasieratu behar da, ondoren guztien arteko konektibitatea bermatuz (guztien arteko ping-ak eginez):



29. irudia: ONOS komando lerroko interfazearen hasieraketa



30. irudia: Mininet-ekin zuhaitz topologiaren abiaraztea

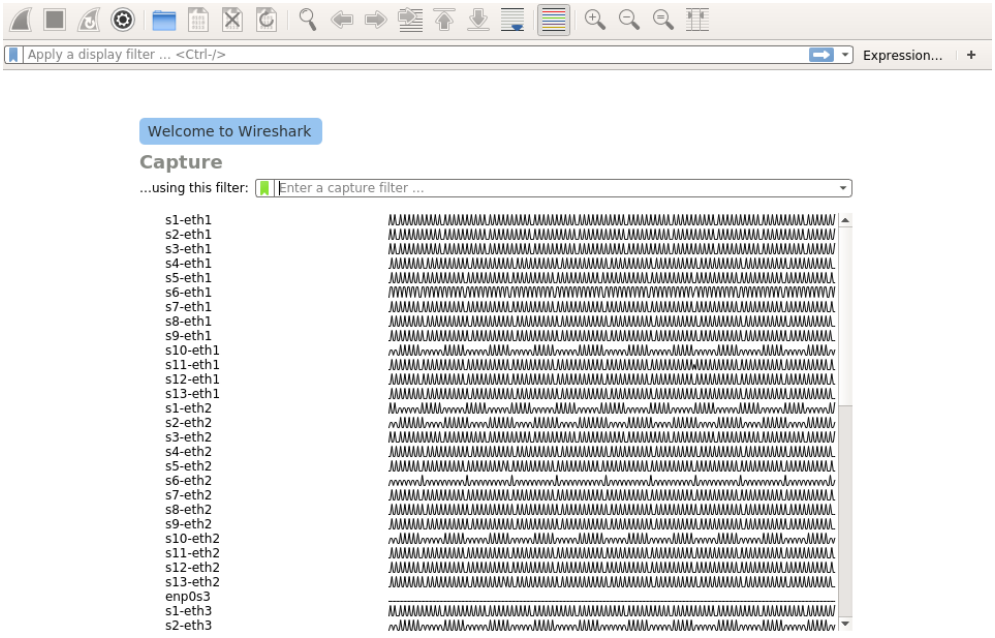
Sarearen kontrol hobea eramateko eta grafikoki bistaratu ahal izateko ONOSen GUIa (Graphical User Interface) ere abiaraziko da, nodo guztiak, loturak eta transmisioak monitorizatu ahal izateko.

Erasotzeko aurretik azaldu den Metasploit Framework-a erabili denez, root baimenekin hasi behar da programa hau ere. Terminal interfaze propioa du honek ere, eta hori erabiliz hautatuko dugu nahi den exploit-a. Kasu honetan, lehen aipatu den moduan, SYN mezuen flooding-a eragingo duena hautatu da, helburutzat makina birtualaren IP helbidea adieraziz. Hurrengo harrapaketan ikus daiteke zehatz zeintzuk diren pausoak:

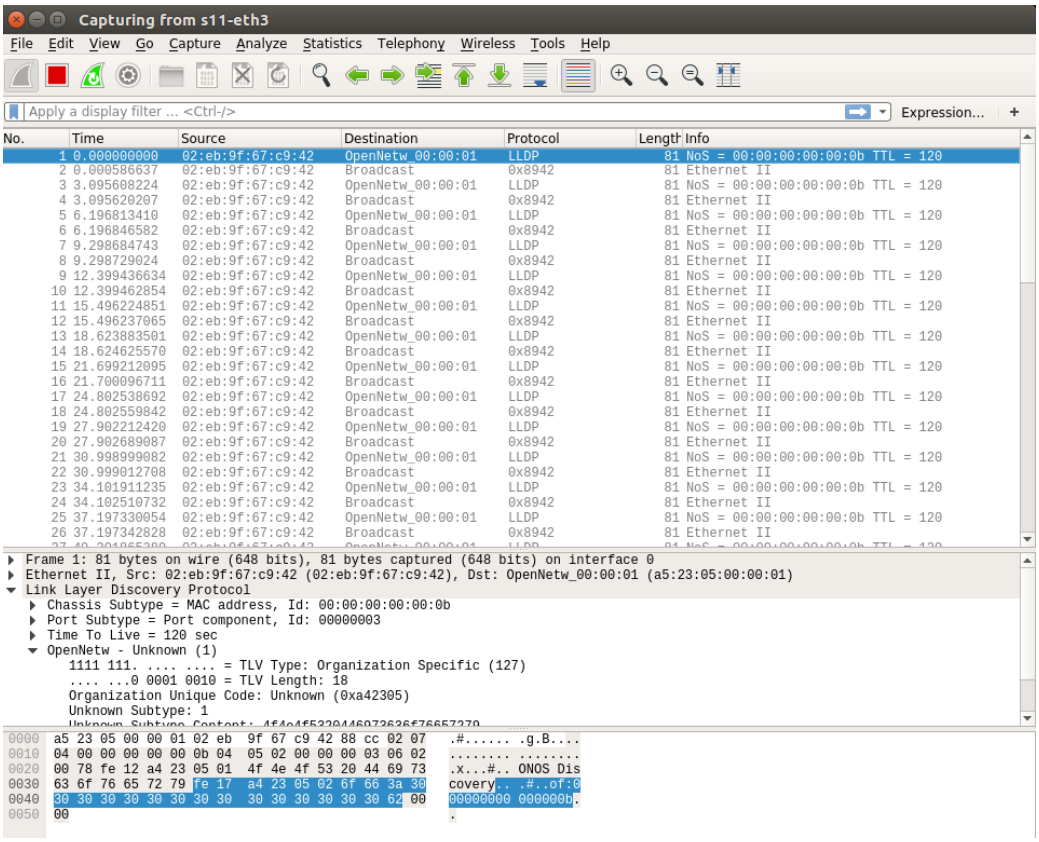
```
asier@asier-VirtualBox:~$ sudo msfconsole
[sudo] password for asier:
[-] Failed to connect to the database: could not connect to server: Connection refused
        Is the server running on host "127.0.0.1" and accepting
        TCP/IP connections on port 5433?

      dBBBBBBb  dBBBP dBBBBBBP dBBBBBb  .                    o
        '   dB'                       BBP
    dB'dB'dB' dBBP     dBP     dBP BB
   dB'dB'dB' dBP      dBP     dBP  BB
  dB'dB'dB' dBBBBP   dBP     dBBBBBBB

                                dBBBBBP  dBBBBBb  dBP    dBBBBP dBP dBBBBBBP
          .                  .                 dB' dBP    dB'.BP
                             |       dBP    dBBBB' dBP    dB'.BP dBP    dBP
                           --o--     dBP    dBP    dBP    dB'.BP dBP    dBP
                             |       dBBBBP dBP    dBBBBP dBBBBP dBP    dBP

                                                                    .
        .
       o                To boldly go where no
                         shell has gone before


       =[ metasploit v4.16.59-dev-                        ]
+ -- --=[ 1769 exploits - 1008 auxiliary - 307 post       ]
+ -- --=[ 537 payloads - 41 encoders - 10 nops            ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > setg interface enp0s9
interface => enp0s9
msf > use auxiliary/dos/tcp/synflood
msf auxiliary(dos/tcp/synflood) > set RHOST 10.0.0.6
RHOST => 10.0.0.6
msf auxiliary(dos/tcp/synflood) > exploit

[*] SYN flooding 10.0.0.6:80...
```

31. irudia: Metasploit SYN flooding

Benetan froga ondo doala ziurtatzeko Wireshark trafiko arakatzailea erabiliko da. Lehenik, Mininet-ek sortutako interfaze birtual guztietan trafikoa dagoela bermatuko dugu (OpenFlow):



32. irudia: Interfaze guztiak Wireshark-en



33. irudia: OpenFlow trafikoa Wireshark-en

8.3. Bigarren froga gauzatzeko gida

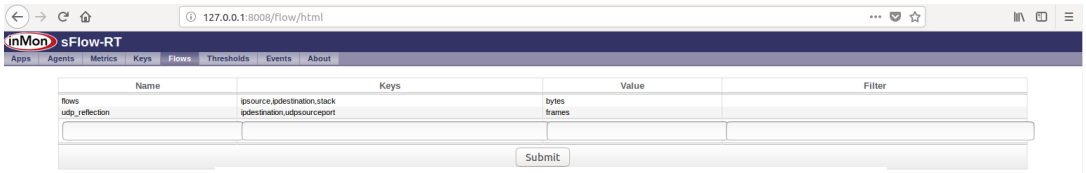
Hasteko, alde batetik sFlow eta script-a eta bestetik Mininet eta ONOS kontroladorea abiarazi behar dira:

```
asier@asier-VirtualBox:~/sflow-rt$ env RTPROP=-Dscript.file=antiflooding.js ./start.sh
2018-07-19T02:07:40+0200 INFORMACIÓN: Listening, sFlow port 6343
2018-07-19T02:07:41+0200 INFORMACIÓN: Listening, HTTP port 8008
2018-07-19T02:07:41+0200 INFORMACIÓN: antiflooding.js started
```

```
asier@asier-VirtualBox:~$ sudo mn --custom ~/onos/tools/dev/mininet/onos.py,sflo
w-rt/extras/sflow.py --link tc,bw=10 --controller onos,1 --topo tree,2,2
*** Creating network
*** Adding controller
*** Creating network
*** Adding hosts:
(unpacking /tmp/onos1)onos1
*** Adding switches:
cs1
*** Adding links:
(onos1, cs1)
*** Configuring hosts
onos1
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(10.00Mbit) (10.00Mbit) (s1, s2) (10.00Mbit) (10.00Mbit) (s1, s3) (10.00Mbit) (1
0.00Mbit) (s2, h1) (10.00Mbit) (10.00Mbit) (s2, h2) (10.00Mbit) (10.00Mbit) (s3,
 h3) (10.00Mbit) (10.00Mbit) (s3, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0 *** ONOS_APPS = drivers,openflow,fwd,proxyarp,mobility
*** Starting controller

*** Starting 1 switches
cs1
*** Enabling sFlow:
cs1
*** Sending topology
(starting onos1)
onos1(checking: karaf. ssh-port.(1 warnings)..... openflow-port...(3 warn
ings)..(4 warnings).(6 warnings)..... client node-status)
*** Waited 27.88 seconds for ONOS startup
*** Starting 3 switches
s1 s2 s3 ...(10.00Mbit) (10.00Mbit) (10.00Mbit) (10.00Mbit) (10.00Mbit) (10.00Mb
it) (10.00Mbit) (10.00Mbit)
*** Enabling sFlow:
s1 s2 s3
*** Sending topology
*** Starting CLI:
mininet-onos>
```

34. irudia: sFlow, Mininet eta ONOS kontroladorea hasieratzea

Erasoa gauzatu aurretik, sFlow softwarearen web interfazean fluxuak miatzeko sarrera bat gehitu behar da, iturriaren IP helbidea, helmugaren IP helbidea eta byte-en kopurua har ditzan. Jarraian, froga bat egin daiteke iperf-ekin TCP banda zabalera neurtzeko.



35. irudia: sFlow-ren web interfazean fluxu berriak

Beraz, horren ostean hping3-rekin erasoa gauzatzea baino ez da falta, hurrengo komandoa erabiliz:

```
mininet-onos> h1 hping3 --flood --udp -k -s 53 h3
```

8.4. Script antiflooding.js

```
var helbidea = '192.168.123.1';
var controls = {};

setFlow('udp_fluxua',
{keys:'ipdestination,udpsourceport',value:'frames'});                    //Kontrolatu
behar den fluxua definitzea
setThreshold('erasoaren_maila',
{metric:'udp_fluxua',value:100,byFlow:true,timeout:2});        //Fluxuan gainditu ezin
den threshold-a, gainditzen bada erasoa litzateke

setEventHandler(function(gert)
{                                                          //Adierazitako threshold-a
gainditzean egin behar duen ekintza
  var [ipdestination,udpsourceport] = gert.flowKey.split(',');
  var port =
topologyInterfaceToPort(gert.agent,gert.dataSource);                     //Interf
azetik nodoaren portua mapeatzeko

  var mezua =
{                                                                        //Aldagai
hau kontroladorera bidaliko da, blokeatu behar den fluxua zein
   flows:
[                                                                        //den
adierazteko. Hau ONOS fluxu rule bat da beraz, dagokion JSON formatuan
    {
     priority:10000,
     timeout:0,
     isPermanent:true,
     deviceId:'of:'+port.dpid,
     treatment: {
        instructions: [
          {
            type: "OUTPUT",
            port: "CONTROLLER"
          }
```

```
        ]
       },
      selector: {
        criteria: [
        {
          type:'IN_PORT',
          port:port.ofport
        },
        {
          type:'ETH_TYPE',
          ethType:'0x800'
        },
        {
          type:'IPV4_DST',
          ip:ipdestination+'/32'
        },
        {
          type:'IP_PROTO',
          protocol:'17'
        },
        {
          type:'UDP_SRC',
          udpPort:udpsourceport
        }
       ]
     }
    }
   ]
  };

  var resp =
http2({                                                                    //Kontrola
doreari mezua bidaltzen zaio
   url:'http://'+helbidea+':8181/onos/v1/flows?appId=ddos',
   //Horretarako logeatu behar da, hasieran definitutako erabiltzaile eta
   headers:{'Content-
Type':'application/json','Accept':'application/json'},               //pasahitzekin
   operation:'post',
   user:'onos',
   password:'rocks',
   body: JSON.stringify(mezua)
 });

 var {deviceId,flowId} = JSON.parse(resp.body).flows[0];
 controls[gert.flowKey] = {
  time:Date.now(),
  threshold:gert.thresholdID,
  agent:gert.agent,
  metric:gert.dataSource+'.'+gert.metric,
  deviceId:deviceId,
  flowId:flowId
};

 logInfo(JSON.stringify(gert));
   //Erroreen log-a
 logInfo(gert.flowKey + "helbidea eta portua dituen nodoa
blokeatzen...");                 //Informazio mezua, konsolan agertuko dena fluxua
blokeatu dela adierazteko
},['erasoaren_maila']);
```

8.5. RYU kontroladorea[31]

```
from operator import attrgetter

import simple_switch_13
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.lib import hub

import socket
import threading
import SocketServer
import subprocess
import logging

# Logging configuration
logging.basicConfig(level=logging.DEBUG)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("ofp_event").setLevel(logging.WARNING)
#logging.getLogger().addHandler(logging.StreamHandler())


# Receiving requests and passing them to a controller method,
# which handles the request
class RequestHandler(SocketServer.BaseRequestHandler):

# Set to the handle method in the controller thread
handler = None

def handle(self):
data = self.request.recv(1024)
RequestHandler.handler(data)


# Simple TCP server spawning new thread for each request
class Server(SocketServer.ThreadingMixIn, SocketServer.TCPServer):
pass


# Client for sending messages to a server
class Client:
```

```
# Initialize with IP + Port of server
def __init__(self, ip, port):
self.ip = ip
self.port = port

# Send an arbitrary message given as a string
# Starts a new thread for sending each message.
def send(self, message):
def do():
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((self.ip, self.port))
try:
sock.sendall(message)
response = sock.recv(1024)
finally:
sock.close()

thread = threading.Thread(target=do)
thread.daemon = True
thread.start()

# The main controller script, extends the already exisiting
# ryu script simple_switch_13
class SimpleMonitor(simple_switch_13.SimpleSwitch13):

# Interval for polling switch statistics
QUERY_INTERVAL = 2
# Bandwith threshold in Kbit/s for assuming an attack
# on a port
ATTACK_THRESHOLD = 4000
# Bandwith threshold in Kbit/s for assuming that the
# attack has stopped after applying an ingress policy
PEACE_THRESHOLD = 10
# Number of repeated poll statistics measurements to
# assume that the judgement on either "attack over"
# "host under DDoS attack" is correct.
SUSTAINED_COUNT = 5
```

```
# Bandwidth threshold in Kbit/s for assuming that a particular
# host is launching a DDoS attack
ATTACKER_THRESHOLD = 1000
# Specifies if polled switch statistics should reported on stout
REPORT_STATS = True

def __init__(self, *args, **kwargs):
# Monitoring
super(SimpleMonitor, self).__init__(*args, **kwargs)

# Set of currently known (assumed) attackers
self.attackers = set()
# Sustained counts for the above judgements
self.sustainedAttacks, self.sustainedPushbackRequests = 0, 0
# Indicates for each switch to which of its ports we applied an ingress policy
self.ingressApplied = {"s1": [False, False, False],
"s11": [False, False, False],
"s12": [False, False, False],
"s21": [False, False, False],
"s22": [False, False, False],
"s2": [False, False, False]}

# Sustained no attack count for switch/port combinations
self.noAttackCounts = {"s1": [0] * 3,
"s11": [0] * 3,
"s12": [0] * 3,
"s21": [0] * 3,
"s22": [0] * 3,
"s2": [0] * 3}

# Mapping from switch/port/destination MAC combinations to flow rates
self.rates = {"s1": [{}, {}, {}],
"s11": [{}, {}, {}],
"s12": [{}, {}, {}],
"s2": [{}, {}, {}],
"s21": [{}, {}, {}],
"s22": [{}, {}, {}]}

# Mapping from switches and ports to
```

```
# attached switchtes/hosts
self.portMaps = {"s1": ["s11", "s12", "s2"],
"s11": ["AAh1", "AAh2", "s1"],
"s12": ["ABh1", "ABh2", "s1"],
"s21": ["BAh1", "BAh2", "s2"],
"s22": ["BBh1", "BBh2", "s2"],
"s2": ["s21", "s22", "s1"]}

# Mapping from datapath ids to switch names
self.dpids = {0x1: "s1",
0xb: "s11",
0xc: "s12",
0x2: "s2",
0x15: "s21",
0x16: "s22"}

# Flow datapaths identified by statistics polling
self.datapaths = {}
# Last acquired byte counts for each FLOW
# to calculate deltas for bandwith usage calculation
self.flow_byte_counts = {}
# Last acquired byte counts for each PORT
# to calculate deltas for bandwith usage calculation
self.port_byte_counts = {}
# Thread for polling flow and port statistics
self.monitor_thread = hub.spawn(self._monitor)

# Pushback state
# Set of hosts, which we suspect to be victims of an attack originating
# in the other network
self.pushbacks = set()
# Set of hosts in other domain to which we were reported an attack
self.other_victims = set()

########################################
# Server Code
########################################

# Lock for the set of victims reported by the other server
```

```
self.lock = threading.Lock()
# IP + PORT for the TCP Server on this controller
ip, port = "localhost", 2000
# IP + PORT for the TCP Server on the other controller
ip_other, port_other = "localhost", 2001

# Handler for incoming requests to the server
RequestHandler.handler = self.handlePushbackMessage

# Server instance
self.server = Server((ip, port), RequestHandler)

# Initiate server thread
server_thread = threading.Thread(target=self.server.serve_forever)
# Server thread will terminate when controller terminates
server_thread.daemon = True
server_thread.start()

# Start client for sending pushbacks to the other server
self.client = Client(ip_other, port_other)

# Handler receipt of a pushback message
def handlePushbackMessage(self, data):
victim = data.strip()[len("Pushback attack to "):]
print("Received pushback message for victim: %s" % victim)
# Avoid race conditions for pushback messages
self.lock.acquire()
try:
self.other_victims.add(victim)
finally:
self.lock.release()

##########################################
# Monitoring Code
##########################################
# Handler for registering new datapaths
# Taken from http://osrg.github.io/ryu-book/en/html/traffic_monitor.html

@set_ev_cls(ofp_event.EventOFPStateChange,
[MAIN_DISPATCHER, DEAD_DISPATCHER])
```

```
def _state_change_handler(self, ev):
datapath = ev.datapath
if ev.state == MAIN_DISPATCHER:
if not datapath.id in self.datapaths:
#logging.debug('register datapath: %016x', datapath.id)
self.datapaths[datapath.id] = datapath
elif ev.state == DEAD_DISPATCHER:
if datapath.id in self.datapaths:
#logging.debug('unregister datapath: %016x', datapath.id)
del self.datapaths[datapath.id]

# Main function of the monitoring thread
# Simply polls switches for statistics
# in the interval given by QUERY_INTERVAL
def _monitor(self):
while True:
for dp in self.datapaths.values():
self._request_stats(dp)
hub.sleep(SimpleMonitor.QUERY_INTERVAL)

# Helper function for polling statistics of a datapath
# Taken from http://osrg.github.io/ryu-book/en/html/traffic_monitor.html
def _request_stats(self, datapath):
#logging.debug('send stats request: %016x', datapath.id)
ofproto = datapath.ofproto
parser = datapath.ofproto_parser

req = parser.OFPFlowStatsRequest(datapath)
datapath.send_msg(req)

req = parser.OFPPortStatsRequest(datapath, 0, ofproto.OFPP_ANY)
datapath.send_msg(req)

# Handler for receipt of flow statistics
# Main entry point for our DDoS detection code.
@set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
def _flow_stats_reply_handler(self, ev):
domainHosts = ['0a:0a:00:00:00:01', '0a:0a:00:00:00:02', '0a:0b:00:00:00:01',
'0a:0b:00:00:00:02']
```

```
# The (suspected) set of victims identified by the statistics
victims = set()

body = ev.msg.body
# Get id of datapath for which statistics are reported as int
dpid = int(ev.msg.datapath.id)
switch = self.dpids[dpid]

if SimpleMonitor.REPORT_STATS:
print "-------------- Flow stats for switch", switch, "---------------"

# Iterate through all statistics reported for the flow
for stat in sorted([flow for flow in body if flow.priority == 1],
key=lambda flow: (flow.match['in_port'],
flow.match['eth_dst'])):
# Get in and out port + MAC dest of flow
in_port = stat.match['in_port']
out_port = stat.instructions[0].actions[0].port
eth_dst = stat.match['eth_dst']

# Check if we have a previous byte count reading for this flow
# and calculate bandwith usage over the last polling interval
key = (dpid, in_port, eth_dst, out_port)
rate = 0
if key in self.flow_byte_counts:
cnt = self.flow_byte_counts[key]
rate = self.bitrate(stat.byte_count - cnt)
self.flow_byte_counts[key] = stat.byte_count
if SimpleMonitor.REPORT_STATS:
print "In Port %8x Eth Dst %17s Out Port %8x Bitrate %f" % (in_port, eth_dst, out_port,
rate)

# Save the bandwith calculated for this flow
self.rates[switch][in_port - 1][str(eth_dst)] = rate

# If we find the bandwith for this flow to be higher than
# the provisioned limit, we mark the corresponding
# host as potential vicitim
if rate > SimpleMonitor.ATTACK_THRESHOLD:
self.noAttackCounts[switch][in_port - 1] = 0
```

```
victim = str(eth_dst)

if victim in domainHosts: # If not in domain, ignore it. (Will be handled by pushback
requests)

victims.add(victim)

# Calculate no sustained attack counts

for port in range(len(self.ingressApplied[switch])):

if not self.ingressApplied[switch][port]:

continue # If ingress is not applied, skip

# If rate for all flows on the links is below safe level,

# increase the sustained no attack count for this link

if all(x <= SimpleMonitor.PEACE_THRESHOLD for x in self.rates[switch][port].values()):

self.noAttackCounts[switch][port] += 1

else:

self.noAttackCounts[switch][port] = 0

victims = victims.intersection({'0a:0a:00:00:00:01', '0a:0a:00:00:00:02'}) # only
consider the protected hosts

# Handle pushback requests from the other host

self.dealWithPushbackRequests()

# Identify the set of victims attacked by hosts located in the other domain

# and directly apply policies to the attackers in the local domain

pushbacks = self.dealWithAttackers(victims)

if pushbacks == self.pushbacks and len(pushbacks) > 0: # Send pushback messages

self.sustainedPushbackRequests += 1

logging.debug("Sustained Pushback Count %s" % str(self.sustainedPushbackRequests))

if self.sustainedPushbackRequests > SimpleMonitor.SUSTAINED_COUNT:

for victim in pushbacks:

self.client.send("Pushback attack to " + victim)

self.sustainedPushbackRequests = 0

elif len(pushbacks) > 0:

self.sustainedPushbackRequests = 0

self.pushbacks = pushbacks

self.checkForIngressRemoval(victims) # If there are no victims, for a sustained
duration, try remove ingress policies

if SimpleMonitor.REPORT_STATS:
```

```
print "------------------------------------------------------"


# Handle pushback requests issued by the controller in the other domain

def dealWithPushbackRequests(self):

victims = set()

# Avoid race conditions pertaining to pushbacks

self.lock.acquire()

try:

victims = self.other_victims

self.other_victims = set()

finally:

self.lock.release()


for victim in victims:

# Identify attackers for the victims

victimAttackers = self.getAttackers(victim)

print("Responding to pushback request, applying ingress on %s to relieve %s" %
(victimAttackers, victim))

# Apply an ingress policy to each attacker

for attacker in victimAttackers:

self.applyIngress(attacker)


# Identify the set of victims attacked by hosts located in the other domain

# and directly apply policies to the attackers in the local domain

def dealWithAttackers(self, victims):

# Set of victims attacked by the other domain

pushbacks = set()

# Set of attackers in the local domain

attackers = set()

for victim in victims:

victimHost, victimSwitch, victimPort = self.getVictim(victim)

print("Identified victim: MAC %s Host %s Switch %s Port %s" % (victim, victimHost,
victimSwitch, victimPort))

victimAttackers = self.getAttackers(victim)

print("Attackers for vicim %s: %s" % (victimAttackers, victimHost))

if not victimAttackers:

# No attackers identified, thus assume it's originating in the other domain

pushbacks.add(victim)
```

```
else:
attackers = attackers.union(victimAttackers)

# Increase the count for confidence in a suspected attack
# by the identifed attacker set if applicable
if attackers:
self.sustainedAttacks += 1
logging.debug("Sustained Attack Count %s" % (self.sustainedAttacks / 3))
else:
self.sustainedAttacks = 0

# If we have exceeded the confidence count for the local attacker
# set, apply ingress policies to all attackers
if self.sustainedAttacks / 3 > SimpleMonitor.SUSTAINED_COUNT:
for attacker in attackers:
self.applyIngress(attacker)

return pushbacks

# Check if the ingress policy should be removed for any port
def checkForIngressRemoval(self, victims):
# If the confidence count for no ongoing attack exceeds the provisioned limit
# check if the bandwith consumption on one of the rate-limited links
# dropped below a "safe" level and remove ingress policy
for switch in self.ingressApplied: # Iterate through all switches/ports
for port in range(len(self.ingressApplied[switch])):
# If rate for all flows on the links for this port have been below a safe level
# for the last couple of statistic readings, remove the ingress policy
if self.noAttackCounts[switch][port] >= self.SUSTAINED_COUNT and
self.ingressApplied[switch][port]:
self.removeIngress(self.portMaps[switch][port])

# Applies ingress to a given attacker's switch/port
def applyIngress(self, attacker, shouldApply=True):
attackerSwitch, attackerPort = self.getSwitch(attacker)
if self.ingressApplied[attackerSwitch][int(attackerPort) - 1] == shouldApply:
return

ingressPolicingBurst, ingressPolicingRate = "ingress_policing_burst=0",
"ingress_policing_rate=0"
```

```
if shouldApply:

self.noAttackCounts[attackerSwitch][int(attackerPort) - 1] = 0

print("Applying ingress filters on %s, on switch %s at port %s" % (attacker,
attackerSwitch, attackerPort))

ingressPolicingBurst, ingressPolicingRate = "ingress_policing_burst=100",
"ingress_policing_rate=40"

else:

print("Removing ingress filters on %s, on switch %s at port %s" % (attacker,
attackerSwitch, attackerPort))


subprocess.call(["sudo", "ovs-vsctl", "set", "interface", attackerSwitch + "-eth" +
attackerPort, ingressPolicingBurst])

subprocess.call(["sudo", "ovs-vsctl", "set", "interface", attackerSwitch + "-eth" +
attackerPort, ingressPolicingRate])

self.ingressApplied[attackerSwitch][int(attackerPort) - 1] = shouldApply


# Removes ingress at the given attacker's switch/port

def removeIngress(self, attacker):

self.applyIngress(attacker, False)


# Returns the victim's switch, and port it is connected to

def getVictim(self, victim):

victimHost = victim[1].upper() + victim[4].upper() + "h" + victim[16]

for switch in self.portMaps:

for port in range(len(self.portMaps[switch])):

if self.portMaps[switch][port] == victimHost:

return victimHost, switch, str(port + 1)


# Returns the local attackers of a given victim

def getAttackers(self, victim):

attackers = set()

for switch in self.rates:

for port in range(len(self.rates[switch])):

if victim not in self.rates[switch][port]:

continue

if self.rates[switch][port][victim] > SimpleMonitor.ATTACKER_THRESHOLD:

attacker = self.portMaps[switch][port]

if not self.isSwitch(attacker):

attackers.add(attacker)


return attackers
```

```
@staticmethod
def isSwitch(victim):
return victim[0] == "s"

def getSwitch(self, node):
for switch in self.portMaps:
if node in self.portMaps[switch]:
return switch, str(self.portMaps[switch].index(node) + 1)

# Convert from byte count delta to bitrate
@staticmethod
def bitrate(bytes):
return bytes * 8.0 / (SimpleMonitor.QUERY_INTERVAL * 1000)

# Handle receipt of port traffic statistics
@set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
def _port_stats_reply_handler(self, ev):
body = ev.msg.body

for stat in sorted(body, key=attrgetter('port_no')):
key = (ev.msg.datapath.id, stat.port_no)

rx_bitrate, tx_bitrate = 0, 0
if key in self.port_byte_counts:
cnt1, cnt2 = self.port_byte_counts[key]
rx_bitrate = self.bitrate(stat.rx_bytes - cnt1)
tx_bitrate = self.bitrate(stat.tx_bytes - cnt2)
self.port_byte_counts[key] = (stat.rx_bytes, stat.tx_bytes)
```

9. Erreferentziak

[1] "The Past, Present, and Future of Software Defined Networking" – University of Maryland http://www.ietf.org/proceedings/84/slides/slides-84-iab-techplenary-6.pdf

[2] "Fourth industrial revolution" – Uzair M. Younus, Albright Stonebridge Group, Washington D.C., May 09, 2017

[3] "Software-Defined Networking: The New Norm for Networks" – Open Networking Foundation, April 13, 2012 https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[4] "Virtual machine monitors: current technology and future trends" – IEEE Computer Society https://ieeexplore.ieee.org/abstract/document/1430630/

[5] "Recommendations for implementing the strategic initiative INDUSTRIE 4.0" – ForschungsUnion, Wirtschaft und Wissenschaft begleiten die Hightech-Strategie http://www.acatech.de/fileadmin/user_upload/Baumstruktur_nach_Website/Acatech/root/de/Material_fuer_Sonderseiten/Industrie_4.0/Final_report__Industrie_4.0_accessible.pdf

[6] "A Survey on Software-Defined Networking" - IEEE Communications Surveys & Tutorials https://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=6834762

[7] "Internet Service Provider (ISP)" – Technopedia https://www.techopedia.com/definition/2510/internet-service-provider-isp

[8] "SDN Security Attack Vectors and SDN Hardening" – NetworkWorld https://www.networkworld.com/article/2840273/sdn/sdn-security-attack-vectors-and-sdn-hardening.html

[9] "What is a denial of service attack (DoS)?" – Cyberpedia, Palo Alto Networks https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos

[10] "What is the Internet of Things?" – Wired http://www.wired.co.uk/article/internet-of-things-what-is-explained-iot

[11] "What is OpenFlow?" – SDXCentral https://www.sdxcentral.com/sdn/definitions/what-is-openflow/

[12] "Ethical hacking" – IEEE Xplore https://ieeexplore.ieee.org/abstract/document/5386933/

[13] "VirtualBox" – VirtualBox.org https://www.virtualbox.org/manual/

[14] "VMware" https://www.vmware.com/

[15] "Mininet" – Mininet
http://mininet.org/

[16] "GNS3" – Universidad Complutense de Madrid
https://www.ucm.es/pimcd2014-free-software/gns3

[17] "Exploit" – Searchsecurity
https://searchsecurity.techtarget.com/definition/exploit

[18] "VMware vs VirtualBox" – Softzone
https://www.softzone.es/2017/03/14/comparativa-vmware-virtualbox/

[19] "Nested VM (nested virtual machine)" – SearchVMware
https://searchvmware.techtarget.com/definition/nested-VM-nested-virtual-machine

[20] "Command Line Interface (CLI)" – Technopedia
https://www.techopedia.com/definition/3337/command-line-interface-cli

[21] "El modelo OSI" – Universitat Politècnica de València (UPV)
http://www.upv.es/visorv/media/5a8d4b8b-239d-fe4e-9a75-3188cb4de3fb/c

[22] "SDN Controllers: A Comparative Study" – American University of Beirut
https://www.researchgate.net/publication/304457462_SDN_controllers_A_comparative_study

[23] "Kali Linux" – Offensive Security
https://www.kali.org/

[24] "Metasploit" – Semantic Scholar
https://www.semanticscholar.org/topic/Metasploit/113283

[25] "Pentesting" – OpenWebinars
https://openwebinars.net/blog/que-es-el-pentesting/

[26] "hping3" – Kali Tools
https://tools.kali.org/information-gathering/hping3

[27] "DDoS Attacks" – Imperva Incapsula (DDoS Protection Center)
https://www.incapsula.com/ddos/ddos-attacks.html

[28] "sFlow-RT"
https://sflow-rt.com/

[29] "Defining Thresholds" – Webnms
https://www.webnms.com/telecom/help/administrator_guide/performance/thresholds/perf_thresholdsintro.html

[30] "A Systematic and Generic Method for Choosing A SDN Controller" – Omayma Belkadi and Yassin Laaziz, LabTIC National School of Applied Sciences in Tangier, Department of

Communication systems and Computer science, Morocco
http://www.ijcncs.org/published/volume5/issue11/p1_5-11.pdf

[31] “DDoS Attack Mitigation System” – Github
https://github.com/mishra14/DDoSAttackMitigationSystem