УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Никола Блажић

Аутоматизација експеримената у ДЕТЕР окружењу

МАСТЕР РАД

Нови Сад, 2019.

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Никола Блажић

Аутоматизација експеримената у ДЕТЕР окружењу

МАСТЕР РАД

Ментор:
Проф. Др Илија Башичевић

Студент:
Никола Блажић
E2 84/2017

Нови Сад, 2019.

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:			
Идентификациони број, **ИБР**:			
Тип документације, **ТД**:		Монографска документација	
Тип записа, **ТЗ**:		Текстуални штампани материјал	
Врста рада, **ВР**:		Завршни (Bachelor) рад	
Аутор, **АУ**:		**Никола Блажић**	
Ментор, **МН**:		**Проф. др. Илија Башичевић**	
Наслов рада, **НР**:		**Аутоматизација експеримената у ДЕТЕР окружењу**	
Језик публикације, **ЈП**:		Српски / ћирилица	
Језик извода, **ЈИ**:		Српски	
Земља публиковања, **ЗП**:		Република Србија	
Уже географско подручје, **УГП**:		Војводина	
Година, **ГО**:		**2019.**	
Издавач, **ИЗ**:		Ауторски репринт	
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6	
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		**7/65/1/0/32/0/0**	
Научна област, **НО**:		Електротехника и рачунарство	
Научна дисциплина, **НД**:		Рачунарска техника	
Предметна одредница/Кључне речи, **ПО**:		**Deterlab, емулација, аутоматизација експеримената, SYN Flood напад**	
УДК			
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, **ВН**:			
Извод, **ИЗ**:		У овом завршном раду је развијена програмска подршка за аутоматизацију експеримената у DETERLab окружењу. Програмска подршка је развијена у Python и Bash језицима. У раду је описан процес реализације експеримента коришћењем развијене програмске подршке. За испитивање је коришћен сценарио емулације Syn Flood напада одбијањем услуге.	
Датум прихватања теме, **ДП**:			
Датум одбране, **ДО**:			
Чланови комисије, **КО**:	Председник:	Проф. др. Небојша Пјевалица	
	Члан:	**Проф. др. Драган Пејић**	Потпис ментора
	Члан, ментор:	**Проф. др. Илија Башичевић**	

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Master Thesis	
Author, **AU**:		**Nikola Blažić**	
Mentor, **MN**:		**PhD Ilija Bašičević**	
Title, **TI**:		**Automatization of experiments in DETER environment**	
Language of text, **LT**:		Serbian, Cyrillic	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Republic of Serbia	
Locality of publication, **LP**:		Vojvodina	
Publication year, **PY**:		**2019.**	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		**7/65/1/0/32/0/0**	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S**/**KW**:		Deterlab, emulation, automation of experiments, SYN Flood attack	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		In this diploma work, a software has been developed to support automation of experiments in DETERLab environment. The software has been developed in Python and Bash script languages. The process of experiment realization using this software has been explained. The software has been tested for emulation of Syn Flood denial of service attacks.	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	**PhD Nebojša Pjevalica**	
	Member:	**PhD Dragan Pejić**	Menthor's sign
	Member, Mentor:	**PhD Ilija Bašičević**	

Садржај

Списак слика

Скраћенице

DoS - Denial Of Service

DDoS - Distributed Denial Of Service

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

ICMP - Internet Control Message Protocol

PCAP - Packet Capture стандардни формат

EWMA - Exponentially Weighted Moving Average

IIR - Infinite Impulse Response

CUSUM - Cumulative Sum

MAGI - Montage AGent Infrastructure: систем за оркестрацију мрежних експеримената у DETERLab окружењу.

AAL - Agent Activation Language

NS3 - Network Simulator 3

DETERLab - DEfense Technology Experimental Research Laboratory

GENI - отворена инфраструктура за умрежавање и дистрибуцију системских истраживања и образовања која обухвата подручје SAD-а

DHT - Distributed Hash Table

ECMP - Equal Cost Multi Pathing

1. Увод

У овом завршном раду је развијена програмска подршка за аутоматизацију експеримената у DETERLab окружењу. Програмска подршка је развијена у Python и Bash језицима. У раду је описан процес реализације експеримента коришћењем развијене програмске подршке. За испитивање је коришћен сценарио емулације Syn Flood напада одбијањем услуге.

1.1 Шта је напад одбијања услуге (DOS напад):

Напад одбијања услуге је напад који привремено или трајно спречава услуге неког сервиса на интернету или у локалној мрежи. Овај напад се обично изводи тако што нападач шаље велики број пакета на мету коју жели да онеспособи, затим велики број пакета нападача надјачава број пакета које сервер може да прими, па се велики број пакета од легитимних клијената изгуби. Већина оваквих напада се јако лако изводи и не захтева специјалне вештине нападача, а интернет мрежа још увек није довољно заштићена од оваквих напада.

Заштита од овог напада подразумева проналажење, класификацију и одстрањивање нападачких пакета из мреже пре него што стигну до сервера. Пакети који су пар нивоа хијерархије пре сервера, у мрежном каналу чине мали удео у односу на легитимне пакете, па самим тим не нарушавају квалитет мреже и могуће их је одстранити. Али уколико пакети нису одстрањени пре него што стигну до сервера, они конвергирају у једну тачку (сервер) и тако преузимају веома значајан удео пакета у односу на легитимне пакете и тиме гуше саобраћај легитимних корисника или чак трајно онеспособљавају рад сервера (док се поново не покрене).

1.2 Типови DOS напада

Постоји више врста DoS напада који на различите начине онеспособљавају нападнут систем.

1. **Преплављивање UDP пакетима (eng. UDP Flood)**

   Нападач шаље велики број UDP пакета на случајно изабране портове ка мети коју напада. Као резултат напада, типични рачунар који је нападнут проверава да ли постоји апликација која слуша порту који се напада, види да ни једна апликација не слуша на том порту и одговара ICMP поруком ”ICMP Destination Unreachable”, која обавештава пошиљаоца да сервис не постоји на том порту. Таква порука нападачу иде у корист јер додатно појачава снагу напада.

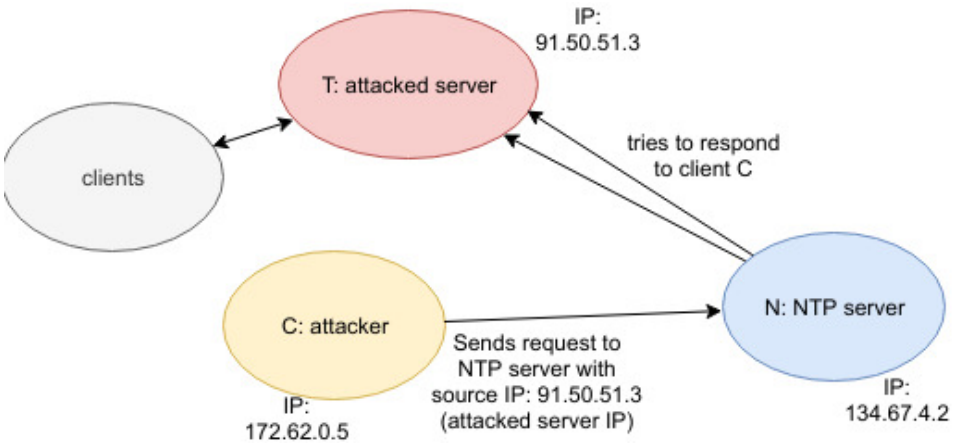
   Такође нападач још може у пакету који шаље да измени адресу са које пакет потиче и тиме омогући да нападнута мета не шаље назад пакете ка нападачу него ка неким случајним адресама, што омогућава нападачу да остане анониман као и да његов канал не буде оптерећен повратним ICMP пакетима који би пристизали од сервера који је нападнут.

2. **Одбијени напад или појачани напад (eng. Reflected Attack, Amplified attack)**

   Уколико постоји неки познати сервис који одговара са више пакета него број пакета који је послат том сервису, тај сервис постаје алат за извршавање појачаног напада на неку другу мету. Наиме нападач мења адресу са које шаље тако да показује на адресу мете, што значи да одговори на питања која су постављена сервису неће се враћати нападачу него иду право на мету коју нападач жели да обори. Захваљујући појачању напада, нападач може са много мање пакета да изврши много већи напад.

   На пример: NTP сервис садржи команду звану MONLIST која се може послати ка NTP серверу за сврхе праћења стања сервера, NTP затим враћа адресе до чак 600 машина са којима је NTP сервер комуницирао. Овај одговор је много већи од самог захтева (MONLIST команде). Што значи да сервер одговара са 100 пакета, који укупно чине 48kB као одговор на захтев који је велик само 234 бајта. То значи да је појачање напада чак 48*1024/234 = 206x.

   Такође разне SNMP команде такође могу бити коришћене за појачање напада до чак преко 600x. Иначе SNMP сервери обично захтевају аутентификацију али су многи SNMP сервери јако лоше заштићени са углавном прилично лошом шифром. Још један добро познати начин напада је коришћењем DNS сервера. Постоје још многи други сервиси који нису добро заштићени и враћају веће поруке од самог захтева и тиме омогућавају овај веома моћан начин DoS напада.



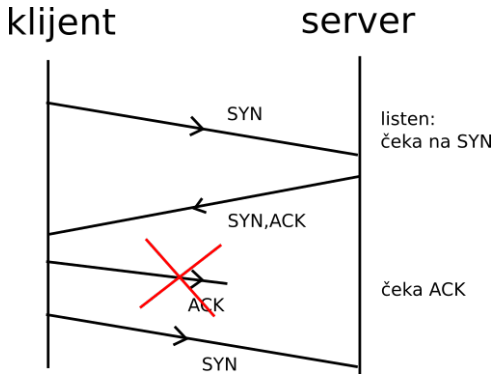
Слика 1: Одбијени напад

3. **Преплављивање SYN пакетима (енг. TCP Syn Flood)**

   TCP сервер у себи садржи ограничен број могућих веза које истовремено могу бити отворене на једном порту на којем слуша, ово је ограничено оперативним системом или оперативном меморијом.

   Овај напад функционише тако што нападач започиње успоставу везе слањем SYN пакета, али је не завршава слањем ACK поруке након што сервер одговори са SYN-ACK поруком. Сервер након примања SYN пакета резервише ресурсе као што је аутомат стања везе и чека да стигне ACK порука од клијента.

   Након пристизања великог броја SYN пакета од нападача, сервер остаје без слободних ресурса за отварање нове везе и сви нови легитмни клијенти морају бити одбијени јер је достигнут максималан број могућих веза које се могу остварити.



Слика 2: Syn flood напад

4. **Напад коришћењем Peer-to-Peer мреже**

   Сервер који служи за дељење IP адреса другим клијентима како би се они међусобно директно повезали и директно комуницирали једни са другим, као на пример

torrent tracker сајт или сервери са дистрибуираном hash табелом (DHT), нападач може, уколико поседује један од тих ”bootstrap” сервера, да преусмери све клијенте да се покушају повезати са IP адресом од мете која се напада, чинећи ово DDoS напад.

5. **Напади на апликативном нивоу (eng. Application Level Attacks)**

   Ово су напади на највишем нивоу и најтежи су за детекцију јер често изгледају као легитимни пакети док су уствари део DoS напада. Рањивост на овај тип напада зависи од апликације која се напада.

   На пример напад Slowloris који ради на принципу отварања много веза са нападнутим web сервером и одржава их отвореним што дуже. То ради тако што непрекидно шаље половичне HTTP пакете и отвара нове везе. Слањем половичних али исправних пакета може се провући кроз традиционалне системе за детекцију DoS напада и тиме нарушити квалитет услуге трошењем процесорског времена нападнутог система. Као и сигурносни систем који има пропуст такав да први започиње сложену обраду информација без тражења од клијената да они први крену сложену обраду, као на пример електронски потпис или криптовање се често злоупотребљава да се са јефтиним слањем пакета зада велики посао за сервер.

6. **Нарушавање сервиса (eng. Degradation of Service)**

   Овакав тип напада нема за циљ да потпуно прекине рад сервиса него само да погорша његов рад тако што проузрокује кашњење и скокове у кашњењу пакета. Овај тип напада је најуспешнији против сервиса који раде у реалном времену, попут видео игрица, видео стрима и сличних сервиса. Пакети се шаљу контролисаном брзином тако да не прекидају сервис, али га чине неупотребивим за кориснике који очекују одзив сервиса у реалном времену.

1.3 Дистрибуирани напад ускраћивања услуге (DDoS)

DDoS (eng. Distributed Denial of Service) је најчешћи облик DoS напада где нападач користи велики број рачунара да нападне мету. Углавном нападач не поседује велики број рачунара, него користи разне начине ширења вируса који се крије у зараженим рачунарима и периодично тражи команду коју да изврши, тј. коју мету да нападне.

Таква мрежа преузетих рачунара који слушају наређење једног извршиоца зове се ботнет, а рачунари у тој мрежи зову се зомби рачунари. Након што нападач објави команду за напад на одређену мету зомби рачунарима је потребно неко време док се сви не укључе у напад, ово време почетка напада је ограничено периода провере задатка. Уколико би период провере задатка био превише мали, зомби рачунари би вршили DDoS напад на сервер нападача који задаје те команде.

При нападу, сваки компјутер углавном шаље само мали број пакета, али маса тих компјутера заједно чини веома моћну силу и веома брзо обарају систем којег нападају.

1.4 Генерисање и валидација мрежних записа у сврху анализе алгоритама за детекцију DoS напада

Пошто су математички модели теоретске природе, они не могу бити коришћени за мрежне експерименте јер представљају веома грубу апроксимацију мреже. Док симулационе технике не могу да симулирају реалистичне апликације него могу само да апроксимирају одређене хадрверске и софтверске функционалности.

Емулација омогућава начин да се искористи реални хардвер и апликације, али је његова функционалност ограничена бројем чворова, типом хардвера и тежином конфигурисања, менаџмента и репродуктивности експеримента у односу на симулације. Реални системи су идеални за валидацију мрежног истраживања али су ограничени у функционалности због њихове сложености.

Научници континуално раде на решавању проблема реалних система. Коришћење софтверски дефинисане мреже је добар приступ проблему конфигурисања мреже који чини мреже веома скалабилне и програмабилне.

Из истраживања [13] је закључено да се реални мрежни записи доста користе за евалуацију и валидацију DDoS истраживања. Прави датасет чине подаци снимљени из реалног система користећи реалистичне мрежне услове што укључује стварни оперативни систем, апликације и платформе. У супротном, ови мрежни записи се могу синтетички генерисати у затвореном лабораторијском окружењу користећи емулације као DETERLab, Emulab, GENI или симулације као HC3 и тако даље. Због њихове екстензивне употребе за валидацију DDoS проучавања, предложено је додавање нове димензије у постојећи скуп валидационих техника.

Један од начина генерисања реалистичног мрежног записа је да се унутар мрежног записа нормалног саобраћаја унесе малициозни саобраћај који је генерисан софтверски, и запис као такав да се користи у обради.

Експериментални програмски пакети као што су DETERLab, HC3, Emulab, GENI омогућавају истраживачима имплементацију и евалуацију DDoS алгоритама за откривање и одбрану.

Генеричка својства која нуде ови програмски пакети за генерисање мрежног записа су:

1. Веродостојност:

Поставка мрежног експеримента треба да поседује критеријум веродостојности, што

значи поузданост и зависност од стварних мрежа. Димензија веродостојности обухвата велику топологију која има довољан број чворова, правих рутера, хетерогене мешавине хардвера и софтвера, и комбинације реалистичног пропусног опсега линкова, капацитета и кашњења.

2. Поновљивост:

   Поставка мрежног експеримента треба да поседује могућност за чување и репродуковање експеримента под истим условима. Међутим, фактори као што су интернет топологија, расположиви пропусни опсег, побољшања и измене у софтверу и хардверу, као и тип позадинског саобраћаја и нападачке мреже чини да је веома тешко поновити експеримент користећи стварне системе.

3. Програмабилност:

   Мрежни експеримент би требао да има флексибилност коришћења нових прилагођених мрежних механизама за праћење, филтрирање, детекцију, додавање или модификовање рутерских алгоритама, стварних хетерогених хардвера, и слично. Употреба софтверских рутера може омогућити флексибилност програмерима.

   Екстензибилност: експеримент би требало да има одредбу о скалирању топологије експеримента у односу на дивљи интернет. Експеримент треба да буде преносив и треба постојати опција да им се приступи даљински.

4. Функционалност истраживања:

   Поред контроле хардверских и софтверских компоненти експеримента, за експерименте засноване на безбедности, такође постоји потреба за олакшањем техничког и социјалног окружења за експерименте као што су велики број генератора саобраћаја и топологије.

5. Ниво апстракције:

   Апстракција је количина сложености којом се систем посматра или програмира. Што је виши ниво, мање је детаља и обрнуто.

   Тип коришћених мрежних елемената: Овај параметар наводи тип мрежних елемената коришћених у експерименту као на пример: реални чворови, меки чворови (софтверски) или мешана варијанта.

Идеалан и реалистичан запис саобраћаја мора да поседује следеће особине:

1. Права изворна и одредишна IP адреса

   Како би симулирали реалистичан сценарио у саобраћају, потребно је да клијенти направе валидне TCP конекције са сервером и приступају стварним страницама на

серверу, што је могуће само ако су и клијентске и серверске IP адресе стварне. Често се у јавним мрежним записима IP адресе измене као и садржај пакета због очувања приватности података у мрежи што може да буде проблематично за коришћење.

2. Широк низ случајних изворних IP адреса

   Мрежни саобраћај иницран са широког опсега IP адреса као што је видљиво у карактеристичном интернет саобраћају формира кључну карактеристику реалистичног мрежног записа. Из тог разлога је потребно да се нападачки саобраћај генерише коришћењем широког опсега IP адреса.

3. Стварни пакети са исправним заглављем

   Да би се осигурао реалистичан саобраћај, пожељно је да пакети имају валидна заглавља.

4. Одговарајући однос нормалног и нападачког саобраћаја.

   Да би се ефикасно проверила било која DDoS нападачка техника на способност разних алгоритама да их детектују, мрежни саобраћај који се генерише мора садржати одговарајући однос легитимног и малициозног нападачког саобраћаја. Овај однос саобраћаја утиче на бројне параметре прагова алгоритама детекције.

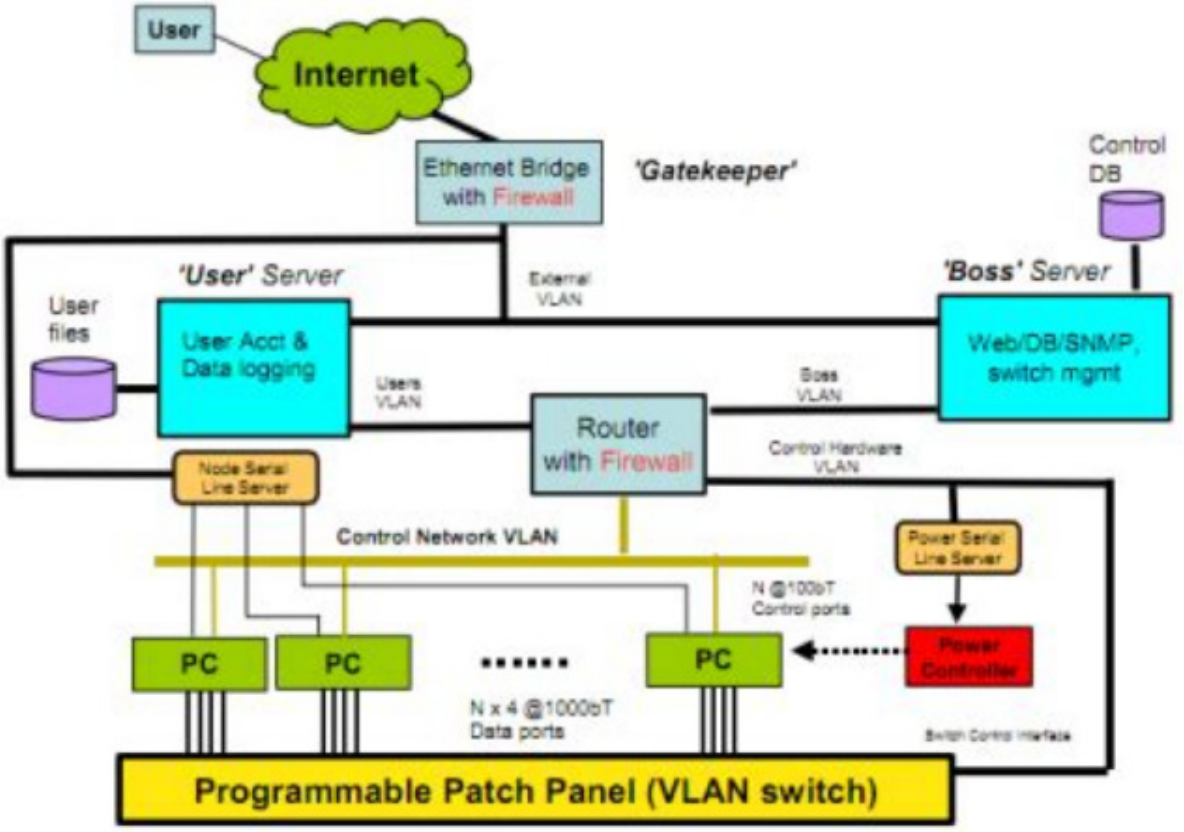
1.5 DETERLab окружење

DETERLab је научно истраживачки објекат за истраживаче сајбер безбедности. Користи се за истраживање, развијање, проналажење, експериментисање и тестирање иновативних сајбер безбедносних технологија. DETERLab је настао од Emulab-а. Док се Emulab концетрише на јаке рачунаре, DETERLab има циљ да омогући велики број слабијих рачунара за коришћење. Пројекти DETERLab-а укључују анализу DDoS напада, разних малигних софтвера попут вируса и ботнет напада. Такође се често користи за тестирање нових протокола и њихово функционисање у оквиру дистрибуираних система, као на пример тестирање blokchain технологија, тестирање разних cloud система, као и тестирање на њихове слабости у случају разних хакерских експлоита и напада. Експеримент на DETERLab-у је сачињен од једне или више машина у интерној DETERLab мрежи, која се налази иза firewall-а. users.isi.deterlab.net (или users скраћено) је контролни сервер за DETERLab, има на себи FreeBSD оперативни систем и представља дељени рачунар на којег се сви корисници повезују у сврху рада са експериментима.

”User” сервер (users.isi.deterlab.net) је контролни сервер на који сваки корисник приликом извршења експеримента мора да се повеже са SSH протоколом, након чега се долази до FreeBSD окружења са којег се коришћењем одређених команди покрећу и управљају

експерименти.

”Boss” сервер (myboss.isi.deterlab.net) је главни сервер који омогућава DETERLab функционисање, он покреће DETERLab web страницу, базу података, врши покретање рачунара у rack-овима коришћењем Power-line контролера, и врши конфигурацију switch-ева по спецификацији корисника. Корисницима није допуштен приступ да се прикаче директно на њега, него само преко одређених команди које су дате за коришћење.



Слика 3: Шира слика DETERLab-а

DETERLab се ослања на switch-еве који подржавају софтверску конфигурацију и преко њих се остварује повезивање мреже између физичких рачунара на основу дефинисане топологије (по жељи корисника). Сви чворови који се покрећу у оквиру експеримента налазе се унутар Firewall-а и могуће им је приступити само преко контролног сервера. Да би им се приступило директно, постоји опција да се направи SSH тунел до тих машина.

2. Прављење и извршавање експеримента

Извршење експеримента се одвија у пар корака:

1. Дизајнирање топологије topology.tcl(.ns)
2. Инстанцирање експеримента, покретање и заузимање ресурса за експеримент (swapin)

   На основу топологије експеримента описане у датотеци topology.tcl, DETER-Lab софтвер аутоматски (након анализе топологије) зна колико физичких чворова је потребно заузети и генерише конфигурацију за експеримент.
3. Генерисање протока на чворовима

   Помоћу оркестратора покрећу се апликације на чворовима које генеришу проток и снимају га са tcpdump програмом.
4. Преглед резултата

   Сваком чвору је могуће приступити са SSH алатом, и преузети резултате од интереса.

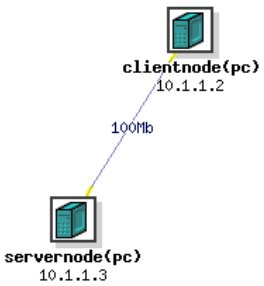
2.1 Дизајн топологије и покретање једноставног експеримента

За сваки експеримент потребно је дефинисати топологију мреже која дефинише чворове, атрибуте чворова и начин повезаности чворова. Начин писања топологије је преузет из нс-2 мрежног симулатора и компатибилан је са NS-2 симулатором па одатле проистиче да се топологија описује у TCL програмском језику. Пример једноставне топологије са 2 чвора названих clientnode и servernode:

```
set ns [new Simulator]
source tb_compat.tcl
```

```
set clientnode [$ns node]
set servernode [$ns node]

set link [$ns duplex-link $clientnode $servernode 100Mbps 0ms DropTail]

$ns rtproto Static
$ns run
```

На слици 4 се види слика топологије која је аутоматски генерисана помоћу овог кода који описује ову једноставну топологију.



Слика 4: Изглед ове топологије, слику генерише DETERLab

Линије `set clientnode [$ns node]` и `set servernode [$ns node]` дефинишу нове чворове назване clientnode и servernode. Може се приметити да имена чворова добијају име по називу промењиве у којој се први пут сачувају `[$ns node]` референце чвора. Али, уколико се користи низ чворова (као што је најчешће случај) као на пример:

```
for {set i 0} {$i < 4} {incr i} {
   set nodes($i) [$ns node]
}
```

У овом случају имена чворова се неће звати nodes(0), ..., nodes(3), него ће називи бити измењени тако да не садрже заграде како би могли бити коришћени у оквиру DNS домена. Значи ови чворови ће се звати nodes-0, ..., nodes-3. Приступ чворовима је могућ коришћењем додељене DNS адресе у формату <име чвора>.<име експеримента>.<име пројекта> [2].

Линија `set link [$ns duplex-link $clientnode $servernode 100Mbps 0ms DropTail]` прави нови уређај на оба чвора, спаја их каналом (физичким или виртуелним) који је постављен између clientnode и servernode чворова, и додељује им IP адресу тако да могу међусобно да комуницирају. Додатни параметри су максимална брзина и кашњење. DropTail је механизам реда чекања (queue) који у принципу функционише као обичан ред чекања али је ограничене величине и у случају да се преплави ред чекања, најстарији пакети се губе и преко њих се преписују нови пакети у принципу кружних бафера. Остатак команди се може видети на сајту DETERLab-а [1]. Експеримент се може покренути преко сајта или

преко команде:

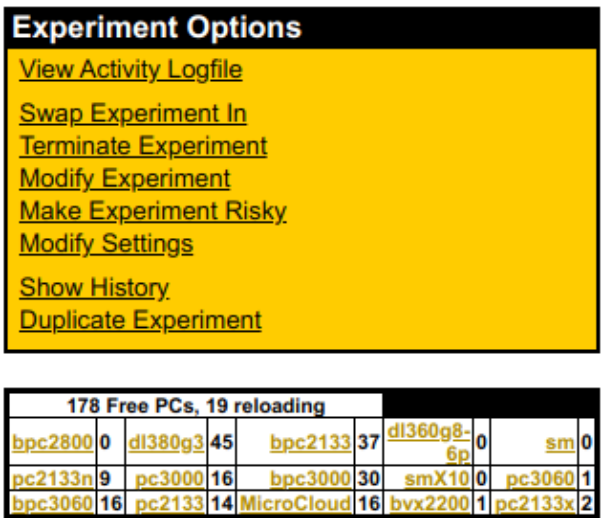
```
export PATH=$PATH:/usr/testbed/bin/
# pravljenje novog eksperimenta
$ startexp -p <project> -e <experiment> -l 240 -a 600 -f <topology-file>

# pokretanje eksperimenta - zauzimanje resursa (swapin)
$ swapexp <project> <experiment> in

# prekidanje eksperimenta - oslobađanje resursa (swapout)
$ swapexp <project> <experiment> out
```

Ове команде за рад са експериментима се налазе у фолдеру ”/usr/testbed/bin/” који иначе није укључен у активну путању ”$PATH”, па је пре коришћења тих команди потребно укључити ту путању (`export PATH=$PATH:/usr/testbed/bin/`) или те команде позивати са пуном путањом нпр. /usr/testbed/bin/startexp. Приликом покретања експеримента (swapin), након што се сачека да се експеримент потпуно учита (након око 10 минута), чвор се може приступити са адресом у облику (<node>.<experiment>.<project>[.isi.deterlab.net]). Пример пинговања једног чвора (нпр. clientnode) на подигнутој мрежи експеримента који се зове ”simple-client-server”.

```
ping clientnode.simple-client-server.dostrace
```

где је ”dostrace” име пројекта, ово име је DNS назив који по DNS стандарду није осетљив на велика и мала слова (case-sensitive), па зато иако је назив пројекта великим словима ”DOSTRACE”, може се навести са малим као што је наведено овде. Затим ”simple-client-server” је име експеримента, а ”clientnode” је име чвора, дефинисано у topology.tcl датотеци где се дефинише топологија. Чворовима је аутоматски додељена DNS адреса.[2]

```
$ ssh <node>.<experiment>.<project>[.isi.deterlab.net]
```



Слика 5: Изглед поља на сајту DETERLab-а где се виде доступни рачунари

На сајту DETERLab-а се може видети листа слободних типова рачунара који су на располагању. У топологији је могуће подесити који тип рачунара да се користи у експерименту:

```
# veći izbor mogućih računara
tb-make-soft-vtype container0 {MicroCloud dl380g3}
tb-set-hardware $clientnode container0
# ili jedan tip
tb-set-hardware $clientnode MicroCloud
```

Приликом бирања групе рачунара за коришћење, треба узети у обзир да се неки од ових група међусобно налазе на различитим локацијама и спојени су брзим линком, али рачунари на истој локацији имају најбржу везу и најмање кашњење. Конектовање на DETERLab главни сервер који је повезан са осталим машинама.

```
$ ssh <username>@users.isi.deterlab.net
```

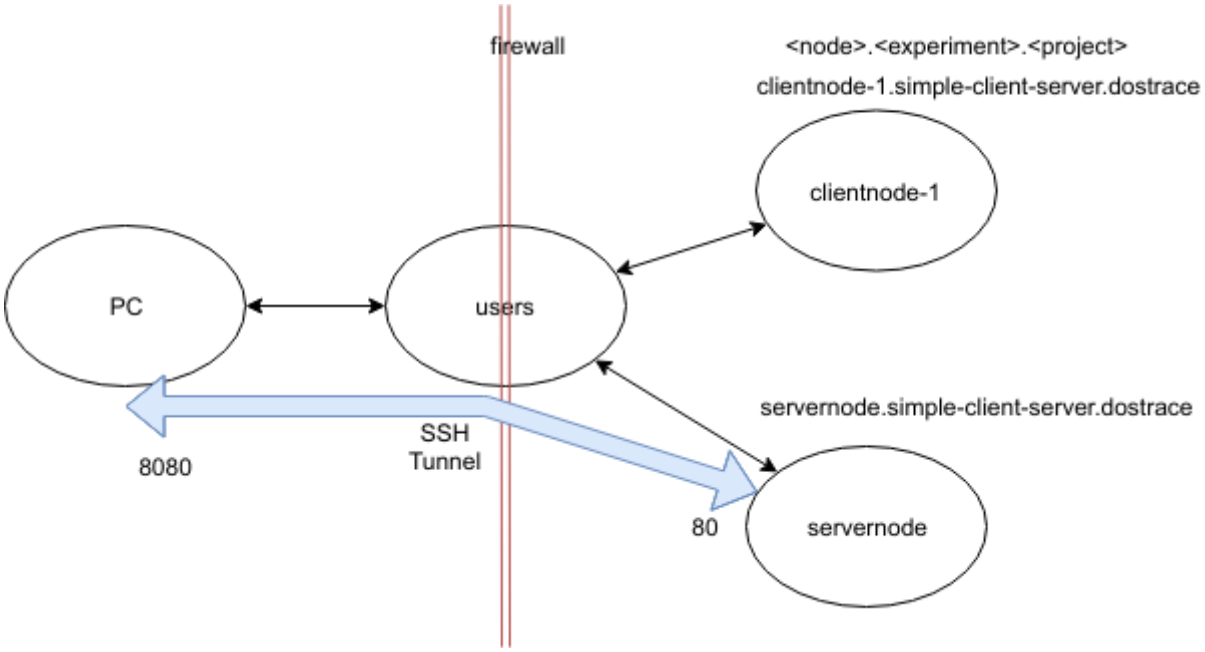
Након повезивања на главни сервер, могуће је повезати се са чворовима. Ово је једини начин повезивања са чворовима јер се они налазе ван интернет мреже, иза firewall-а.

```
$ ssh <node>.<experiment>
```

Директно повезивање са рачунарима иза firewall-а није могуће, али постоји могућност да се направи SSH тунел [18] којим се омогућава повезивање са одабраним портом на машини као на слици 6. На пример да би порт 2000 на локалном рачунару проследио податке ка порту 80 (HTTP порт) на чвору унутар експеримента, потребно је извршити команду:

```
$ ssh <username>@users.isi.deterlab.net -L \
  2000:servernode.simple-client-server.dostrace:80 -N
```

Помоћу овог могуће је приступити на пример apache web серверу који је дигнут на једном чвору у експерименту.



Слика 6: SSH конекција

```
ftnunsaa@clientnode:~$ ifconfig
eth2      Link encap:Ethernet  HWaddr 00:0e:0c:68:a7:21
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          inet6 addr: fe80::20e:cff:fe68:a721/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4 errors:0 dropped:0 overruns:0 frame:0
          TX packets:15 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:560 (560.0 B)  TX bytes:1466 (1.4 KB)

eth3      Link encap:Ethernet  HWaddr 00:11:43:d5:f4:eb
          inet addr:192.168.1.99  Bcast:192.168.3.255  Mask:255.255.252.0
          inet6 addr: fe80::211:43ff:fed5:f4eb/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5187 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1086 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6885764 (6.8 MB)  TX bytes:160536 (160.5 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:48 errors:0 dropped:0 overruns:0 frame:0
          TX packets:48 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:7364 (7.3 KB)  TX bytes:7364 (7.3 KB)
```

Слика 7: ifconfig на clientnode чвору

На слици 7 види се да чвор clientnode има 3 уређаја (lo, eth2, eth3) од чега је lo увек присутан localhost уређај, док је eth3 веза са главним сервером и clientnode чвором, а eth2 је веза између clientnode и servernode чворова.

```
ftnunsaa@clientnode:~$ ping servernode
PING servernode-link (10.1.1.3) 56(84) bytes of data.
64 bytes from servernode-link (10.1.1.3): icmp_seq=1 ttl=64 time=0.304 ms
64 bytes from servernode-link (10.1.1.3): icmp_seq=2 ttl=64 time=0.455 ms
^C
--- servernode-link ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.304/0.379/0.455/0.077 ms
ftnunsaa@clientnode:~$
ftnunsaa@clientnode:~$ ping servernode.basicexp
ping: unknown host servernode.basicexp
ftnunsaa@clientnode:~$ ping servernode.basicexp.dostrace
PING pc134.isi.deterlab.net (192.168.1.134) 56(84) bytes of data.
64 bytes from pc134.isi.deterlab.net (192.168.1.134): icmp_seq=1 ttl=64 time=0.205 ms
64 bytes from pc134.isi.deterlab.net (192.168.1.134): icmp_seq=2 ttl=64 time=0.105 ms
```

Слика 8: ping servernode и ping servernode.basicexp

На слици 8 може се видети разлика у IP адреси у односу на DNS адресу која се пингује. Иако је адреса servernode.basicexp (која се користи на users машини) доступна овде, та 192.x.x.x адреса се користи између users машине и чворова, и не треба се користити јер преоптерећује мрежу коју други корисници исто користе (мрежа са главним сервером). Адреса која се користи за комуникацију између чворова је типа 10.x.x.x.

2.2 Оркестрација (MAGI Orchestrator) - извршење сложеног експеримента

Рад са мрежама често подразумева тестирање са великим бројем чворова и сложеном топологијом. Без програма попут MAGI оркестратора [4, 3], било би потребно ручно повезати се на сваки чвор помоћу SSH протокола и покренути све потребне апликације. Све ове апликације би требале некако да се подесе да међусобно комуницирају, и због великог броја чворова (нпр. 100 чворова), ово постаје веома мукотрпно и неефикасно. Са MAGI оркестратором се може изразити и аутоматизовати процедурални ток рада за експеримент и тако обезбедити детерминистичку контролу над различитим компонентама у експерименту. AAL (Agent Activation Language) је описни језик базиран на YAML описном језику који MAGI користи да опише процедурални ток извршења експеримента. Цела процедура извршења експеримента мора се изразити као део orchestrate.aal датотеке која се састоји из 3 описна дела:

1. дефиниција групе чворова (groups):

Углавном постоји више чворова који врше исту улогу, па се по том принципу и у

оркестратору дефинишу групе, које се користе за доделу агената.

```
groups:
   client_group: [clientnode-1,clientnode-2]
   server_group: [servernode]
```

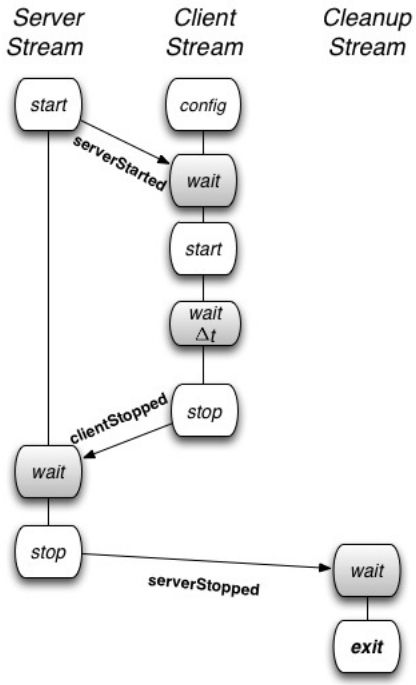
2. дефиниција агента (agents):

   Помоћу секције ”agents” се дефинише шта ће која група чворова да ради и које апликације ће бити покренуте на њима. Могуће је покренути више апликација на једном чвору, тако што се дефинише више агента који припадају истој групи чворова. На овом примеру клијенти морају посећивати сајт сервера, а сервери морају дићи тај сајт. То се описује тако што се дефинише клијентски агент ”client_agent” који покреће ”http_client” апликацију којој се додељују почетни аргументи (execargs).

```
agents:
   client_agent:
      group: client_group
      path: /share/magi/modules/http_client/http_client.tar.gz
      execargs: {servers: [servernode], interval: '5', sizes:
          'minmax(1000,10000)'}
   server_agent:
      group: server_group
      path: /share/magi/modules/apache/apache.tar.gz
      execargs: []
```

3. дефиниција токова догађа (eventstreams):

   У претходном кораку су дефинисани агенти који су покренути на одређеним чворовима али још увек не врше никакву функционалност. Та функционалност се описује у овом делу преко токова оркестрације који се извршавају редом од горе на доле. Ове токове је сад потребно дефинисати по принципу као на слици 9. Токови који се извршавају су ”serverstream”, ”clientstream” и ”cleanupstream”:



Слика 9: Ток догађаја који треба да се опише у овом експерименту

```
eventstreams:
  serverstream:
    - type: event
      agent: server_agent
      method: startServer
      trigger: serverStarted
      args: {}

    - type: trigger
      triggers: [ { event: clientStopped} ]

    - type: event
      agent: server_agent
      method: stopServer
      trigger: serverStopped
      args: {}

  clientstream:
    - type: trigger
      triggers: [ { event: serverStarted } ]

    - type: event
      agent: client_agent
      method: startClient
      args: {}

    - type: trigger
      triggers: [ { timeout: 60000 } ]

    - type: event
      agent: client_agent
```

```
      method: stopClient
      trigger: clientStopped
      args: {}

  cleanupstream:
    - type: trigger
      triggers: [ {event: serverStopped, target: exit} ]
```

Ови токови још нису покренути, и потребно их је навести у ”streamstarts” секцији како би били покренути на почетку извршавања тока догађаја.

```
streamstarts: [ serverstream, clientstream, cleanupstream ]
```

На почетку серверског тока је наведен догађај којим се покреће метода ”startServer” из агента ”server_agent” и емитује се сигнал (trigger) ”serverStarted” који покреће клијентски ток (јер клијентски ток чека на тај trigger `triggers: [ { event: serverStarted } ]`) а затим чека на клијента да сигнализира сигнал ”clientStopped”. Након што прими тај сигнал којег клијент емитује, он стопира серверског агента позивајући методу ”stopServer” из агента ”server_agent” и притом емитује сигнал ”serverStopped” на који ”cleanupstream” чека након чега се завршава експеримент (требало би извршити припрему и паковање резултата за преузимање).

Унутар eventstream токова могу се наћи 2 типа наредбе где једна служи да шаље сигнал одређеним чворовима, а друга чека да прими сигнал од свих чворова који могу да емитују тај сигнал:

1. event (- type: event) - шаље сигнал:

   Event врши неку акцију (догађај), нека метода се позива и опционо се емитује неки сигнал (trigger). Уколико су 2 догађаја један за другим, извршиће се истовремено, уколико је потребно да се изврше секвенцијално, потребно је убацити trigger између њих. Између токова се такође врши синхронизација.

2. trigger (- type: trigger) - чека на сигнал:

   Trigger паузира извршавање тока док ”event” не емитује ”trigger”.

```
- type: event
agent: client_agent
method: startClient
trigger: clientStarted
args: {}
```

Како би се оркестрација могла извршити над чворовима, пре покретања претходне команде, потребно је покренути magi_bootstrap.py на свим чворовима у мрежи, а то се постиже додавањем команде у датотеку за топологију (topology.tcl).

```
set magi_start "sudo python /share/magi/current/magi_bootstrap.py"
tb-set-node-startcmd $clientnode "$magi_start"
tb-set-node-startcmd $servernode "$magi_start"
```

Па ће топологија са изменама изгледати овако:

```
set ns [new Simulator]
source tb_compat.tcl

set clientnode [$ns node]
set servernode [$ns node]

set magi_start "sudo python /share/magi/current/magi_bootstrap.py"
tb-set-node-startcmd $clientnode "$magi_start"
tb-set-node-startcmd $servernode "$magi_start"
set link [$ns duplex-link $clientnode $servernode 100Mbps 0ms DropTail]

$ns rtproto Static
$ns run
```

Оркестратор се покреће са командом на контролном серверу:

```
$ python /share/magi/current/magi_orchestrator.py -p <project> -e <experiment> \
   --events <aal-path>
```

```
04-23 04:36:55.356 magi.orchestrator.orchestrator INFO     Running Event Streams
stream serverstream   : sent : (04:36:55) startServer(None) --> server_group  (fires trigger: serverStarted)
stream cleanupstream  : sent : (04:36:55) stopCmd(None) --> attack_group
stream clientstream   : trig : (04:36:57) trigger completed: serverStarted: {'retVal': True}
stream clientstream   : sent : (04:36:57) execute(['cleanup']) --> monitor_group  (fires trigger: cleaned_up)
stream clientstream   : trig : (04:36:58) trigger completed: cleaned_up: {'retVal': True}
stream clientstream   : sent : (04:36:58) startCollection(['',  '-z gzip -C 20 ... ) --> monitor_group  (fires trigger: sta
rt_attack)
stream clientstream   : sent : (04:36:58) startExperiment(None) --> attack_group
stream clientstream   : sent : (04:36:58) startClient(None) --> client_group
stream attackstrea... : trig : (04:37:03) trigger completed: start_attack: {'retVal': True}
stream attackstrea... : trig : (04:37:13) trigger completed: timeout: 10
stream attackstrea... : trig : (04:37:13) trigger completed: timeout: 0
stream attackstrea... : jump : (04:37:13) Execution path of stream has jumped to target: attackstream.
stream attackstream   : sent : (04:37:13) startCmd(None) --> attack_group
stream attackstream   : trig : (04:37:15) trigger completed: timeout: 2
stream attackstream   : sent : (04:37:15) stopCmd(None) --> attack_group
stream attackstream   : trig : (04:38:16) trigger completed: timeout: 60
stream attackstream   : trig : (04:38:16) trigger completed: timeout: 0
stream attackstream   : jump : (04:38:16) Execution path of stream has jumped to target: attackstream.
stream attackstream   : sent : (04:38:16) startCmd(None) --> attack_group
stream attackstream   : trig : (04:38:18) trigger completed: timeout: 2
```

Слика 10: Терминал са исписом из оркестратора

Лог фајлови и подаци MAGI оркестратора се налазе на сваком од чворова на путањи /var/log/magi:

```
$ ls /var/log/magi
config  db  logs
$ ls /var/log/magi/db
mongodb
```

2.2.1 Могући проблеми при извршењу MAGI оркестратора

Један од проблема који се може јавити јесте да база података која се налази на првом чвору порасте и до 500MB, и тиме заузме сав простор на диску који је по default-у 2GB.

```
# problem, nedostatak prostora na disku glavnog čvora
ftnunsaa@clientnode-1:~$ df -h
Filesystem    Size Used Avail Use% Mounted on
/dev/simfs    2.0G 2.0G  25M 99% /

# razlog
ftnunsaa@clientnode-1:~$ du -sh /var/log/magi/db/
545M /var/log/magi/db/

ftnunsaa@clientnode-1:~$ du -sh /tmp/
301M /tmp/

# treba isto primetiti da se /tmp/ ne nalazi u ram memoriji
ftnunsaa@clientnode-1:~$ df -h
Filesystem    Size Used Avail Use% Mounted on
/dev/simfs    2.0G 2.0G  25M 99% /
/dev/simfs    1.9T 1.1T 684G 62% /users/ftnunsaa
/dev/simfs    2.8T 1.3T 1.3T 50% /proj/DOSTRACE
/dev/simfs    485G 231G 216G 52% /share
none          256M 4.0K 256M  1% /dev
none          4.0K    0  4.0K  0% /sys/fs/cgroup
none           52M 1.1M  51M  2% /run
none          5.0M    0  5.0M  0% /run/lock
none          256M    0  256M  0% /run/shm
none          100M    0  100M  0% /run/user

# sto znači 545M + 301M + system = 2GB (popunjem prostor)
```

Ово значи да уколико пукне magi_daemon.py на првом чвору, ово је један од разлога. Као решење је обрисати фолдер /var/log/magi/db и покренути magi_bootstrap.py поново.

```
sudo rm -r /var/log/magi/db
sudo python /share/magi/current/magi_bootstrap.py
```

2.2.2 Алати MAGI оркестратора

Користећи magi_status.py могуће је проверити стање MAGI daemon-а на чворовима и рестартовати их по потреби:

```
# provera stanja daemon-a
$ python /share/magi/current/magi_status.py -p <project> -e <experiment> \
   -a <aal-path>

# restartovanje daemon-a na clientnode1 i clientnode2
$ python /share/magi/current/magi_status.py -p <project> -e <experiment> \
   -a <aal-path> -r -n clientnode1,clientnode2
```

MAGI оркестратор омогућава и плотовање дијаграма користећи magi_graph.py алат. Пре плотовања потребно је направити фајл са конфигурацијом плот дијаграма:

```
  graph:
    type: line
    xLabel: Time(sec)
    yLabel: Bytes
    title: Traffic plot
  db:
    agent: monitor_agent
    filter:
        host: servernode
        peerNode: clientnode
        trafficDirection: out
    xValue: created
    yValue: bytes
```

Затим је потребно покренути magi_graph.py:

```
$ python /share/magi/current/magi_status.py -p <project> -e <experiment> \
   -a <agent-name> -c graph_config.cfg
```

2.2.3 Прављење агента за оркестратор - пример агента "minimal_process_agent"

Агент је програм којег оркестратор покреће на чворовима и има контролу над извршењем одређених метода дефинисаних у агенту. Све што оркестратор ради је покреће агенте на чворовима и контролише ток њиховог извршења тако што позива методе које агент омогућава. Агенти се помоћу одговарајућег API-ја могу писати у више програмских језика: python, C, C++.

За почетак, потребно је направити нову датотеку названу minimal_process_agent.idl, ово је датотека YAML формата и у њему се дефинишу подаци попут назива агента, детаљан опис агента, начин покретања агента (да ли као процес или као нова нит у оквиру оркестратора), подржана верзија MAGI оркестратора на којој се може агент покренути, име датотеке која се покреће приликом стартовања агента, и дефиниција методе које имплементира агент.

minimal_process_agent.idl:

```
name: ExampleProcessAgent
magi_version: 1.7
display: ExampleProcessAgent 1.2
description: this agent does nothing
execute: thread
mainfile: minimal_process_agent.py
inherits:
    - DispatchAgent

methods:
    - name: startProcessAgent
      help: starts agent
      args:
      - name: param1
        type: string
```

```
      help: arguments to cmd

    - name: param2
      type: string
      help: arguments to cmd
```

minimal_process_agent.py:

```
import sys

from magi.util.agent import DispatchAgent, agentmethod
from magi.util.processAgent import initializeProcessAgent

class ExampleProcessAgent(DispatchAgent):
   def __init__(self):
      DispatchAgent.__init__(self)

   @agentmethod()
   def startProcessAgent(param1, param2):
      return

   @agentmethod()
   def confirmConfiguration(self):
      return True

def getAgent(**kwargs):
   agent = example_process_agent()
   agent.setConfiguration(None, **kwargs)
   return agent

if __name__ == "__main__":
   agent = example_process_agent()
   kwargs = initializeProcessAgent(agent, sys.argv)
   agent.setConfiguration(None, **kwargs)
   agent.run()
```

getAgent је неопходна функција коју оркестратор користи за инстанцирање агента, као и confirmConfiguration функција која се позива након што су self промењиве класе учитане и служи ради омогућавања провере ограничења неких промењивих, на пример нека вредност излази ван опсега или није доброг типа и слично. Уколико confirmConfiguration врати вредност False, оркестрација се прекида и исписује се одређена порука грешке. Класа агента готово увек наслеђује класу DispatchAgent која у петљи чека поруку која тражи позив неке методе и позива ту методу са датим параметрима, дакле служи као основни механизам позива удаљене методе (eng. Remote Procedure Call, RPC).

Све датотеке агента се затим упакују у .tar.gz архиву са командом:

```
$ tar -czf ../minimal process_agent.tar.gz *
```

Овај датотека minimal process_agent.tar.gz се даље укључује у оркестратор на овај начин:

```
agents:
  client_agent:
   group: client_group
```

```
path: /users/<username>/process_agent.tar.gz
execargs: {}
```

3. Помоћна скрипта

3.1 Идеја и принцип рада

Рад директно преко SSH конекције са DETERLab рачунаром је представљао проблем јер је потребно често преносити датотеке са локалног рачунара на DETERLab рачунар и обрнуто. То је било могуће коришћењем sshfs или rsync алата, али је преузимање резултата са чворова и даље било прилично споро јер је веза чворова са контролним рачунаром и чворовима прилично спора. Експериментисањем на разне начине показало се да је брже скидати фајлове директно са чворова на локални рачунар уместо прво са чворова на контролни рачунар, али је команда за ово прилично дугачка:

```
rsync -azvvP --ignore-missing-args -e "ssh -A $ssh_addr ssh -o
    StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null"
    "$user@$node.$exp.$proj":/tmp/output .
```

Без скрипте, било је потребно повезати се са SSH алатом на контролни рачунар тј. на адресу users.isi.deterlab.net, пренети фајлове са локалног рачунара на кориснички користећи SFTP протокол (scp или rsync алат), и затим покренути експеримент преко оркестратора и затим преузети резултате са одређених чворова. Иако ово звучи једноставно, овај процес непотребно одузима драгоцено време и концентрацију која може бити искоришћена на планирање даљег процеса експеримента или нешто друго.

Коришћењем скрипте приметио сам знатно повећање продуктивности у ради са DETERLab-ом, као и могућност лаке репродукције експеримента од стране нових корисника DETERLab-а.

Цео процес и потреба за аутоматизацијом мотивисан је широко примењеном аутоматизацијом процеса компајлирања (Makefile, cmake, ant и сл.), и поред постојећег алата попут MAGI оркестратора који је веома користан, осећа се потреба за још једним алатом који олакшава целокупан процес извршења експеримента.

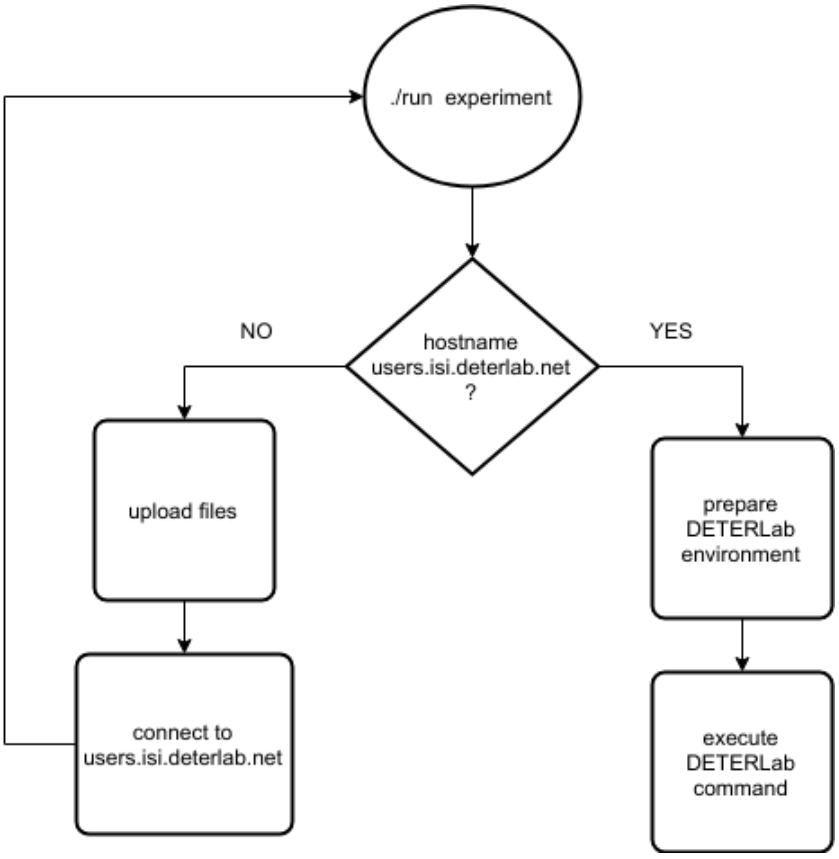
Све ово је омогућено јер се све датотеке експеримента групишу у један фолдер у

којем се налазе све информације о томе како да се аутоматски изврши експеримент. Па је у пар једноставних команди могуће извршити експеримент:

```
# kreiranje eksperimenta na DETERLab-u
./run make_exp
# priprema za izvršenje eksperimenta (potrebno je sačekati oko 10 minuta)
./run swapin
sleep 10m
# izvršenje eksperimenta i preuzimanje rezultata
./run experiment
```

Све ове команде се извршавају на локалном рачунару што значи да није потребно водити рачуна о преносу фајлова са разних места ради извршења експеримента или чувања резултата.

Рад на локалном рачунару је много бржи од рада преко интернета, на DETERLab рачунару који се налази на другом континенту што представља главни разлог кашњења у испису на терминал при раду преко SSH конекције. Након што се експеримент изврши, са свих одабраних чворова ће резултати бити аутоматски преузети, што значи да је овиме омогућен лак и брз начин да се врше кратки експерименти са пуно различитих варијација.



Слика 11: Принцип рада скрипте

На слици 11 је приказан основни принцип рада скрипте. Види се да се исти код извршава на локалном рачунару са којег се синхронизује цео садржај овог фолдера где

је експеримент на DETERLab контролни рачунар, и затим се повезује на тај рачунар и на њему се покреће исти овај код који се извршава на другачији начин јер препознаје да се налази на users.isi.deterlab.net рачунару тако што проверава његово име на мрежи (hostname).

3.2 Коришћење

Програми од којих зависи функционалност ове скрипте:

1. линукс оперативни систем

   Скрипта је предвиђена да ради на линукс систему на којем се веома лако инсталира. Али постоји могућност да ради и на windows систему, али је теже подесити због специфичних линукс програма од којих зависи. На систему windows 10 постоји подршка за линукс подсистем преко којег би се ова скрипта врло лако покренула. зависи извршење.

2. bash интерпретер

   Извршитељ скрипте. На виндовсу bash је доступан у оквиру cygwin или git bash програмског пакета.

3. ssh (openssh)

   Врши конекцију са корисничким рачунаром. На windows рачунару је доступан у оквиру windows 10 од 2015. године, а као алтернатива је доступан програм Putty.

4. rsync

   Алат који врши пренос података преко SFTP протокола и притом користи диференцирање и компресију ради убрзања преноса података.

Скрипта се састоји од фајлова:

1. topology.tcl

   Садржи опис топологије експеримента.

2. experiment.sh

   Информације о експерименту које се користе у осталим фајловима експеримента.

3. gen-aal.sh

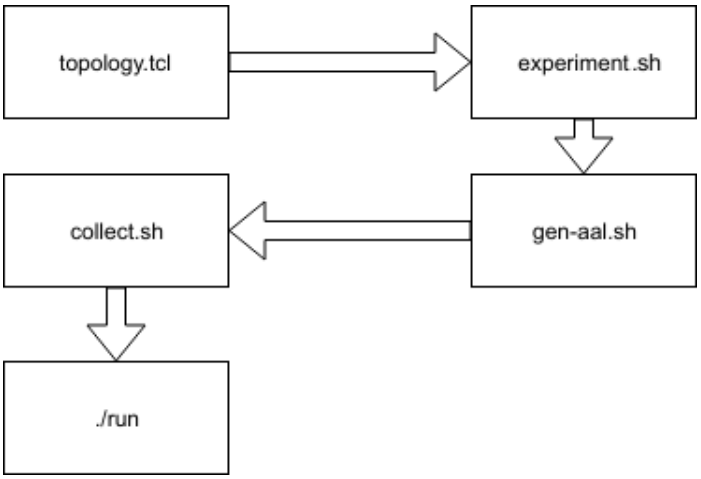
   Генерише topology.aal датотеку која служи за оркестрацију експеримента.

4. collect.sh

   Врши конверзије и компресије резултата пред преузимање.

5. run

скрипта која садржи команде за покретање експеримента



Слика 12: Кораци прављења експеримента са скриптом

Уобичајени кораци за прављење експеримента користећи ову скрипту су дати на слици 12. Прво се пише опис топологије у topology.tcl фајлу где се задаје број чворова, изглед мреже и атрибути мреже у експерименту. Ово је фајл којег DETERLab користи за генерисање информација о броју потребних чворова, њихов начин повезивања и правило за доделу IP адреса и подешавања табеле рутирања како би чворови могли међусобно да комуницирају на начин како је описано у topology.tcl датотеци. Ова датотека се често пише са екстензијом .ns јер је компатибилна са NS2 мрежним симулатором од којег и потиче.

У следећем кораку се врши измена experiment.sh фајла који садржи константе које се деле између свих bash скрипти, а те константе су подаци о повезивању на DETERLab машину (корисничко име, име пројекта и име експеримента) и информације о чворовима (имена чворова које се користе у експерименту нпр. attackers=”attackernode-10, attackernode-11, attackernode-12”).

Затим измена gen-aal.sh датотеке подразумева дефинисање понашања чворова који су дефинисани у топологији. Чворови се групишу по функционалности и додељују им се одређени агенти (програми који се извршавају на овим чворовима), и врши се опис извршења експеримента.

Како би резултати аутоматски били преузети, потребно их је сместити у директоријум /tmp/output/. Овај процес смештања и припреме резултата за преузимање се извршава у collect.sh скрипти коју претходно gen-aal.sh скрипта море додати као агента свим чворовима од којих се преузима резултат извршења експеримента.

```
groups:
   ...
agents:
   # dodavanje agenta ($datadir/cmd.tgz je agent za izvršenje komande date kao
       parametar)
   collect_agent:
      group: monitor_group
      path: $datadir/cmd.tgz
      execargs: {cmd: '$expdir/collect.sh'}

eventstreams:
   ...

   cleanupstream:
      # na kraju toka se dodaje izvršenje collect.sh agenta
      - type: event
        agent: collect_agent
        method: execute
        trigger: collection_done
        args: {}
```

3.2.1 Датотека experiment.sh

У датотеци experiment.sh потребно је дефинисати све дељене константе са другим bash скриптама као на пример collect.sh и gen-aal.sh. Овде се дефинише корисничко име за ssh, име пројекта, име експеримента, избор коришћења виртуелних машина у сврху повећања искоришћења физичких ресурса.

Уколико се користи виртуелизација тј. контејнеризација (./run make_cont_exp), могуће је сместити око 10-15 виртуелних машина на једну физичку машину. Овде се дефинишу и дељене промењиве као на пример групе чворова и друге:

```
export user=ftnunsaa
export proj=DOSTRACE
export exp=dumbbell-lan-big
# ukoliko se koristi kontejnerizovani eksperiment (make_cont_exp) uneti parametre za
    generisanje eksperimenta
# --packing => broj virtuelnih mašina po jednoj fizičkoj
# --pnode-type => lista tipova fizičkih mašina koje će se koristiti
export containerize_params="--pnode-type MicroCloud,dl380g3 --packing=12"
export name='lan-big-7'

# koji pristup da se koristi za generisanje AAL datoteke
export gen_aal=gen-aal.sh # gen-aal.sh or gen-aal.py or topology.aal

# definicije čvorova
export clients=`join clientnode-{1..95}`
export attackers=`join clientnode-{96..100}`
export servers=`join servernode-{1..5}`
# <=> servernode-1,servernode-2,servernode-3,...,servernode-5

export tcpdump_expr=""

export routers=router-1,router-2
export some_clients=`join clientnode-{90..95}`
```

```
export monitor=$some_clients,$routers,$attackers,$servers
export nodes=$clients,$routers,$attackers,$servers
```

Промењивама routers, monitor, nodes дефинисане су логичке групе чворова које врше одређену улогу дефинисану у AAL фајлу. Ти чворови су размакнути зарезима како би то одговарало YAML запису листе (јер је AAL заправо YAML формат), а ове промењиве ће бити коришћене у gen-aal.sh скрипти која генерише topology.aal датотеку приликом покретања експеримента. Функција "join" је помоћна функција која раздваја задате параметре зарезом, а param-{1..5} је bash правило које генерише аргументе који су нумерисани:

```
$ echo param-{1..5}
param-1 param-2 param-3 param-4 param-5
```

Комбиновањем функције join и овог правила могуће је скраћеним записом дефинисати имена чворова која се користе у gen-aal.sh фајлу. На пример:

```
servers=$(join servernode-{1..5})
# ovo je ekvivalentno sa:
servers="servernode-1,servernode-2,servernode-3,servernode-4,servernode-5"
```

Ова експанзија је корисна у примеру промењиве clients, јер линија:

```
clients=$(join clientnode-{1..95})
```

генерише јако велики број clientnode референци, па је на овај начин лако подешавати број клијената који се понаша нормално и издвојити чворове који ће имати улогу нападача:

```
attackers=$(join clientnode-{96..100})
```

3.2.2 Датотека gen-aal.sh

Ово је једноставна bash скрипта која користи промењиве из gen-aal.sh за генерисање AAL фајла. Овде се дефинишу групе чворова, агенти који се покрећу на тим групама чворова од којих зависи улога чворова. На пример, неки чворови имају улогу сервера и на њима се покреће серверски програм apache, на клијентским чворовима се покреће http_client који се повезује са сервером и врши неке захтеве серверу. На рутерима се покреће collect_agent који записује саобраћај који се после може (аутоматски) преузети и проучити користећи друге алате (попут wireshark-а или неки други за обраду резултата). Често експеримент садржи велики број чворова, па је потребно уносити велики број назива чворова у групу ручно у topology.aal датотеку:

```
groups:
   client_group: [
          clientnode-1, clientnode-2, clientnode-3, clientnode-4,
          clientnode-5, clientnode-6, clientnode-7, clientnode-8,
          clientnode-9, clientnode-10, clientnode-11, clientnode-12,
          clientnode-13, clientnode-14, clientnode-15, clientnode-16,
```

```
        clientnode-17, clientnode-18, clientnode-19, clientnode-20,
        clientnode-21, clientnode-22, clientnode-23, clientnode-24,
        clientnode-25, clientnode-26, clientnode-27, clientnode-28,
        clientnode-29, clientnode-30, clientnode-31, clientnode-32,
        clientnode-33, clientnode-34, clientnode-35, clientnode-36,
        clientnode-37, clientnode-38, clientnode-39, clientnode-40,
        clientnode-41, clientnode-42, clientnode-43, clientnode-44,
        clientnode-45, clientnode-46, clientnode-47, clientnode-48,
        clientnode-49, clientnode-50 ]
  server_group: &slist [ servernode-1, servernode-2, servernode-3, servernode-4,
      servernode-5 ]
```

Па је због овога много лакше аутоматски генерисати AAL фајл са gen-aal.sh скриптом која учитава све вредности промењивих из experiment.sh и користи те вредности да генерише тај фајл.

```
groups:
  client_group: &clist [$clients]
  monitor_group: [$monitor]
  server_group: &slist [$servers]
  attack_group: [$attackers]
```

Пошто је topology.aal датотека YAML формата, који подржава референцирање листе као овде са &clist, и &slist, на другом месту би користили *clist и *slist за дереференцирање вредности.

Пример gen-aal.sh скрипте:

```
#!/usr/bin/env bash
[ $# == 1 ] || exit
source experiment.sh

cat > $1 <<EOF
# grupe se definišu lako korišćenjem promenjivih
# $clients, $monitor, $servers, itd... učitanih iz experiments.sh fajla
groups:
  client_group: &clist [$clients]
  monitor_group: [$monitor]
  server_group: &slist [$servers]
  attack_group: [$attackers]

agents:

  client_agent:
    # client_agent se pokreće na client_group čvorovima
    group: client_group
    # $mods <=> /share/magi/modules , $mods je promenjiva definisana u experiments.sh
        datoteci
    path: $mods/http_client/http_client.tar.gz
    # parametri za http_client
    execargs: {servers: *slist, interval: '1', sizes: 'minmax(1000,1000)'}

   # agent koji se pokreće nad čvorovima monitor grupe
   # koji pokreće program za prikupljanje saobraćaja koji
   # protiče kroz te čvorove
  monitor_agent:
```

```
      group: monitor_group
      path: $mods/tcpdump/tcpdump.tar.gz
      execargs: {}

    # agent koji se pokreće na serverskim čvorovima je apache server
    server_agent:
      group: server_group
      path: $mods/apache/apache.tar.gz
      execargs: {}

    # koristi cmd agent za kojeg je dat kod u primeru izvršenja eksperimenta
    collect_agent:
      group: monitor_group
      path: $datadir/cmd.tgz
      execargs: {cmd: 'bash ./collect.sh', cwd: '$expdir'}

streamstarts: [ serverstream, clientstream, cleanupstream ]

eventstreams:

   serverstream:
     # pokretanje servera (apache server)
     - type: event
       agent: server_agent
       method: startServer
       trigger: serverStarted
       args: {}

     # čekanje na završetak klijentskih operacija posećivanja servera
       - type: trigger
         triggers: [ { event: clientStopped} ]

     # zaustavljanje servera (apache server)
       - type: event
         agent: server_agent
         method: stopServer
         trigger: serverStopped
         args: {}

  clientstream:

     # čekanje na početak rada servera
       - type: trigger
         triggers: [ { event: serverStarted } ]

       # čišćenje ostataka od prošlog izvršavanja eksperimenta
       - type: event
         agent: collect_agent
         method: execute
         trigger: cleanedUp
         args: { cmdargs: 'preprocess' }

     # čekanje da se očiste prethodni rezultati od prošlog eksperimenta (tj. da se
         završi prethodna komanda)
       - type: trigger
         triggers: [ { event: cleanedUp } ]

       # početak snimanja saobraćaja
```

```
    - type: event
      agent: monitor_agent
      method: startCollection
      args: { expression: '$tcpdump_expr', tcpdump_args: '-z gzip -C 200' }

    # označavanje vremena početka napada
    - type: event
      agent: attack_agent
      method: startExperiment

    # klijenti počinju posećivanje servera
    - type: event
      agent: client_agent
      method: startClient
      trigger: clientStarted
      args: {}

   # trajanje eksperimenta je 10 sekundi
    - type: trigger
      triggers: [ { timeout: 10000 } ]

    # prekid rada klijenata (prekid posećivanja servera)
    - type: event
      agent: client_agent
      method: stopClient
      trigger: clientStopped
      args: {}

    # prekid kolekcije saobraćaja
    - type: event
      agent: monitor_agent
      method: stopCollection
      trigger: collectionStopped
      args: {}

  cleanupstream:
    # čekanje na završetak stopCollection naredbe (koja emituje signal
        collectionStopped)
    - type: trigger
      triggers: [ { event: collectionStopped } ]

    # pozivanje skripte collect.sh koja pakuje rezultate i priprema ih za preuzimanje
    - type: event
      agent: collect_agent
      method: execute
      trigger: collectionDone
      args: {cmdargs: 'postprocess'}

    # nakon završenog prikupljanja rezultata, završava se eksperiment (target: exit)
    - type: trigger
      triggers: [ { event: collectionDone, target: exit } ]
EOF
```

Оператор ”« EOF” значи да се испод уноси текст који мора да се заврши кључном речи EOF, јер је та реч задата у оператору «. Ово такође значи да у нормалном тексту не сме да се појави реч EOF јер би то значило завршетак уноса текста.

```
cat > $1 <<EOF
...
EOF
```

3.2.3 Датотека gen-aal.py

Овај начин генерисања AAL датотеке је осмишљен из истог разлога као и разлог за коришћење ген-aal.sh, али се овде иде корак даље, уместо само коришћења промењивих дефинисаних у experiment.sh, овде се још додатно користи python при генерисању догађаја. Мислим да је грешка што је уведен YAML за описивање догађаја и да је од почетка требао бити направљен овакав један API који би омогућио лакши и прегледнији начин описивања тока догађаја. Пошто у оркестратору не постоји опција да се експеримент описује у python-у, одлучио сам направити додатни алат који ово омогућава.

За писање једноставнијих AAL датотека или експеримената који се неће пуно мењати ради различитих параметара експеримената, ова скрипта не значи пуно, али у случају сложених експеримената се види повећање продуктивности.

```
clients=repl('$clients').split(',')
monitor=repl('$monitor').split(',')
servers=repl('$servers').split(',')
attackers=repl('$attackers').split(',')

# GROUPS
client_group = groups.new('client_group', clients)
monitor_group = groups.new('monitor_group', monitor)
server_group = groups.new('server_group', servers)
attack_group = groups.new('attack_group', attackers)

# AGENTS
client_agent = agents.new('client_agent', client_group,
    '$mods/http_client/http_client.tar.gz',
   {'servers': servers, 'interval': '1', 'sizes': 'minmax(1000,1000)'})

monitor_agent = agents.new('monitor_agent', monitor_group,
    '$mods/tcpdump/tcpdump.tar.gz')
server_agent = agents.new('server_agent', server_group, '$mods/apache/apache.tar.gz')
collect_agent = agents.new('collect_agent', monitor_group, '$datadir/cmd/',
   execargs={'cmd': 'bash ./collect.sh', 'cwd':'$expdir'})
attack_agent = agents.new('attack_agent', attack_group, path='$datadir/cmd.tgz',
   execargs={'cmd': '$datadir/syn 10.0.1.1 80 1', 'mark_time': 1 })

with stream('serverstream', start=True):
    # pokretanje servera (apache server)
   server_agent('startServer', trigger='serverStarted')

   # čekanje na završetak klijentskih operacija posećivanja servera
   trigger('clientStopped')

   # zaustavljanje servera (apache server)
   server_agent('stopServer', trigger='serverStopped')
```

```
with stream('clientstream', start=True):
   # čekanje na početak rada servera
   trigger('serverStarted')

   # čišćenje ostataka od prošlog izvršavanja eksperimenta
   collect_agent('execute', {'cmdargs':'preprocess'}).wait()

   # početak snimanja saobraćaja
   monitor_agent('startCollection',
      { 'expression': '$tcpdump_expr', 'tcpdump_args': '-z gzip -C 200' },
      trigger='startAttack')

   # označavanje vremena početka napada
   attack_agent('startExperiment')

   # klijenti počinju posećivanje servera
   client_agent('startClient', trigger='clientStarted')

   # trajanje eksperimenta je 10 sekundi
   trigger(10)

   # prekid rada klijenata (prekid posećivanja servera)
   client_agent('stopClient', trigger='clientStopped')

   # prekid kolekcije saobraćaja
   monitor_agent('stopCollection', trigger='collectionStopped')

with stream('cleanupstream', start=True):

   # čekanje na završetak stopCollection naredbe (koja emituje signal
       collectionStopped)
   trigger('collectionStopped')

   # pozivanje skripte collect.sh koja pakuje rezultate i priprema ih za preuzimanje
   collect_agent('execute', {'cmdargs':'postprocess'})

   # nakon završenog prikupljanja rezultata, završava se eksperiment (target: exit)
   trigger('serverStopped','exit')
```

Овај код представља исти експеримент као што је приказан у претходној секцији за gen-aal.sh датотеку. Види се да је код у python језику, много компактнији, прегледнији и лакши за рад, користећи функције и све погодности програмског језика.

Позивање ”.wait” нпр. collect_agent(’execute’, ’cmdargs’:’preprocess’).wait() у овом случају дефинише trigger=’trig0’ овој операцији и затим иза ове операције додаје trigger(’trig0’), уколико је trigger већ претходно дефинисан, такав какав је дифинисан ће бити коришћен.

Дакле код:

```
collect_agent('execute', {'cmdargs':'preprocess'}, trigger='preprocessDone').wait()
```

је скраћена верзија и представља исто што и:

```
collect_agent('execute', {'cmdargs':'preprocess'}, trigger='preprocessDone')
# čekanje na prethodnu komandu
```

```
trigger('preprocessDone')
```

У коду испод, приказане су веома корисне могућности које нуди gen-aal.py:

```
def on_off_attacks(lst):
   for (a,b) in lst:
      attack_agent('startCmd')
      trigger([[a], ['clientStopped', 'stopattack']])
      attack_agent('stopCmd')
      trigger([[b], ['clientStopped', 'stopattack']])
   # ovaj tok se može pojaviti samo jednom u generisanom AAL, jer će ponovno
       generisanje biti odbačeno
   with stream('stopattack'):
      attack_agent('stopCmd')

with stream('attackstream_start', start=True):
   trigger('start_attack')
   trigger(8, 'attackstream2')

   # ovaj tok neće biti generisan, jer ne postoji referenca na njega
   with stream('attackstream'):
      on_off_attacks( [(5,25), (5,50)] )
      loop()

   # ovaj tok će biti generisan jer trigger(8, 'attackstream2') pokazuje na njega
   with stream('attackstream2'):
      on_off_attacks( [(15,80), (5, 30), (3, 10), (15,50)] )
      loop()
with stream('changestream', start=True):
   import random
   trigger('clientStarted')
   for i in range(10):
      client_agent('changeTraffic', args={'stepsize':random.randint(100,1000)})
      trigger(random.randint(1,10))
      loop()
```

Овде је дефинисана функција on_off_attacks која омогућава лако генерисање напада у шаблону трајање напада и трајање паузе између напада. Па је у attackstream2 приказан напад који траје 15 секунди, па затим 80 секунди траје пауза, па следећи напад траје 5 секунди и 30 секунди пауза, и тако даље.

Додатна могућност је генерисање наредби са random вредностима. Иако неки агенти омогућавају унос дистрибуције случајне промењиве и тиме омогућавају случајни унос, на овај начин је случајни унос омогућен и за промењиве којима то није дозвољено, као у овом случају.

Функција loop() може примити параметар колико секунде да чека пре него што скочи назад на почетак тренутног stream-a, и ова функција замењује trigger(0, 'changestream').

Још једна интересантна ствар за напоменути овде је да ток који није назначен као почетни start=True односно то значи да се не налази у streamstarts AAL директиви која се аутоматски генерише:

```
streamstarts: [ attackstream_start, changestream ]
```

Уколико се неки ток дефинише више пута као на пример у позиву функције on_off_attacks ток 'stopattack' у том случају биће дефинисан само онај који је први пут извршен, док ће поновна дефиниција истог тока бити игнорисана. Ово омогућава да се дефинишу опциони токови који се појављују у употреби само при коришћењу ове функције.

Једноставном променом промењиве gen_aal тако да показује на неку датотеку са екстензијом *.py, омогућена је примена оваквог типа скрипте за оркестрацију.

```
export gen_aal=gen-aal.py
```

3.2.4 Датотека collect.sh

Датотека collect.sh служи за прикупљање резултата након извршења експеримента, на пример врши компресију pcap фајлова на чворовима и записује IP адресе на чворовима које олакшавају идентификацију појединих клијената и сервера у експерименту.

```
#!/usr/bin/env bash
cmd=$1
shift
. run

case $cmd in
   preprocess)
      cd /tmp/
      rm -rf output
      rm -f *.cap* *.gz *.json collect.log
      ;;
   postprocess)
      cd /tmp/
      find -regextype sed -regex '.*\.cap[0-9]*' -exec gzip {} \;
      mkdir -p output
      mv *.gz *.txt output/
      ip addr > output/ipaddr.txt
      ;;
esac
```

На крају се ови резултати аутоматски прикупљају са rsync алатом којег покреће ./run скрипта.

3.2.5 Извршавање команди

Унутар датотеке "run" се налази цео код скрипте који није потребно мењати и омогућава олакшан рад са DETERLab-ом нудећи бројне пречице за честе случајеве коришћења DETERLab-а.

1. `./run make_exp`

   Прављење експеримента који није контејнеризован

2. `./run make_cont_exp`

Прављење контејнеризованог експеримента

3. `./run swapin`

   Покретање експеримента (овом командом се врши заузимање ресурса на DETERLab-у, потребно је сачекати бар 10 минута док се чворови не покрену).

   Команда `./run swapout` врши контра акцију, ослобађање ресурса DETERLab-а на коришћење другим корисницима.

4. `./run experiment`

   Уплоадовање, покретање експеримента и преузимање резултата. Уколико експеримент не постоји, треба га направити прво са `./run make_exp` или `./run make_cont_exp` .

5. `./run upload`

   Upload-овање података са локалног рачунара на DETERLab. Ова команда се имплицитно позива при скоро свакој команди којој је то потребно, па је није потребно експлицитно користити.

6. `./run download [<node><@group>]`

   Преузимање резултата са чвора.

```
./run download clientnode-1
```

   Или преузимање резултата са групе чворова:

```
./run download @monitor
```

7. `./run ssh [<node-name><@group>] [<command>] [shell]`

   Ова команда за руковање SSH конекцијама је веома корисна и вишеструка.

```
./run ssh
```

   SSH Конектовање на контролни сервер у интерактивном режиму, притом се аутоматски учитавају промењиве из experiment.sh датотеке и путања до testbed програма. Додавањем још једног параметра, врши се конекција на одређени чвор, опет у интерактивном режиму.

```
$ ./run ssh clientnode-12
[local/ssh] ssh to node clientnode-12: bash -i
ftnunsaa@clientnode-12:/tmp$
```

   Извршавање команди над једним или више чворова, приказани су разни примери коришћења.

```
# dodavanje nove rute na router-2 čvoru
./run ssh router-2 "sudo ip route add 10.0.0.0/24 via 10.0.2.2 dev eth2"
# dodavanje brisanje rute na router-1 i router-2 čvorovima
./run ssh router-1,router-2 "sudo ip route del 10.0.0.0/8 via 10.0.2.2 dev eth2"
# clients je promenjiva definisana u experiment.sh datoteci, sa @ operatorom je
    naznačena
./run ssh @clients "bash ~/skripta.sh"
# pokretanje magi_bootstrap.py bez unosa startne komande u topologiju
./run ssh @nodes "sudo python /share/magi/current/magi_bootstrap.py"
# ubijanje DoS napada koji je izmakao kontroli
./run ssh @attackers "pkill syn_flood"
```

На овај начин се може послати команда групи чворова који су дефинисани у датотеци experiments.sh, и ова команда је корисна уколико је потребно ручно покренути неку команду. Овако се може на брзину тестирати неки програм или покренути неку скрипту на свим чворовима, рецимо за прекид експеримента уколико се није прекинуо како треба и негде је застој.

Такође понекад се дешава да на неким чворовима пукне magi_daemon.py из разлога описаних у секцији 2.2.1 па неће да крене оркестрација експеримента, постоји могућност да се поново покрене коришћењем ове команде:

```
./run ssh @nodes "sudo pkill mongodb;\
  sudo rm -r /var/log/magi/db;\
  sudo python /share/magi/current/magi_bootstrap.py"
```

Поред самог слања команди, могуће је и исписати резултат тих команди додавајући на команду реч shell чиме се форсира испис излазних порука у терминал.

На пример да се испишу сви активни портови на свим серверима (@servers) могуће је искористити команду:

```
# ispis aktivnih portova na svim serverima
$ ./run ssh servernode-1,servernode-2 "sudo netstat -ltn" shell
[local/ssh] ssh to node servernode-1: sudo netstat -ltn
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address     Foreign Address      State
tcp       0      0 0.0.0.0:22          0.0.0.0:*            LISTEN
tcp       0      0 0.0.0.0:25          0.0.0.0:*            LISTEN
tcp6      0      0 :::22               :::*                 LISTEN
tcp6      0      0 :::25               :::*                 LISTEN
tcp6      0      0 :::80               :::*                 LISTEN

[local/ssh] ssh to node servernode-2: sudo netstat -ltnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address     Foreign Address      State
tcp       0      0 0.0.0.0:22          0.0.0.0:*            LISTEN
tcp       0      0 0.0.0.0:25          0.0.0.0:*            LISTEN
tcp6      0      0 :::80               :::*                 LISTEN
tcp6      0      0 :::22               :::*                 LISTEN
tcp6      0      0 :::25               :::*                 LISTEN
```

Приликом извршавања ових ssh команди на више чворова, те команде се извршавају паралелно 5 одједном тј. врши се повезивање са 5 чворова истовремено и затим се резултат прикупља и исписује се на крају. Овиме се значајно убрзава извршење команди на много чворова истовремено, уместо један по један.

8. `./run portforward <node> <local-port> <remote-port>`

   Прави SSH тунел ка чвору који се налази иза firewall-а.

4. Пример извршења експеримента

Како би цео поступак извршења експеримента био потпуно јасан, овде је детаљно описан процес извршења експеримента и анализа резултата у експерименту који се састоји од клијената, сервера и нападача који врше SYN Flood DDoS напад.

Нападачки програм [17] који врши SYN Flood DDoS напад је писан у језику C, и не може да се користи директно из оркестратора јер не користи C API за комуникацију са оркестратором, него се мора користити агент који омогућава извршење екстерних програма. Па иако већ постоји постојећи агент за извршење екстерних команди, потребан је специјализовани агент који записује времена почетка напада и престанка напада као и могућност прекида напада из оркестратора, методом stopCmd.

Времена напада се касније користе у евалуацији алгоритама за детекцију DDoS напада за утврђивање исправности и кашњења детекције напада. Код за агента који се користи у овом експерименту је дат испод.

cmd_agent.py:

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

import logging
import os, sys, time
import subprocess, signal
from magi.testbed import testbed
from magi.util.agent import DispatchAgent, agentmethod
from magi.util.processAgent import initializeProcessAgent
import magi.util.execl as execl

class cmd_agent(DispatchAgent):
   """
   starts/stops command
   """
   def __init__(self):
      DispatchAgent.__init__(self)
      self.pids = []
```

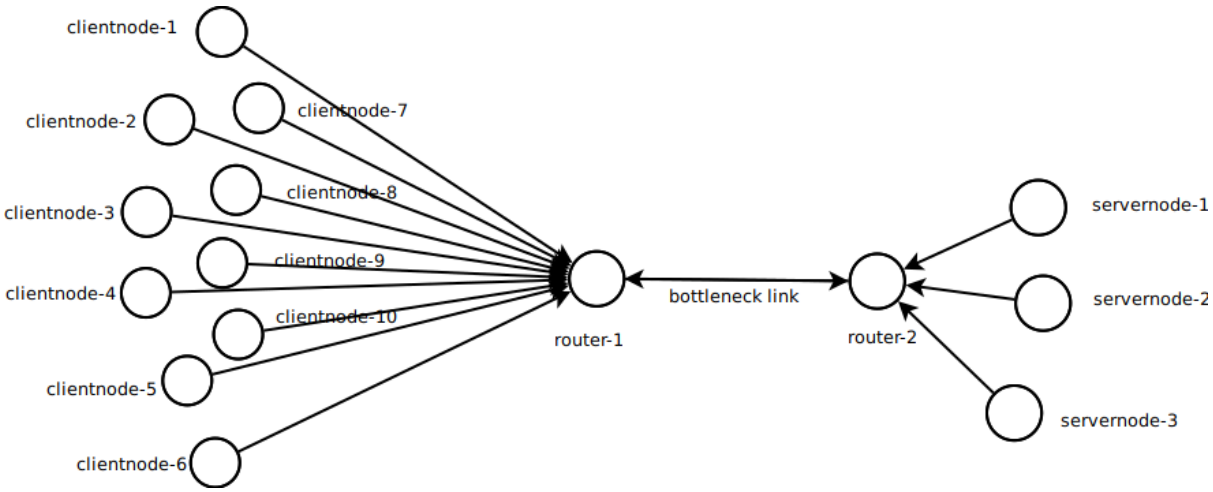
```
    self.maxCmds=1
    self.cmd = ''
    self.cwd = os.getcwd()
    self.log = logging.getLogger(__name__)
    self.ofs = time.time()
    self.times = {'start':[], 'end':[]}
    self.mark_time = 0
    self.mark_path = '/tmp/output/attack-times.csv'

@agentmethod()
def execute(self, msg, cmdargs=''):
    cmd = self.cmd + ' ' + args
    self.log.info('Running cmd: %s' % cmd)
    output = execl.execAndRead(cmd, cwd=self.cwd)[0]
    self.log.info('Output: ' + output)
    return True

@agentmethod()
def startCmd(self, msg, cmdargs=''):
    cmd = self.cmd + ' ' + args
    if len(self.pids) < self.maxCmds:
        self.pids.append(execl.spawn(cmd, cwd=self.cwd, close_fds=True, shell=False))
        self.log.info('start cmd: ' + cmd + ' PID: ' + str(self.pids[-1]))
    if self.pids:
        self.times['start'].append( time.time()-self.ofs )
    return True

@agentmethod()
def startExperiment(self, msg):
    self.ofs = time.time()

@agentmethod()
def stopCmd(self, msg):
    self.log.info('stopCmd() called, PIDS ' + str(self.pids))
    for pid in self.pids:
        self.log.info('stopCmd() killing %s' % pid)
        try:
            os.kill(pid, signal.SIGKILL)
        except:
            self.log.info('stopCmd() killing %s FAILED' % pid)
            raise

    if self.pids:
        self.times['end'].append( time.time()-self.ofs )
        self.pids = []
    return True

@agentmethod()
def stop(self, msg):
    self.stopCmd(msg)
    DispatchAgent.stop(self, msg)
    if self.mark_time:
        with open(self.mark_path,'w') as f:
            try:
                f.write(','.join(['%.2f' % a for a in self.times['start']])+'\n')
                f.write(','.join(['%.2f' % a for a in self.times['end']])+'\n')
            except Exception as e:
                f.write('some exception: ' + str(e))
```

```
   @agentmethod()
   def confirmConfiguration(self):
      return True

def getAgent(**kwargs):
   agent = cmd_agent()
   agent.setConfiguration(None, **kwargs)
   return agent

if __name__ == "__main__":
   agent = cmd_agent()
   kwargs = initializeProcessAgent(agent, sys.argv)
   agent.setConfiguration(None, **kwargs)
   agent.run()
```

Методе:

- execute покреће екстерну команду и чека да се извршење те команде заврши.
- startCmd покреће екстерну команду са параметрима и бележи време покретања, команда наставља да се извршава асинхроно све док се не заустави са stopCmd или stopExperiment.
- startExperiment означава покретање експеримента тако што запише временску референцу у односу које се рачуна време извршења команди које се записује у attack-times.csv датотеци.
- stopCmd зауставља извршавање команде која је претходно покренута са startCmd методом.
- stopExperiment прекида извршење команде уколико још увек траје и записује времена напада ”attack-times.csv”.
- getAgent је обавезна метода сваког агента за инцијализацију.

Топологија које се користи у овом експерименту је dumbbell топологија, и приказана је на слици 13.



Слика 13: Топологија која је описана у овом експерименту

Код за топологију је испод и објашњен је коментарима:

```
# standardni deo ns koda
set ns [new Simulator]
source tb_compat.tcl

# početna komanda koja se pokreće na svakom čvoru kako bi se mogla koristiti
# magi orkestracija na tim čvorovima
set magi_start "sudo python /share/magi/current/magi_bootstrap.py"

# broj klijenata i servera
set num_clients 100
set num_servers 5

# instanciranje 2 rutera koji će biti spojeni jednim kanalom
# router(1) će biti povezan sa klijentima, a router(2) sa serverima
set router(1) [$ns node]
set router(2) [$ns node]

# početne komande na oba rutera
tb-set-node-startcmd $router(1) "$magi_start"
tb-set-node-startcmd $router(2) "$magi_start"

set c_lan ""
set s_lan ""

# leva strana (klijenti)
for {set i 1 } {$i <= $num_clients } { incr i } {
   set clientnode($i) [$ns node]
   # početna komanda na klijentima
   tb-set-node-startcmd $clientnode($i) "$magi_start"
   append c_lan "$clientnode($i) "
}

# desna strana (serveri)
for {set i 1 } {$i <= $num_servers } { incr i } {
   set servernode($i) [$ns node]
   # početna komanda na serverima
   tb-set-node-startcmd $servernode($i) "$magi_start"
```

```
   append s_lan "$servernode($i) "
}

# LAN mreža za klijente
set lanClients [$ns make-lan "$router(1) $c_lan" 100Mb 0ms]

# LAN mreža za servere
set lanServers [$ns make-lan "$router(2) $s_lan" 100Mb 0ms]

# dupleks veza između dva rutera
set linkRouters [$ns duplex-link $router(1) $router(2) 100Mb 0ms DropTail]

# standarni deo ns koda
$ns rtproto Static
$ns run
```

Операције које треба да се изврше над одговарајућим чворовима у експерименту су задате у gen-aal.sh датотеци:

```
#!/usr/bin/env bash
[ $# == 1 ] || exit
. experiment.sh

cat > $1 <<EOF
# ovde se definišu grupe čvorova koje će posle biti referencirane
groups:
  client_group: &clist [$clients]
  monitor_group: [$monitor]
  server_group: &slist [$servers]
  attack_group: [$attackers]

agents:

  # agent za klijente koji vrši zahteve serveru za veb stranicu
  # određene veličine
  client_agent:
    group: client_group
    path: $mods/http_client/http_client.tar.gz
    execargs: {servers: *slist, interval: '5', sizes: 'minmax(500,10000)'}

  # agent koji se pokreće nad čvorovima monitor grupe
  # koji pokreće program za prikupljanje saobraćaja koji
  # protiče kroz te čvorove
  monitor_agent:
    group: monitor_group
    path: $mods/tcpdump/tcpdump.tar.gz
    execargs: { }

  # agent koji se pokreće na serverskim čvorovima je apache server
  server_agent:
    group: server_group
    path: $mods/apache/apache.tar.gz
    execargs: []

  # koristi cmd agent za koji je prethodno dat kod
  collect_agent:
    group: monitor_group
```

```
    path: $datadir/cmd.tgz
    execargs: {cmd: '$expdir/collect.sh'}

  # agent napadača koji se pokreće na napadačkoj grupi (attack_group)
  # takođe koristi cmd agent koji pokreće program "syn"koji vrši napad
  attack_agent:
    group: attack_group
    path: $datadir/cmd.tgz
    execargs: { cmd: '$datadir/syn --attack-ips 10.0.0.xxx --ip 10.0.1.2 --port 80
      --threads 1 --duration 3 --attack-sleep 10000', mark_time: 1 }

streamstarts: [ serverstream, clientstream, cleanupstream, attackstream_start ]

eventstreams:

  serverstream:
     # pokretanje servera (apache server)
     - type: event
       agent: server_agent
       method: startServer
       trigger: serverStarted
       args: {}

     # čekanje na završetak klijentskih operacija posećivanja servera
     - type: trigger
       triggers: [ { event: clientStopped} ]

     # zaustavljanje servera (apache server)
     - type: event
       agent: server_agent
       method: stopServer
       trigger: serverStopped
       args: {}

  clientstream:
      # čišćenje ostataka od prošlog izvršavanja eksperimenta
      - type: event
        agent: collect_agent
        method: execute
        trigger: cleaned_up
        args: { args: 'cleanup' }

     # čekanje na početak rada servera
     - type: trigger
       triggers: [ { event: serverStarted } ]

      # čekanje da se očiste prethodni rezultati od prošlog eksperimenta
      - type: trigger
        triggers: [ { event: cleaned_up } ]

      # početak snimanja saobraćaja
      - type: event
        agent: monitor_agent
        trigger: start_attack
        method: startCollection
        args: { expression: '$tcpdump_expr', tcpdump_args: '-z gzip -C 200' }

      # početak beleženja vremena za napadače
```

```
    - type: event
      agent: attack_agent
      method: startExperiment
      args: {}

    # klijenti počinju posećivanje servera
    - type: event
      agent: client_agent
      method: startClient
      args: {}

   # trajanje eksperimenta je 120 sekundi
    - type: trigger
      triggers: [ { timeout: 120000 } ]

   # prekid rada klijenata (prekid posećivanja servera)
    - type: event
      agent: client_agent
      method: stopClient
      trigger: clientStopped
      args: {}

    # prekid kolekcije saobraćaja
    - type: event
      agent: monitor_agent
      method: stopCollection
      trigger: pcap_done
      args: {}

    # čekanje na završetak prethodne naredbe (koja emituje signal pcap_done)
    - type: trigger
      triggers: [ { event: pcap_done } ]

    # pozivanje skripte collect.sh koja pakuje rezultate i priprema ih za preuzimanje
    - type: event
      agent: collect_agent
      method: execute
      trigger: collection_done
      args: {}

attackstream_start:
   # čekanje na signal da napad može da počne
    - type: trigger
      triggers: [ { event: start_attack } ]

    # početno vreme čekanja je 10 sekundi
    - type: trigger
      triggers: [ { timeout: 10000 } ]

    # skok na stream ispod (attackstream)
    - type: trigger
      triggers: [ { timeout: 0, target: 'attackstream' } ]

attackstream:

    # početak napada
    - type: event
      agent: attack_agent
```

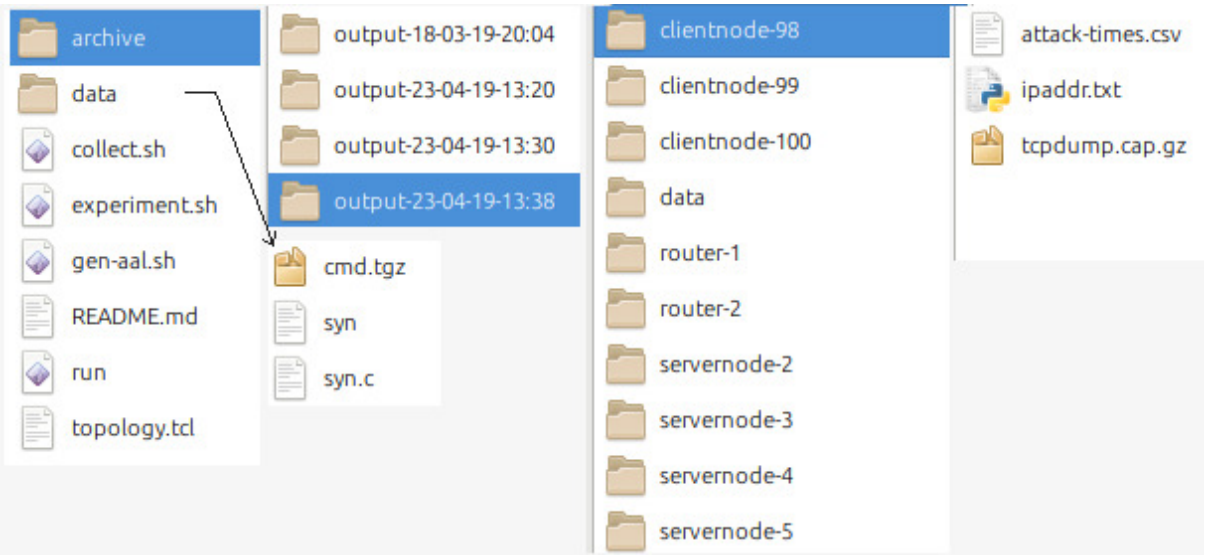
```
      method: startCmd
      args: {}

    # trajanje napada je 2 sekunde
    - type: trigger
      triggers: [ { event: clientStopped, target: cleanupstream },
       { timeout: 2000 } ]

    # prekid napada
    - type: event
      agent: attack_agent
      method: stopCmd
      args: {}

    # 60 sekundi razmak između napada
    - type: trigger
      triggers: [ { event: clientStopped, target: cleanupstream },
       { timeout: 60000 } ]

    # skok na početak ovog stream-a
    - type: trigger
      triggers: [ { timeout: 0, target: 'attackstream' } ]


  changestream:
    # promena jačine saobraćaja na slučajnu vrednost
    - type: event
      agent: client_agent
      method: changeTraffic
      args: { stepsize: 1000 }

   # svake sekunde se ponavlja ovaj "changestream"tok
    - type: trigger
      triggers: [ { timeout: 1000 } ]

    # skok na početak ovog toka
    - type: trigger
      triggers: [ { target: 'changestream' } ]

  cleanupstream:
    # prekid napada ukoliko još uvek traje
    - type: event
      agent: attack_agent
      method: stopCmd
      args: {}

    # čekanje na završenje obrade rezultata nakon čega se završava eksperiment
    - type: trigger
      triggers: [ {event: collection_done, target: exit},
       {event: serverStopped, target: exit} ]
################################################
EOF
```

Након извршења команде ”./run swapin” потребно је да прође око 10 минута да се чворови покрену (тад стиже e-mail са информацијом да је експеримент покренут), али чворови још нису спремни за комуникацију, па је потребно да прође још 10 минута да сви чво-

рови буду спремни за комуникацију, али ни ово још не значи да се експеримент може истог момента покренути, јер потребно је сачекати још неких 5 минута за иницијализацију MAGI bootstrap програма који омогућава оркестрацију експеримента. Што значи да се експеримент може покренути тек након неких 25 минута од swapin-a. Са командом "./run experiment" се покреће експеримент као што се и може видети на слици 14 испод.

```
04-23 04:36:55.356 magi.orchestrator.orchestrator INFO     Running Event Streams
stream serverstream   : sent : (04:36:55) startServer(None) --> server_group  (fires trigger: serverStarted)
stream cleanupstream  : sent : (04:36:55) stopCmd(None) --> attack_group
stream clientstream   : trig : (04:36:57) trigger completed: serverStarted: {'retVal': True}
stream clientstream   : sent : (04:36:57) execute(['cleanup']) --> monitor_group  (fires trigger: cleaned_up)
stream clientstream   : trig : (04:36:58) trigger completed: cleaned_up: {'retVal': True}
stream clientstream   : sent : (04:36:58) startCollection(['',  '-z gzip -C 20 ... ) --> monitor_group  (fires trigger: sta
rt_attack)
stream clientstream   : sent : (04:36:58) startExperiment(None) --> attack_group
stream clientstream   : sent : (04:36:58) startClient(None) --> client_group
stream attackstrea... : trig : (04:37:03) trigger completed: start_attack: {'retVal': True}
stream attackstrea... : trig : (04:37:13) trigger completed: timeout: 10
stream attackstrea... : trig : (04:37:13) trigger completed: timeout: 0
stream attackstrea... : jump : (04:37:13) Execution path of stream has jumped to target: attackstream.
stream attackstream   : sent : (04:37:13) startCmd(None) --> attack_group
stream attackstream   : trig : (04:37:15) trigger completed: timeout: 2
stream attackstream   : sent : (04:37:15) stopCmd(None) --> attack_group
stream attackstream   : trig : (04:38:16) trigger completed: timeout: 60
stream attackstream   : trig : (04:38:16) trigger completed: timeout: 0
stream attackstream   : jump : (04:38:16) Execution path of stream has jumped to target: attackstream.
stream attackstream   : sent : (04:38:16) startCmd(None) --> attack_group
stream attackstream   : trig : (04:38:18) trigger completed: timeout: 2
stream attackstream   : sent : (04:38:18) stopCmd(None) --> attack_group
stream clientstream   : trig : (04:38:58) trigger completed: timeout: 120
stream clientstream   : sent : (04:38:58) stopClient(None) --> client_group  (fires trigger: clientStopped)
stream clientstream   : sent : (04:38:58) stopCollection(None) --> monitor_group  (fires trigger: pcap_done)
stream attackstream   : trig : (04:38:59) trigger completed: clientStopped: {'retVal': True}
stream attackstream   : jump : (04:38:59) Execution path of stream has jumped to target: cleanupstream.
stream serverstream   : trig : (04:38:59) trigger completed: clientStopped: {'retVal': True}
stream serverstream   : sent : (04:38:59) stopServer(None) --> server_group  (fires trigger: serverStopped)
stream cleanupstream  : sent : (04:38:59) stopCmd(None) --> attack_group
stream serverstream   : DONE : (04:38:59) complete.
stream clientstream   : trig : (04:39:00) trigger completed: pcap_done: {'retVal': True}
stream clientstream   : sent : (04:39:00) execute(None) --> monitor_group  (fires trigger: collection_done)
stream clientstream   : DONE : (04:39:00) complete.
stream cleanupstream  : trig : (04:39:01) trigger completed: serverStopped: {'retVal': True}
stream cleanupstream  : jump : (04:39:01) Execution path of stream has jumped to target: exit.
04-23 04:39:01.353 magi.orchestrator.orchestrator INFO     Running Exit Streams
```

Слика 14: Испис терминала оркестратора

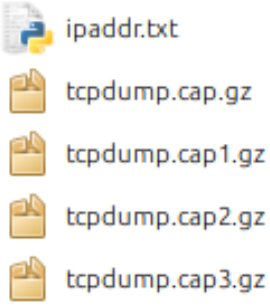


Слика 15: Структура директоријума експеримента и резултата извршења експеримента

На слици 15 се види изглед стабла директоријума експеримента, место чувања резултата и изглед тих резултата. У директоријуму Archive се налази листа резултата који су означени датумом и временом преузимања, а у тим директоријумима се налазе излазне датотеке од сваког од чвора чији резултати су преузети. На пример, на овој слици clientnode-98 представља нападача јер знамо из experiments.sh да нападаче чине чворови од са бројем од 96 до 100.

```
attackers=$(join clientnode-{96..100})
```

Резултујућа PCAP [10] датотека ”tcpdump.cap.gz” се даље обрађује са другим програмом. Уколико кроз неки чвор протиче јако велики број пакета, тако да величина PCAP датотеке надмашује величину од 200MB, онда се та датотека ”ротира” у gzip формат где се отвара велика компресија захваљујући томе што је садржај пакета у експерименту предвидив и репетитиван. Ово се ради из разлога што tcpdump имплементација снимања саобраћаја заузима RAM меморију, и ако би се цео PCAP датотеке снимао одједном, не би било довољно RAM меморије.



Слика 16: Листа tcpdump.cap*.gz датотека

На слици 16 може се видети делови PCAP фајлова који су упаковани у gzip. Како би обрада била могућа, ове датотеке се морају распаковати и спојити у једну PCAP датотеку која се после обрађује другим програмима. Ово спајање је могуће одрадити са програмом mergecap који долази са wireshark-ом или tcpdump алатом. Mergecap истовремено отпакује и спаја упаковане делове PCAP датотеке у једну велику PCAP датотеку.

```
$ mergecap tcpdump.cap.gz tcpdump.cap1.gz tcpdump.cap2.gz tcpdump.cap3.gz -w
    big-cap.cap
```

4.1 Обрада резултата експеримента

Резултати експеримента се даље обрађују статистичком анализом, обрадом ентропије разних делова саобраћаја. Ради обраде потребно је искористити претходно поменуту датотеку attack-times.csv у којој су записана времена кад су почињали и до кад су трајали напади. Један од алгоритама за детекцију DDoS напада је CUSUM алгоритам који користи ентропију као метрику на основу које се препознаје напад.

4.1.1 Ентропија

> “Entropy is the general trend of the universe toward death and disorder.”
>
> — James R. Newman

Ентропија је мера неуређености система. Што значи, што је неки систем више хаотичан (random) то је ентропија већа, а то значи да је код случајне промењиве униформне расподеле ентропија највећа.

(а) мала ентропија (б) велика ентропија

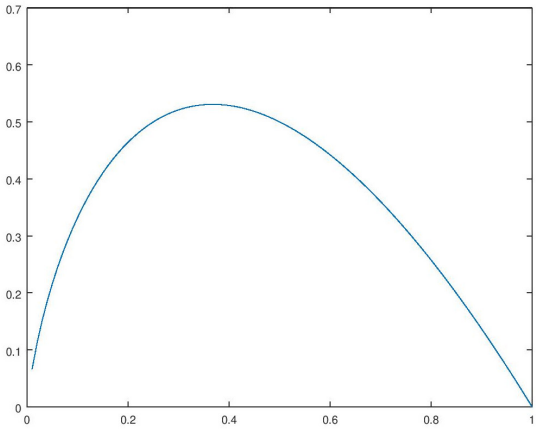
Слика 17: Пример хистограма различитих ентропија

Због ових особина, ентропија се користи у детекцији DoS напада јер се тип саобраћаја битно мења приликом DoS напада, питање је само да ли ће се променити однос количине нормалног и нападачког саобраћаја или ће се променити однос адреса на које пакети иду или са којих долазе, или неки други параметар у мрежи. Па сходно томе је јасно зашто је ентропија изабрана метода од стране многих истраживача детекције DoS напада [6, 8, 9, 14].

Формула за рачунање Shannon ентропије је:

$$H(P) = -\sum_{i=1}^{N} p_i log(p_i) \tag{1}$$

P - представља скуп свих вероватноћа и њихова сума мора бити једнака јединици.



Слика 18: Изглед функције f(p) = -p*log(p)

Постоји више дефиниција ентропије које се користе за различите сврхе:

1. У физици

   У термодинамици ентропија представља меру стања гаса, количину компресије, меру агрегатних стања, на пример чврсто агрегатно стање има малу ентропију јер је структура веома густо распоређена па је чврстог стања. $S = k_B ln\Omega$

2. У теорији информација

   Често се ентропија користи за оцену квалитета компресије података без губитака. Мала ентропија значи да се информација може доста компресовати и пренети са малом количином информација. Велика ентропија значи да се информација не може компресовати и мора се цела пренети. Такође се користи у оцени квалитета шифре која се користи за електронско потписивање и криптовање.

Види се да је значај ентропије велики и постоји много варијанти ентропије код којих је одређена варијанта погоднија за одређене ситуације.

Такође постоји још једна варијанта ентропије која се зове Tsalis [7, 14, 8] ентропија за коју неки аутори сматрају да је, због неких карактеристика мрежног саобраћаја, ова верзија ентропије погоднија и даје боље резултате од претходне, Shannon ентропије.

Формула за Tsalis ентропију је:

$$H(Z) = \frac{1 - \sum_{i=1}^{N} p_i^q}{q - 1} \tag{2}$$

Постоји још пар верзија ентропије које су истраживане у сврху детекције DoS напада [8, 9, 5]:

Формула за Ubriaco ентропију је:

$$H(Z) = \sum_{i=1}^{N} (-log(p(z_i)))^2 p(z_i) \tag{3}$$

Формула за Renyi ентропију је:

$$H(Z) = \frac{1}{1 - \alpha} log \sum_{i=1}^{N} p(z_i)^\alpha \tag{4}$$

Формула за Bhatia Singh ентропију је:

$$H(Z) = -\frac{1}{sinh(\alpha)} \Big(\sum_{i=1}^{N} p(z_i) sinh(\alpha log(p(z_i))) \Big) \tag{5}$$

4.1.2 Алгоритам за детекцију промена (CUSUM)

CUSUM (скр. CUmulative SUM) је алгоритам који омогућава детекцију промене у систему која довољно дуго одступа од нормалних вредности неког система. CUSUM [11] алгоритам се користи за детекцију TCP SYN Flood DDoS напада бројањем SYN пакета. Ова метода се показала јако ефикасна за детекцију SYN Flood напада али је ограничена само на такав тип напада. Како би ова метода била од веће користи уз одређене модификације могућа је детекција и остале велике већине DoS напада. Та модификација је у овом случају коришћење EWMA филтера над ентропијама разних појава у мрежном саобраћају (нпр. ентропија над портовима, IP адресама, величинама пакета, ентропија флегова пакета, ентропија над величином токова пакета (flow size distribution) [5], и још многих других особина које утичу на промену ентропије при DoS нападу).

Једначина за CUSUM алгоритам је:

$$\begin{aligned} &S_0 = 0, \\ &S_{n+1} = max(0, S_n + x_n - \omega_n) \end{aligned} \tag{6}$$

EWMA [6, 8] једначина:

$$S_t = \begin{cases} Y_1, & t = 1 \\ \alpha Y_t + (1-\alpha) S_{t-1}, & t > 1 \end{cases} \tag{7}$$

Комбиновањем ових једначина добија се верзија CUSUM алгоритма са EWMA ентропијом:

$$\begin{aligned} &\mu_n = \beta_1 y_n + (1-\beta_1)\mu_{n-1}, \mu_0 = y_0 \\ &d_n = max\{0, d_{n-1} + y_n - (\mu_n + K)\}, d_0 = 0 \\ &\sigma_n^2 = \beta_2 (y_n - \mu_n)^2 + (1-\beta_2)\sigma_{n-1}^2, H = h\sigma_n, \sigma_0 = 0 \end{aligned} \tag{8}$$

μ_n - представља средњу вредност промењиве y_n

σ_n - представља дисперзију тј. стандардну девијацију промењиве y_n

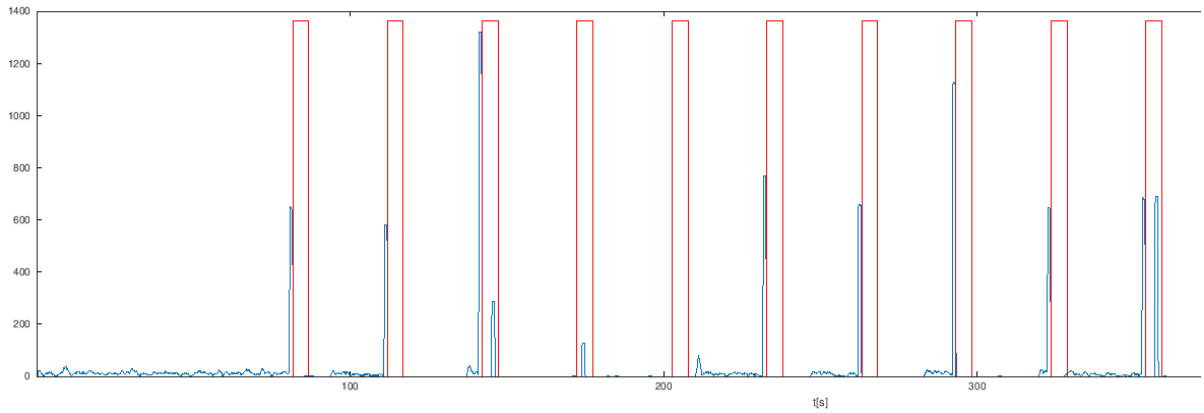
Кад је $d_n > H$, детектована је промена вредности која надмашује дозвољене границе и посматрајући ентропију пакета закључујемо да је велика вероватноћа да се ради о DoS нападу. Да би се промењива d_n повећала, потребно је да вредност y_n пређе вредност средње вредности претходних y_n вредности као и додатну тежину задату са промењивом K. Дакле ако промењива d_n већа од промењиве H то значи да је промена y_n довољно дуго била повишена да надмаши скалирану вредност дисперзије промењиве y_n, другим речима y_n је довољно расла да надмаши његову варијацију при нормалним условима која постоји кад нема DOS напада, што значи да постоји абнормално повећање ентропије тј. велика је вероватноћа да се ради о DOS нападу.

Параметри $\beta 1$ и $\beta 2$ означавају брзину промене средње вредности и дисперзије. Овакав концепт споријих промена средњих вредности (као инерција) се још назива и експоненцијално тежински кретање средње вредности (eng. Exponentially weighted moving average - EWMA), и представља једноставан филтер са бесконачним импулсним одзивом (eng. Infinite impulse response - IIR) у нископропусном режиму.

Matlab код за CUSUM алгоритам:

```
function r = interpolate(alpha, a, b)
  r = alpha * a + (1-alpha) * b;
```

```
end

function [detection, d, h] = cusum(samples, M)
   ewma_alpha1 = 0.2;
   ewma_alpha2 = 0.05;
   K = 0.3;
   s = 3;

   n = length(samples);
   d         = zeros(1, n);
   h         = zeros(1, n);
   detection = zeros(1, n);

   mi_a = samples(1);
   prev_d = 0;

   for j = 1:n
      sample = samples(j);
      mi_a = interpolate(ewma_alpha1, sample, mi_a);
      prev_d = max(0, prev_d + sample - (mi_a + K) );
      d(j) = prev_d;
      s = interpolate(ewma_alpha2, (sample - mi_a)^2, s);
      h(j) = M*sqrt(s);
      if d(j) > h(j)
         detection(j) = 100;
      end
   end
end
```



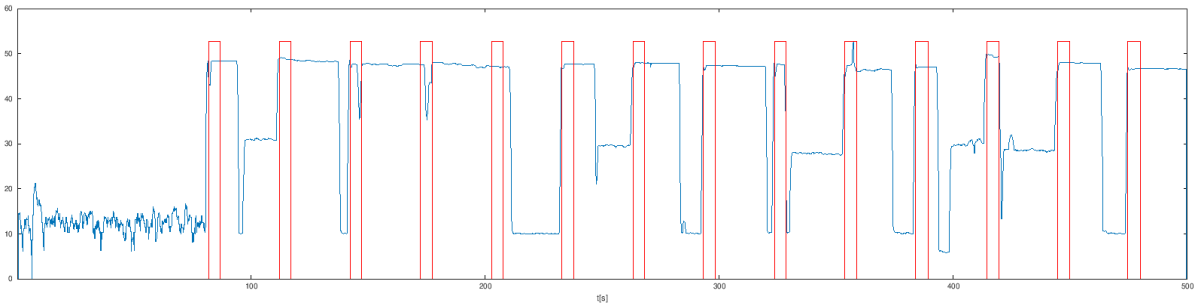
Слика 19: SYN пакети

Слика 19 представља SYN пакете који врше успостављање везе, а црвеном бојом су означени тренутци напада у којима се види нагли скок SYN пакета. У уводном делу, на слици 1 може се видети принцип напада, једино што је адреса ка којој се пакет одбија је случајна. Резултати експеримента (*.pcap датотеке) се прослеђују другом програму који врши обраду ових података.

Временска линија пакета се дели у интервале, тако да 10 интервала чине једну секунду. У сваком интервалу се бележе информације свих пакета који се нађу у том интервалу,

и обрада се врши коришћењем клизећег прозора где сваки прозор садржи 10 интервала.

```
// sliding window swap
// 1 2 [ 3 4 5 ] 6 7
// 1 2 3 [ 4 5 6 ] 7
// remove 3, add 6
total_packets += (intervals[j1].num_packets - intervals[j].num_packets);
```



Слика 20: Ентропија одредишних портова пре исправке табеле рутирања

Слика 20 приказује ентропију одредишних портова, на којој се може видети да почетком напада долази до пораста ентропије јер нападачи шаљу нападачке пакете са лажираних изворних адреса, на које после сервер одговара покушавајући да успостави везу. Сервер одговара на SYN пакете (који врше почетак TCP руковања тј. успоставе везе у TCP протоколу) пакетима SYN-ACK након чега очекује још један ACK пакет како би се веза успоставила до краја. Слика 20 не изгледа онако како би се очекивало од DDoS напада јер дуго након напада, сервер остаје заузет одговарањем на почетни напад. Даљим истраживањем, испоставило се да се чворови router-1 и router-2 након напада добацују пакетима без смањивања Time To Live параметра у поруци. Ово се дешава јер је погрешно подешена табела рутирања на router-1 и router-2 чворовима. Изглед ове табеле је приказан испод:

IP route табела за router-1:

```
ftnunsaa@router-1:/tmp$ ip r
10.0.0.0/24 dev eth1  proto kernel  scope link  src 10.0.0.10
10.0.2.0/24 dev eth2  proto kernel  scope link  src 10.0.2.1
192.168.0.0/22 dev eth0  scope link
192.168.0.0/16 via 192.168.1.254 dev eth0
172.16.0.0/12 dev eth0  proto kernel  scope link  src 172.16.6.231
10.0.0.0/8 via 10.0.2.2 dev eth2 realm 1
224.0.0.0/4 dev eth0  scope link
ftnunsaa@router-1:/tmp$
```

Слика 21: IP route табела за router-1

```
ftnunsaa@router-2:/tmp$ ip r
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.1
10.0.2.0/24 dev eth2  proto kernel  scope link  src 10.0.2.2
192.168.0.0/22 dev eth0  scope link
192.168.0.0/16 via 192.168.1.254 dev eth0
172.16.0.0/12 dev eth0  proto kernel  scope link  src 172.16.6.232
10.0.0.0/8 via 10.0.2.1 dev eth2 realm 1
224.0.0.0/4 dev eth0  scope link
ftnunsaa@router-2:/tmp$
```

Слика 22: IP route табела за router-2

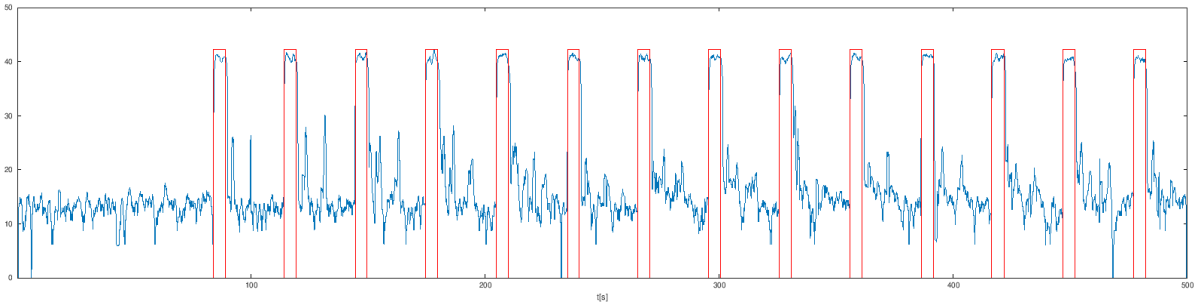
Међутим SYN-ACK пакети које сервер шаље не стижу до лажних клијената које су нападачи фалсификовали, па сервер врши поновно слање тих пакета у одложеним интервалима што се може видети на слици 20 као повремени пад ентропије (сервер чека једно време) па затим делимични пораст ентропије (сервер поново шаље задњи пут SYN-ACK пакете) па све до новог напада где поново расте ентропија и циклус се понавља периодично јер у овом експерименту напад се понавља периодично.

```
ftnunsaa@router-2:/tmp$ sudo ip route del 10.0.0.0/8 via 10.0.2.1 dev eth2 realm 1
ftnunsaa@router-2:/tmp$ sudo ip route add 10.0.0.0/24 via 10.0.2.1 dev eth2 realm 1
```

Овиме је промењена netmask-a IP адресе 10.0.0.0 са /8 на /24 што значи да се ова рута више сад односи само на пакете са IP адресе 10.0.0.xxx уместо 10.xxx.xxx.xxx. Што значи да, рутер 2 више неће прослеђивати пакете који су ван те маске рутеру 1. Такође ово се исто може урадити на router-1 чвору:

```
ftnunsaa@router-1:/tmp$ sudo ip route del 10.0.0.0/8 via 10.0.2.2 dev eth2 realm 1
ftnunsaa@router-1:/tmp$ sudo ip route add 10.0.0.0/24 via 10.0.2.2 dev eth2 realm 1
```

Након ових измена може се видети на слици 23 се види ентропија каква се очекује од исправне мапе рутирања.

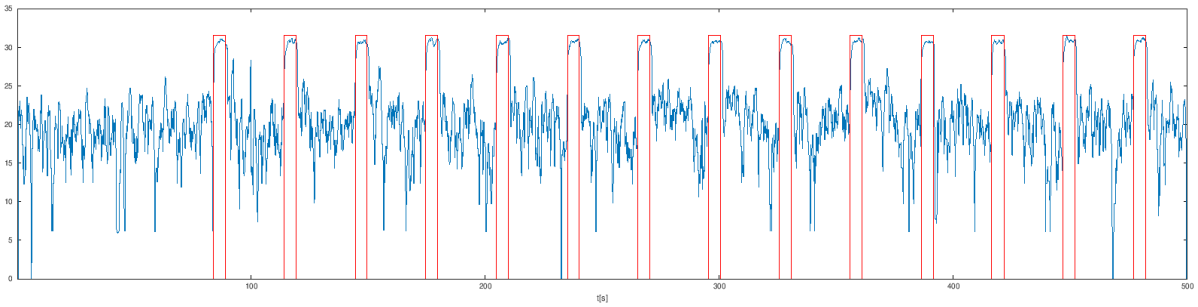


Слика 23: Ентропија одредишних портова после исправке табеле рутирања

Овде се види како скок ентропије одлично упада у интервал напада, што значи да

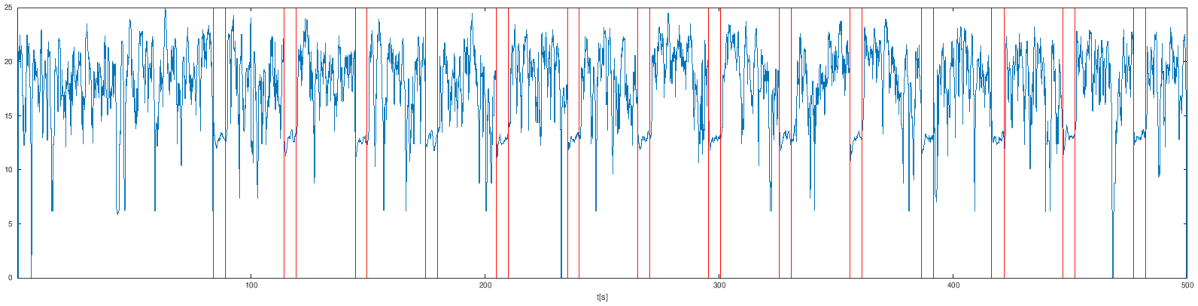
приликом напада пакети пристижу ка много различитијим портовима него легитимни пакети.

Уколико нападачи нападају стално исти порт, овај напад неће бити видљив на дијаграму ентропије одредишних портова али ће можда бити видљив на дијаграму ентропије изворних IP адреса (слика 24), одредишних IP адреса (слика 25) или ентропије токова (слика 26).



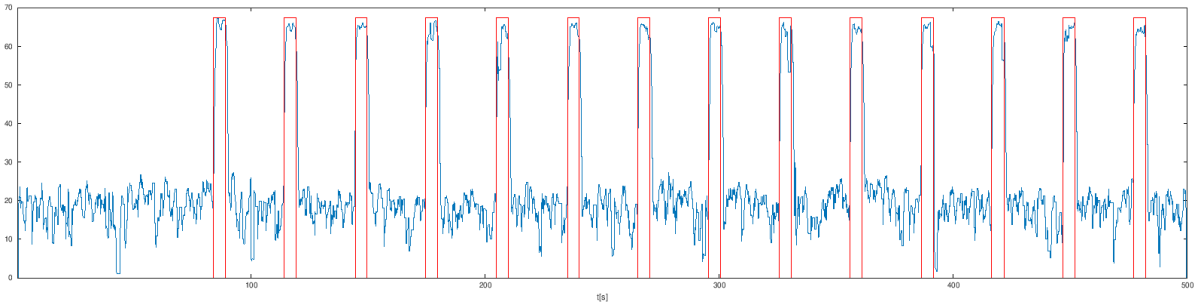
Слика 24: Ентропија изворишних IP адреса

На дијаграму ентропије изворишних адреса је мало теже уочити разлику напада у односу на нормалан саобраћај па би у овом случају теже било искористити ентропију ове особине за детекцију DoS напада, али за друге типове напада, овај тип ентропије би можда више одговарао.



Слика 25: Ентропија одредишних IP адреса

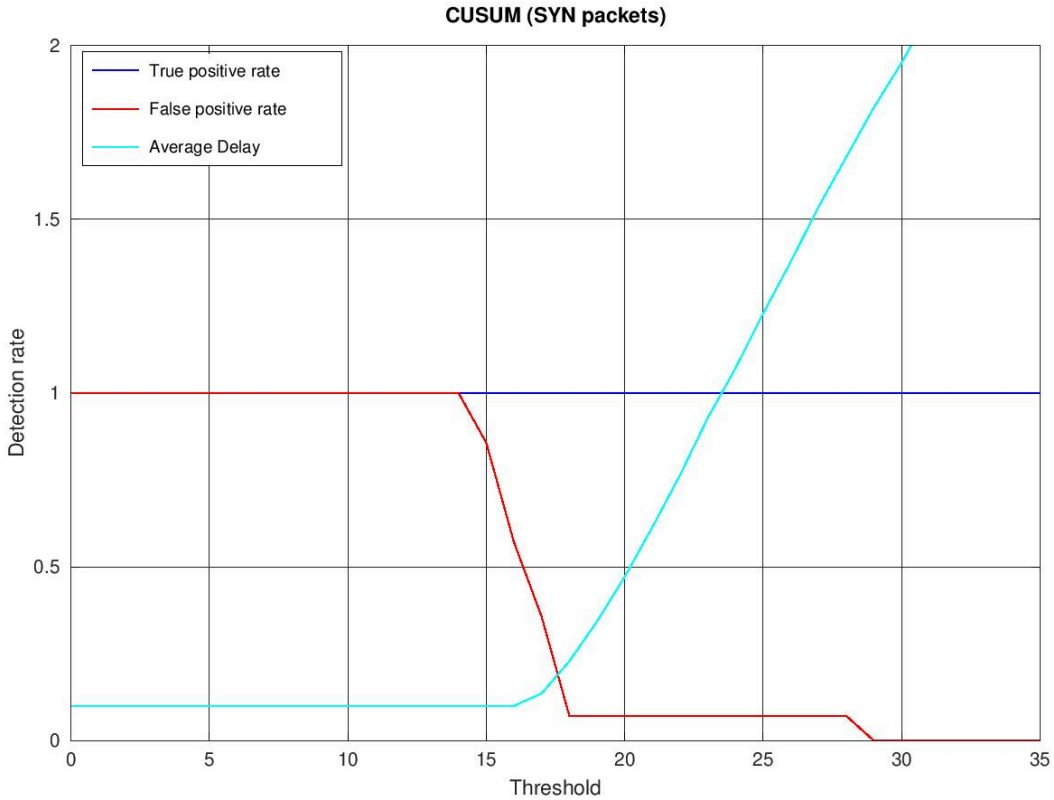
На ентропији одредишних IP адреса уместо повећања ентропије види се смањење ентропије, ово се тумачи чињеницом да клијенти комуницирају са свих 5 сервера у одређеном односу, али приликом напада већина пакета напада једну IP адресу, па долази до тог смањења ентропије.



Слика 26: Ентропија токова

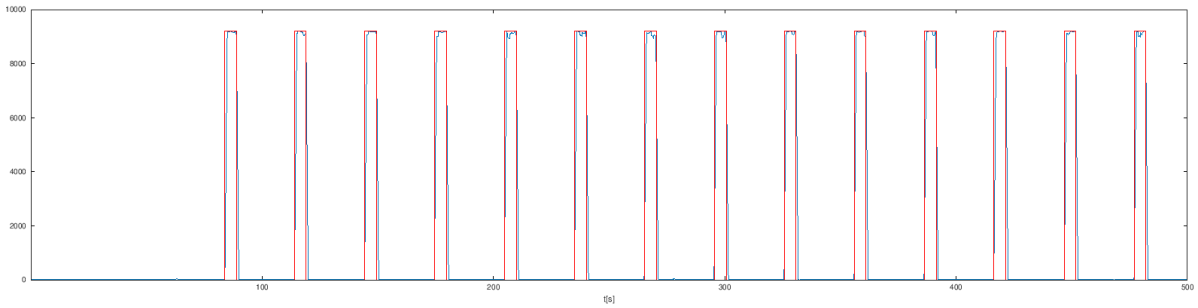
Ентропија токова [5] мери ентропију величине пакета по IP адресама, овај тип ентропије може ухватити широк спектар DDoS напада.

Користећи записана времена напада, осим означивања на овим дијаграмима кад се десио напад, постоји још могућност да се тестира параметар CUSUM алгоритма и пронађе оптимална вредност која омогућава проналажење напада уз што мање кашњење.



Слика 27: Дијаграм успешности детекције DOS напада у зависност од параметра CUSUM алгоритма

На слици 27 је исцртано тамно плавом линијом проценат успешно детектованих напада (вредност 1 представља 100%), и црвеном линијом проценат лажно детектованих напада. Светло плава линија представља кашњење детекције, што се више повећава праг детекције CUSUM алгоритма то је веће кашњење.



Слика 28: Број SYN пакета

На дијаграму SYN пакета се јасно види кад је напад па је овај дијаграм најпогоднији за детекцију овог типа DDoS напада, али ограничење овог приступа је то што се може детектовати само овај тип напада и због ових ограничења се мерење ентропије и примењује, јер има способност да детектује велики спектар DDoS напада једноставно проналажењем скокова ентропије у односу на нормалан саобраћај. CUSUM алгоритам се временом адаптира на стање мреже па ће рецимо ноћу кад је мањи саобраћај DDoS бити подједнако тачно детектован као и дању кад је много већи саобраћај и неће бити лажних детекција које би могле проузроковати проблеме.

5. Испитивање скрипте

Скрипта је тестирана инкрементално приликом самог развијања као и коришћењем у извршењу разних експеримената који су успешно извршени помоћу ове скрипте, али у оквиру овог рада, реализована је емулација SYN Flood DoS напада.

Показало се да скрипта много олакшава експериментисање јер омогућава да поред аутоматског извршавања, експеримент се брзо и лако дефинише и мења.

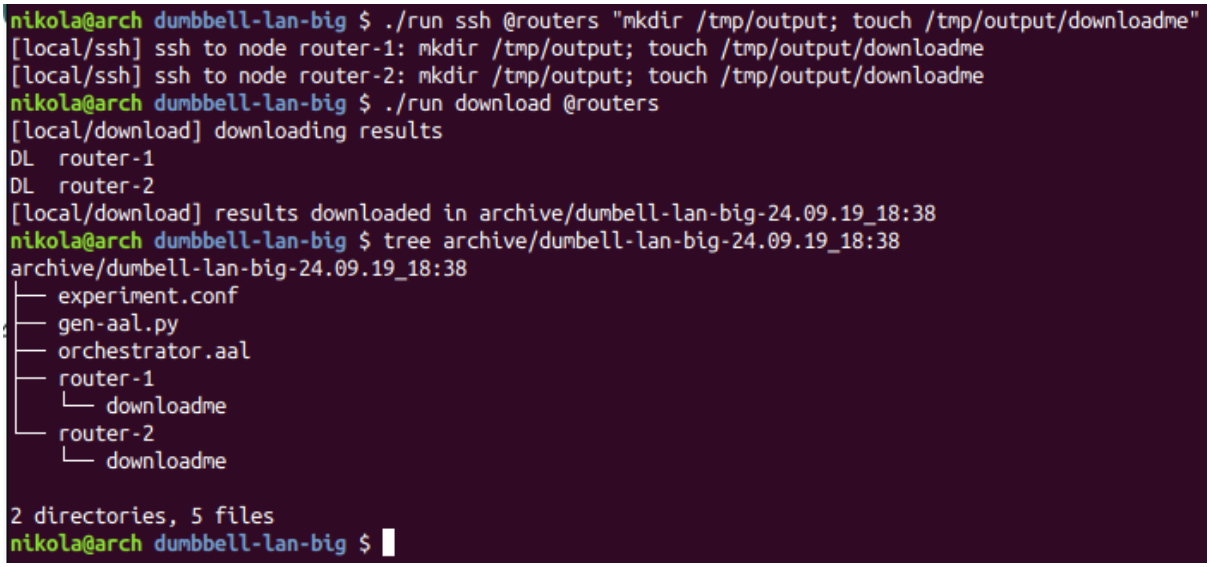
Уз одређене претпоставке и ограничења на коришћење скрипте на начин на који је скрипта предвиђена да ради и за то тестирана, може се сматрати да је скрипта исправна под условом да је коришћена на начин на који је предвиђено.

У супротном, самом чињеницом да је скрипта писана у bash-у, исправност уноса аргумената се не проверава и не постоји адекватан испис грешке који би корисника упутио ка грешки у уносу аргумената.

```
nikola@arch dumbbell-lan-big $ ./run ssh router-2 "touch /tmp/myfile"
[local/ssh] ssh to node router-2: touch /tmp/myfile
nikola@arch dumbbell-lan-big $ ./run ssh @routers "ls /tmp/ | grep myfile" shell
[local/ssh] ssh to node router-1: ls /tmp/ | grep myfile
[local/ssh] ssh to node router-2: ls /tmp/ | grep myfile

[local/ssh] router-1

[local/ssh] router-2
myfile
nikola@arch dumbbell-lan-big $
```

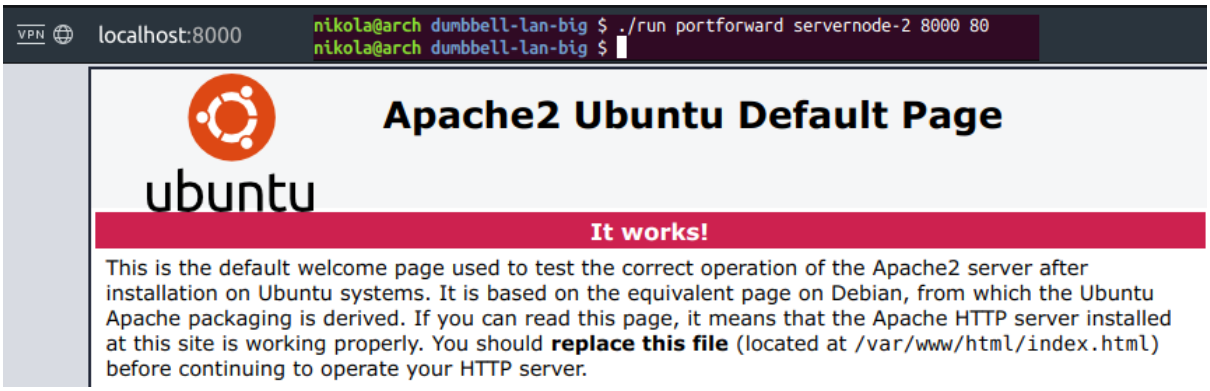
Слика 29: Извршење SSH команди на групи чворова

На слици 29 се види креирање датотеке /tmp/myfile само на router-2 чвору и проверавање на оба рутера да ли се та датотека налази тамо.



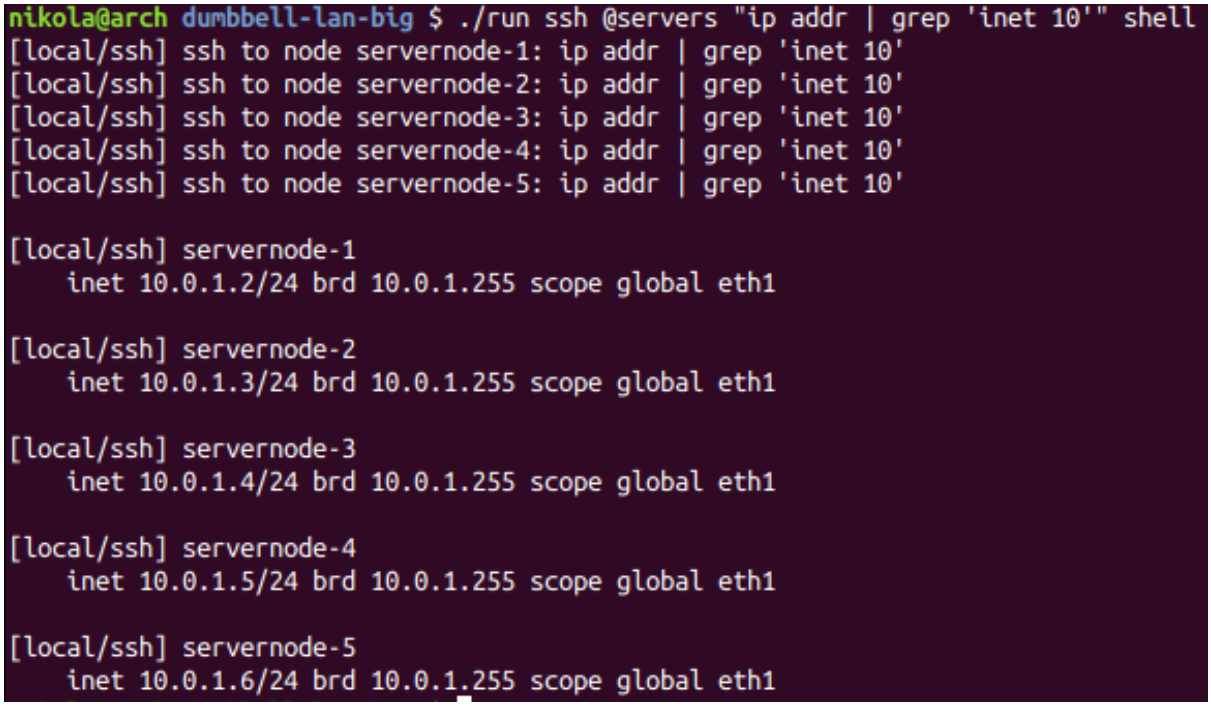
Слика 30: Тестирање преузимања датотека

На слици 30 се види креирање датотеке ”downloadme” која се налази у /tmp/output/ директоријуму, и позивање команде download за скидање резултата који се налазе у /tmp/output/ директоријуму на свим чворовима. Стабло директоријума резултата преузетих је такође приказано на слици.



Слика 31: Тестирање SSH тунела односно port forward-а

На слици 31 је приказан пример прослеђивања порта са локалног порта 8000 ка порту 80 који се налази на чвору servernode-2.



Слика 32: Приказ исписа IP адреса које почињу са 10.x.x.x

На слици 32 приказан је начин на који је лако могуће исписати IP адресе одабраних чворова.

6. Закључак

У овом раду је реализована скрипта која врши аутоматизацију извршења експеримента у DETERLab окружењу. Пречице које су понуђене у оквиру ове скрипте су веома корисне за успостављање велике контроле над окружењем које нуди DETERLab.

Потребу за додатном аутоматизацијом извршења експеримента у односу оне коју нуди оркестратор види се и код других корисника DETERLab-а који такође праве разноразне скрипте које су углавном предвиђене за коришћење у оквиру тог конкретног експеримента. [19]

Унутар ове скрипте је постављен шаблон који би се генерално требао користити за извршење сваког експеримента, на пример за преузимање резултата постоји датотека collect.sh која се извршава на чворовима појединачно и врши чишћење претходних резултата експеримента или паковање нових резултата за преузимање. Све глобалне константе везане за извршење експеримента се налазе у једној датотеци experiment.sh, а подаци, агенти и разне апликације које се користе у експерименту се налазе у data фолдеру. Овај шаблон такође може да служи да олакша дељење кодова експеримената између корисника.

Могуће је веома лако приступити било ком чвору или групи чворова унутар експеримента и покренути одређене команде преко којих се могу отклонити разне грешке и проблеми приликом извршавања експеримента. На овај начин је могуће на брзину експериментисати без коришћења оркестратора ради брзог тестирања исправности апликације која се користи у експерименту, на пример могуће је на брзину покренути неку апликацију на више чворова и исписивати резултате у терминалу са друге стране мреже.

Преузимање резултата је као и извршење команди над групама чворова много убрзано коришћењем паралелног повезивања и истовременог преузимања датотека са више чворова.

Поред аутоматизације одрађен је још један алат који омогућава писање оркестрације експеримента у python програмском језику који значајно олакшава писање оркестрације експеримента, са много већим могућностима које нуди python програмски језик у односу

на саму AAL датотеку која је YAML формата и подржава само могућност референцирања и писање у њему је споро и подложно грешкама приликом лошег поравнања блокова. Писање експеримента у python језику омогућава лако писање сложених експеримената, док писање AAL датотеке захтева унапред планирање на папиру, исцртавање дијаграма тока догађаја који би се после реализовао у AAL датотеку јер је непрегледна и самим тим тешко се врше измене.

7. Литература

[1] DETERLab topology commands https://docs.deterlab.net/core/ns-commands/ [27.09.2019.]

[2] DETERLab nodes documentation https://docs.deterlab.net/core/using-nodes/ [27.09.2019.]

[3] Magi Orchestrator Guide https://docs.deterlab.net/orchestrator/orchestrator-guide/ [27.09.2019.]

[4] Magi Orchestrator https://github.com/deter-project/magi [27.09.2019.]

[5] Ilija Basicevic, Stanislav Ocovaj, Miroslav Popovic: The value of flow size distribution in entropy-based detection of DoS attacks. Security and Communication Networks 9(10): 958-965 (2016)

[6] Ilija Basicevic, Stanislav Ocovaj, Miroslav Popovic: Evaluation of entropy-based detection of outbound denial-of-service attacks in edge networks. Security and Communication Networks 8(5): 837-844 (2015)

[7] Ilija Basicevic, Stanislav Ocovaj, Miroslav Popovic: Use of Tsallis entropy in detection of SYN flood DoS attacks. Security and Communication Networks 8(18): 3634-3640 (2015)

[8] Bojović Petar, Basicevic Ilija, Očovaj Stanislav, Popović Miroslav: A Practical Approach to Detection of DDoS Attacks Using a Hybrid Detection Method. 10.13140/RG.2.2.10763.18721/1 (2017)

[9] Bojović Petar, Basicevic Ilija, Očovaj Stanislav, Popović Miroslav: Application of entropy formulas in detection of denial-of-service attacks. 10.1002/dac.4067 (2019)

[10] pcapng file format https://github.com/pcapng/pcapng [27.09.2019.]

[11] Cumulative Sum Algorithm for Detecting SYN Flooding Attacks, Tongguang Zhang, Department of Computer and Information Engineering, Xinxiang College (2011)

[12] James Gross, Mesut Güneş, Klaus Wehrle: Modeling and Tools for Network Simulation, Springer; 2010 edition (September 23, 2010)

[13] Sunny Behal, Krishan Kumar: Trends in Validation of DDoS Research, Procedia Computer Science (2016)

[14] The Tsallis entropy and the Shannon entropy of a universal probability https://arxiv.org/abs/0805.0154 [27.09.2019.]

[15] DDoS-Scripts https://github.com/vbooter/DDoS-Scripts [27.09.2019.]

[16] Program za analizu entropije: Network Entropy Analysis https://github.com/223323/network-entropy-analysis [27.09.2019.]

[17] DETERLab eksperimenti koji koriste ovu skriptu https://github.com/223323/deterlab-experimentation-script [27.09.2019.]

[18] SSH tunnel tips https://www.revsys.com/writings/quicktips/ssh-tunnel.html [27.09.2019.]

[19] Još jedan primer DETERLab skripte https://github.com/rynge/DETERLabTesting/blob/master/run-test [27.09.2019.]