УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Никола Блажић

Подршка за симулацију напада одбијањем услуге у мрежном симулатору нс-3

ДИПЛОМСКИ РАД
- Основне академске студије -

Нови Сад, 2017.

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Никола Блажић

Подршка за симулацију напада одбијањем услуге у мрежном симулатору нс-3

ДИПЛОМСКИ РАД
- Основне академске студије -

Ментор:
Проф. Др Илија Башичевић

студент:
Никола Блажић
РА170/2013

Нови Сад, 2017.

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**:		**Подршка за симулацију напада одбијањем услуге у мрежном симулатору нс-3**
Језик публикације, **ЈП**:		Српски / латиница
Језик извода, **ЈИ**:		Српски
Земља публиковања, **ЗП**:		Република Србија
Уже географско подручје, **УГП**:		Војводина
Година, **ГО**:		**2017.**
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)		**7/31/0/0/17/3/0**
Научна област, **НО**:		Електротехника и рачунарство
Научна дисциплина, **НД**:		Рачунарска техника
Предметна одредница/Кључне речи, **ПО**:		napadi odbijanjem usluge, SYN Flood napad, ns3 simulator, entropija
УДК		
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, **ВН**:		
Извод, **ИЗ**:		U radu je ugrađena podrška za simulaciju SYN Flood napada odbijanjem usluge u ns3 mrežni simulator. Pri tom se koristi dumbbell topologija. Realizovan je takođe modul za parsiranje PCAP datoteka i proračun entropije parametara mrežnog saobraćaja. Na taj način je omogućena simulacija napada odbijanjem usluge u simulatoru i evaluacija metoda za njihovu detekciju koji su zasnovani na entropiji.
Датум прихватања теме, **ДП**:		
Датум одбране, **ДО**:		
Чланови комисије, **КО**:	Председник:	**Проф. др. Небојша Пјевалица**
	Члан:	**Доц. др. Иван Каштелан** — 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**:	Bachelor Thesis
Author, **AU**:	**Nikola Blažić**
Mentor, **MN**:	**PhD Ilija Bašičević**
Title, **TI**:	**Support for simulation of denial of service attack in ns3 simulator**
Language of text, **LT**:	Serbian
Language of abstract, **LA**:	Serbian
Country of publication, **CP**:	Republic of Serbia
Locality of publication, **LP**:	Vojvodina
Publication year, **PY**:	**2017.**
Publisher, **PB**:	Author's reprint
Publication place, **PP**:	Novi Sad, Dositeja Obradovica sq. 6
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)	**7/31/0/0/17/3/0**
Scientific field, **SF**:	Electrical Engineering
Scientific discipline, **SD**:	Computer Engineering, Engineering of Computer Based Systems
Subject/Key words, **S**/**KW**:	denial of service attacks, SYN Flood attack, ns3 simulator, entropy
UC	
Holding data, **HD**:	The Library of Faculty of Technical Sciences, Novi Sad, Serbia
Note, **N**:	
Abstract, **AB**:	In this thesis work, the ns3 simulator has been extended to support simulation of SYN Flood DoS attacks. In simulation, dumbell topology is used. Also, software for parsing of PCAP files and calculation of entropy of traffic parameters has been developed. The results of this work can be used for simulation of DoS attacks and evaluation of entropy based detection methods.
Accepted by the Scientific Board on, **ASB**:	
Defended on, **DE**:	

Defended Board, **DB**:	President:	**Prof. dr. Nebojsa Pjevalica**	
	Member:	**Doc. dr. Ivan Kastelan**	Menthor's sign
	Member, Mentor:	**Prof. dr. Ilija Bašičević**	

Sadržaj

Spisak slika

Skraćenice:

DoS - Denial Of Service

DDoS - Distributed Denial Of Service

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

ICMP - Internet Control Message Protocol

NS3 - Network Simulator 3

EWMA - Exponentially Weighted Moving Average

IIR - Infinite Impulse Response

CUSUM - Cumulative Sum

1. Uvod

U radu je ugrađena podrška za simulaciju SYN Flood napada odbijanjem usluge u ns3 mrežni simulator. Pri tom se koristi dumbbell topologija. Realizovan je takođe modul za parsiranje PCAP datoteka i proračun entropije parametara mrežnog saobražaja. Na taj način je omogućena simulacija napada odbijanjem usluge u simulatoru i evaluacija metoda za njihovu detekciju koji su zasnovani na entropiji.

1.1 Šta je napad odbijanjem usluge (DOS napad):

Napad odbijanjem usluge je napad koji privremeno ili trajno sprečava usluge nekog servisa na internetu ili u lokalnoj mreži. Ovaj napad se obično izvodi tako što napadač šalje veliki broj paketa na metu koju želi da onesposobi. Broj paketa nadjačava broj paketa koje server može da primi, pa se veliki broj paketa od legitimnih klijenata gubi. Tokom napada ulazni bafer servera se puni do kraja, i ne može da primi više paketa; višak paketa se gubi, a u nekim slučajevima dolazi do pucanja servera i trajnog onesposobljavanja (sve dok se server ne restartuje). Većina ovakvih napada se jako lako izvodi i ne zahteva specijalne veštine napadača, a internet mreža još uvek nije dovoljno zaštićena od ovakvih napada. Zaštita od ovog napada podrazumeva detekciju, klasifikaciju i odstranjivanje napadačkih paketa iz mreže pre nego što stignu do samog servera. Paketi koji su par nivoa hijerarhije pre servera, u mrežnom kanalu čine mali deo u odnosu na legitimne paketa, pa samim tim ne narušavaju kvalitet mreže i moguće ih je odstraniti. Ukoliko paketi nisu odstranjeni pre nego što stignu do servera, paketi konvergiraju u jednu tačku (tj. server) i tako preuzimaju veoma značajan udeo paketa u odnosu na legitimne pakete. Napad odbijanjem usluge se još analogno može opisati naletom velikog broja ljudi u red čekanja na neku službu i nisu tu radi usluge nego samo imaju cilj da smetaju i gube vreme drugim ljudima koji su tu radi usluge.

2. Teorijske osnove

2.1 Osnovni protokoli u komunikaciji

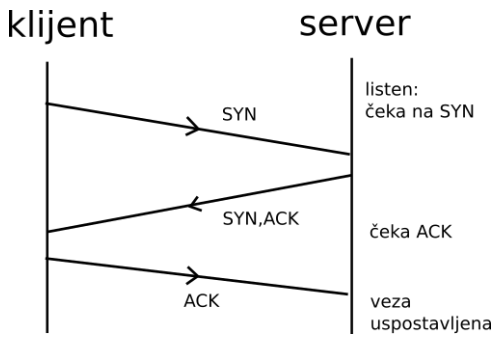
2.1.1 TCP

TCP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0			NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
16	128	Checksum																Urgent pointer (if URG set)															
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

Slika 1: TCP zaglavlje

Ovaj protokol zahteva uspostavljanje kanala komunikacije.



Slika 2: TCP uspostava veze (3-way-handshake)

Uspostavljanje komunikacije se obavlja takozvanim trostrukim rukovanjem (3-way-handshake) Kad klijent želi da ostvari vezu sa serverom, on šalje SYN poruku serveru kao

zahtev za uspostavu veze. Server odgovara sa ACK,SYN paketom kojim potvrđuje ostvarivanje veze, nakon čega klijent odgovara sa ACK porukom i tada je veza uspostavljena.

2.1.2 UDP

Ovaj protokol ne podržava uspostavu veze, podržava grupno slanje (multicast) i slanje svima (broadcast), ali ne koristi ponovno slanje paketa (eng. retransmission), paketi koji su izgubljeni u prenosu se ne šalju ponovo i pošiljalac i ne zna da je paket izgubljen.

UDP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Length																Checksum															

Slika 3: UDP zaglavlje

2.2 Tipovi DOS napada

Postoji više vrsta DoS napada koji na različite načine onesposobljavaju napadnut sistem.

1. Preplavljivanje UDP paketima (eng. UDP Flood)

   Napadač šalje veliki broj UDP paketa na slučajne portove na metu koju napada, kao rezultat napada, tipični računar koji je napadnut će proveriti aplikacije koje slušaju na tom portu, videće da ni jedna aplikacija ne sluša na tom portu i odgovara ICMP porukom "ICMP Destination Unreachable" što je inače predviđeno da obavesti pošiljaoca da servis ne postoji na tom portu. Takva poruka napadaču ide u korist jer dodatno pojačava snagu napada. Takođe napadač još može u paketu koji šalje da izmeni adresu sa koje paket potiče i time omogući da napadnuta meta ne šalje nazad pakete ka napadaču nego ka nekim slučajnim adresama, što omogućava napadaču da ostane anoniman kao i da njegov kanal ne bude opterećen povratnim ICMP paketima koji bi pristizali od mete.

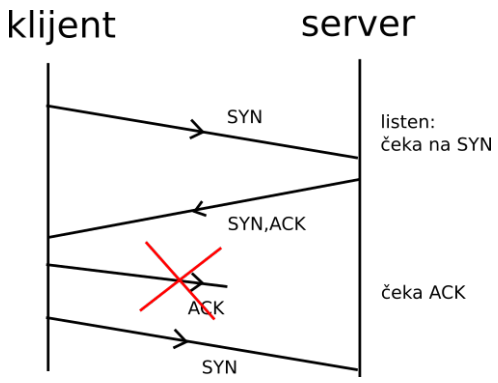
2. Odbijeni napad (eng. Reflected Attack još se zove i Spoofed Attack)

   Ukoliko postoji neki poznati servis koji odgovara sa više paketa nego broj paketa koji je poslat tom servisu, taj servis postaje alat za izvršavanje pojačanog napada na neku drugu metu. Naime napadač izmeni adresu sa koje šalje na adresu mete, što znači da odgovori na pitanja koja su postavljena servisu se neće vraćati napadaču

nego idu pravo na metu koju napadač želi da obori. Zahvaljujući pojačanju napada napadač može sa mnogo manje paketa da izvrši mnogo veći napad.

3. Preplavljivanje SYN paketima (eng. TCP Syn Flood)

   TCP Server u sebi sadrži ograničen broj mogućih veza koje istovremeno mogu biti otvorene na jednom portu na kojem sluša. Ovaj napad funkcioniše tako što napadač započinje uspostavu veze slanjem SYN paketa, ali je ne završava slanjem ACK poruke nakon što server odgovori sa SYN,ACK porukom. Server nakon primanja SYN paketa rezerviše resurse kao što je automat stanja veze i čeka da stigne ACK poruka od klijenta. Nakon pristizanja velikog broja SYN paketa od napadača, server ostaje bez slobodnih resursa za otvaranje nove veze i svi novi legitmni klijenti moraju biti odbijeni jer je dostignut maksimalan broj mogućih veza koje se mogu ostvariti.



Slika 4: Syn flood napad

4. Korišćenjem Peer-to-Peer mreže

   Server koji služi za deljenje IP adresa drugim klijentima kako bi se oni međusobno direktno povezali i direktno komunicirali jedni sa drugim, kao na primer torrent tracker sajt ili serveri sa distribuiranom hash tabelom, napadač može, ukoliko poseduje jedan od tih servera, da preusmeri sve klijente da se pokušaju povezati sa IP adresom od mete koja se napada.

5. Napadi na aplikativnom nivou (Application Level Attacks)

   Ovi napadi su napadi na najvišem nivou, najteži su za detekciju jer često izgledaju kao legitimni paketi dok su ustvari deo DOS napada. Ranjivost na ovaj tip napada zavisi od aplikacije koja se napada. Na primer napad Slowloris koji radi na principu otvaranja mnogo veza sa napadnutim web serverom i održava ih otvorenim što duže. To radi tako što neprekidno šalje polovične HTTP pakete i otvara nove veze.

Slanjem polovičnih ali ispravnih paketa može se provući kroz tradicionalne sisteme za detekciju DOS napada.

6. Narušavanje servisa (eng. Degradation of Service)

   Ovakav tip napada nema za cilj da potpuno prekine rad servisa nego samo da pogorša njegov rad, da prouzrokuje skokove u kašnjenju paketa. Ovaj tip napada je najučinkovitiji protiv servisa koji rade u realnom vremenu. Paketi se šalju kontrolisanom brzinom tako da ne prekidaju servis, ali ga čine neupotrebivim za korisnike koji očekuju odziv servisa u realnom vremenu.

2.2.1 Distribuirani napad odbijanjem usluge (eng. DDoS)

Distribuirani napad odbijanjem usluge (eng. Distributed Denial of Service - DDoS) je najčešći oblik napada gde napadač koristi veliki broj računara da napadne metu. Često napadač ne poseduje veliki broj računara, nego širi virus koji se sakriva u računaru na kojem je pokrenut i periodično od napadača traži komandu koju da izvrši. Računar koji je preuzet, dok ne traje napad se ponaša normalno i ne radi ništa sumnjivo korisniku, ali periodnično posećuje sajt na kojem se nalazi naredba koju treba da izvrši. Takva mreža preuzetih kompjutera koji slušaju naređenje jednog izvršioca se zove **botnet**, a računari u toj mreži se zovu **zombi računari**. Kako bi DDoS napad bio uspešan potrebno je da svi zauzeti kompjuteri započnu napad u isto vreme. Pri napadu, svaki kompjuter uglavnom šalje samo mali broj paketa, masa kompjutera koji vrše napad istovremeno veoma brzo obaraju sistem koji napadaju.

2.3 Wireshark

Wireshark je najviše korišćen program otvorenog koda sa grafičkim interfejsom koji je namenjen za analizu protokola kao i špiuniranje (eng. sniffing) paketa u mreži i ima mogućnost da sačuva pakete u PCAP datoteku što omogućava da se protekli mrežni saobraćaj otvori ponovo u wiresharku ili obradi nekim drugim programom.

2.3.1 PCAP Format

PCap (**p**acket **cap**ture) je binarni format koji je nastao za potrebe programa TCP-Dump, to je program otvorenog koji je napisan 1988. godine i služi za analizu paketa. Program može da sačuva snimljen protok kroz neki mrežni uređaj u pcap datoteku. Danas se često koristi nova verzija formata .pcapng (pcap next generation) [4]. Zahvaljujući otvorenosti formata, PCAP format je postao glavni format koji se koristi za

skladištenje paketa sa mreže u datoteku. Za čitanje ovog formata, koristi se biblioteka libpcap ili winpcap za windows platformu.

Struktura PCAP datoteke:

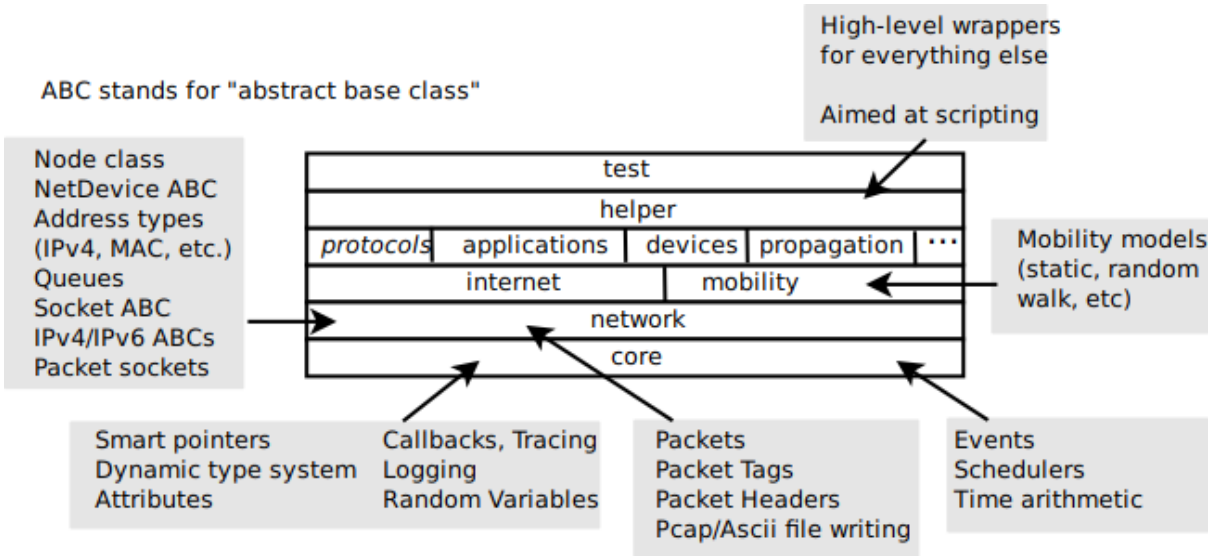
Global Header
Packet Header
Packet Data
Packet Header
Packet Data
...

Svaki paket sadrži ovo zaglavlje, vreme kad je paket ušao u mrežu i dužinu paketa:

```
typedef struct pcaprec_hdr_s {
     guint32 ts_sec;      /* timestamp seconds */
     guint32 ts_usec;     /* timestamp microseconds */
     guint32 incl_len;    /* number of octets of packet saved in file */
     guint32 orig_len;    /* actual length of packet */
} pcaprec_hdr_t;
```

2.4 Program NS-3 - Network Simulator 3

NS-3 je program otvorenog koda za simulaciju diskretnih događaja u mreži, namenjen je za istraživanja i edukaciju. NS-3 simulator je rađen u C++ programskom jeziku, a postoji i sprega sa Python programskim jezikom. Pa je sa tim izbor za postavku simulacije i izvršenje simulacije moguć pomoću jezika C++ ili Python-a. Program podržava generisanje PCAP datoteke koja se posle može obraditi sa drugim programom, kao na primer računanje entropije u ovom slučaju. Program se sadrži iz modula jezgra, mrežnog i internet modula kao i velikog broja ostalih modula koji se ređe koriste.



Slika 5: Organizacija ns3 softvera

2.4.1 Spajanje objekata (eng. Object Aggregation)

Kako se interni kod ne bi trebao menjati prilikom dodavanja novih modula i novih mogućnosti, NS-3 koristi razne šeme za razdvajanje modula kao što je forma dinamičkog nasleđivanja. Svaki objekat u ns-3 može da sadrži skup drugih objekata koji su dodati dinamički pri izvršavanju programa.

```
Ptr<Node> node;
Ptr<TCPSocketFactory> socket_factory = CreateObject<TCPSocketFactory>();
node->AggregateObject(socket_factory); // dodaje objekat u skup objekata
Ptr<TCPSocketFactory> factory1 = node->GetObject<TCPSocketFactory>(); // pronalazi
    objekat TCPSocketFactory
Ptr<SocketFactory> factory2 = node->GetObject<SocketFactory>(); // isto vraca
    objekat TCPSocketFactory
```

Svaki objekat u NS-3 simulatoru se čuva u specijalnom pokazivaču koji broji reference, i kad se broj referenci smanji na 0, objekat se automacki uništava.

```
node -> AggregateObject(socket_factory);
```

dodaje objekat socket_factory u skup ”nasleđenih” objekata.

```
node->GetObject<SocketFactory>();
```

takođe vraća TCPSocketFactory ali ga pretvori u pokazivač na SocketFactory.

2.4.2 Čvorovi (eng. Node)

Glavna jedinica za postavku simulacije je čvor (eng. Node). Svaki čvor koji se napravi postaje globalni u okviru projekta i sadrži se u singleton klasi NodeList. Svakom

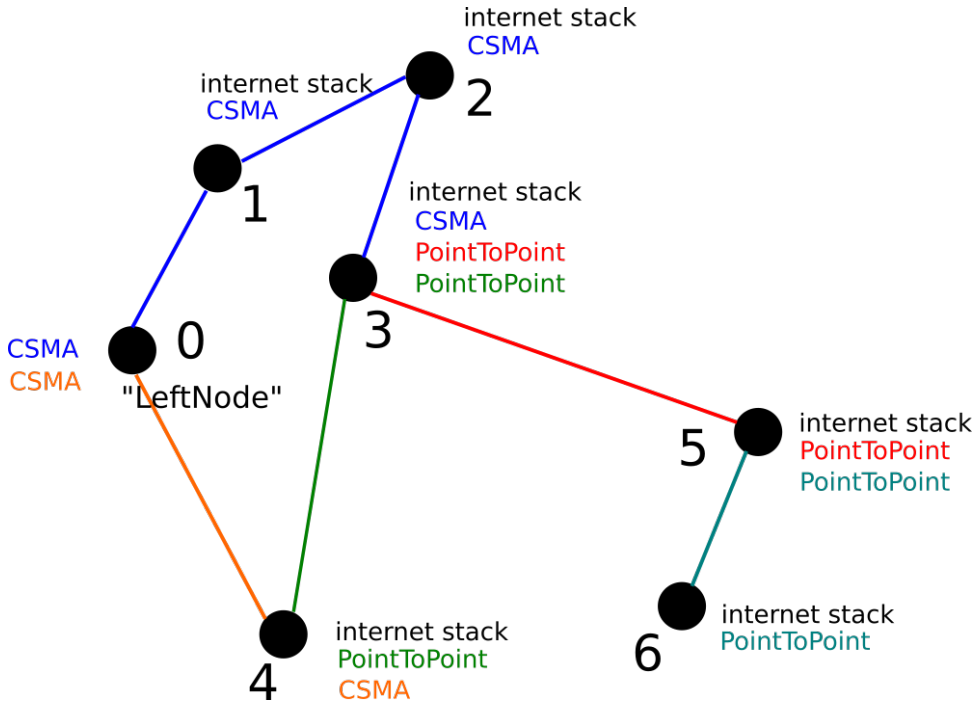
čvoru moguće je zadati neko ime koje se koristi u log fajlovima kao i u nazivu generisane PCAP datoteke.

```
Ptr<Node> node = NodeList::GetNode(0);
Names::Add("LeftNode", node);
```

Inače moguće je bilo kom objektu zadati ime pomoću ovog singletona "Names", a zatim preuzeti objekat po imenu sa

```
Ptr<Object> obj = Names::Find("imeobjekta");
```

Na čvor se instaliraju razni uređaji kao i aplikacije koje se izvršavaju na tom čvoru.



Slika 6: Čvorovi u ns3

Sa slike 6 se vide razni čvorovi sa različitim uređajima i različitim protokolima instaliranim. Svi čvorovi osim čvora 0 (u prethodnom kodu je tom čvoru još i dat naziv "LeftNode") imaju internet stack instaliran pomoću čega mogu svi međusobno da razmenjuju podatke pomoću IP adresa i tabele rutiranja koje se instaliraju pri instalaciji internet stack-a. Sa obzirom da čvor 0 nema instaliran internet stack, jedini način komunikacije je pomoču korišćenja fizičke (MAC) adrese i generičkim socket-om (PacketSocket).

2.4.3 Informacije o klasi (TypeId)

Ns3 koristi menadžer tipova, TypeId koji sadrži sve informacije potrebne za instanciranje klase datog tipa. TypeId je određen jedinstvenim nazivom klase koji tipično počinje sa ns3::, na primer: "ns3::ConstantRandomVariable". Pored toga sadrži i atribute

koji se mogu postaviti pre ili posle instanciranja klase. Postavka atributa pre instanciranja klase se vrši pomoću fabrike objekata. Atributi koji se mogu dodeliti klasi mogu biti definisani bilo kod tipa, npr. StringValue, BooleanValue, ... Za sve tipove u ns3 je definisan njegov odgovarajući atributski tip i njegov naziv se sastoji od imena tipa i nastavak Value, npr. NekiTipValue. Svi tipovi podržavaju deserializaciju iz StringValue u instancu objekta nekog tipa, na primer:

```
onOffTCPHelper.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[
    Constant=1]"));
```

OnOffApplication kojeg onOffTCPHelper pravi definiše atribut koji je pokazivač na bazni objekat RandomVariableStream kojeg sve klase za slučajne promenjive nasleđuju kao i klasa ns3::ConstantRandomVariable. StringValue("ns3::ConstantRandomVariable[Constant=1]") instancira klasu ns3::ConstantRandomVariable i dodeli atributu Constant vrednost 1. Ovo je skraćeno za:

```
Ptr<ConstantRandomVariable> random = CreateObject<ConstantRandomVariable>();
random->SetAttribute("Constant", IntegerValue(1));
```

2.4.4 Fabrika objekata (eng. ObjectFactory)

Fabrika objekata se jako često koristi i služi kao konstruktor za instanciranje novih objekata jer omogućava postavku početnih atributa pre stvaranja objekta sa tim atributima.

```
ObjectFactory factory;
factory.SetTypeId("ns3::TCPSocketFactory");
factory.Set("DataRate", StringValue("100Kbps"));
Ptr<TCPSocketFactory> obj = f.Create();
```

2.4.5 Globalna i podrazumevana konfiguracija

Jedna od korisnih globalnih promenjivih je ChecksumEnabled koja je definisana u socket-u, i omogućava računanje kontrolne sume paketa čime se paketi u wireshark programu prikazuju kao ispravni.

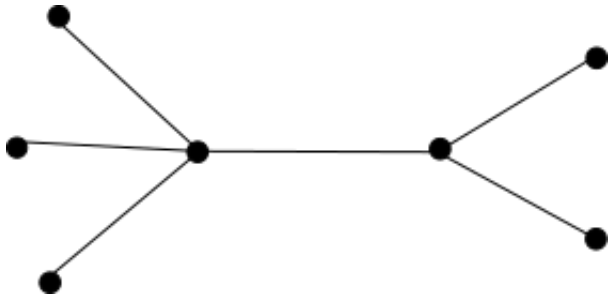
```
GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));
```

Za postavku podrazumevanih vrednosti pri instanciranju neke klase koristi se Config::SetDefault

```
Config::SetDefault("ns3::TCPL4Protocol::SocketType", StringValue("ns3::TCPVegas"));
Config::SetDefault("ns3::TCPSynFloodSocket::RandomizeSourceAddress", BooleanValue(
    true));
```

2.4.6 Primer postavke simulacije

Postavka simulacije će biti prikazana na primeru Dumbbell topologije koja se posle koristi i za testiranje TCPSynFlood agenta.



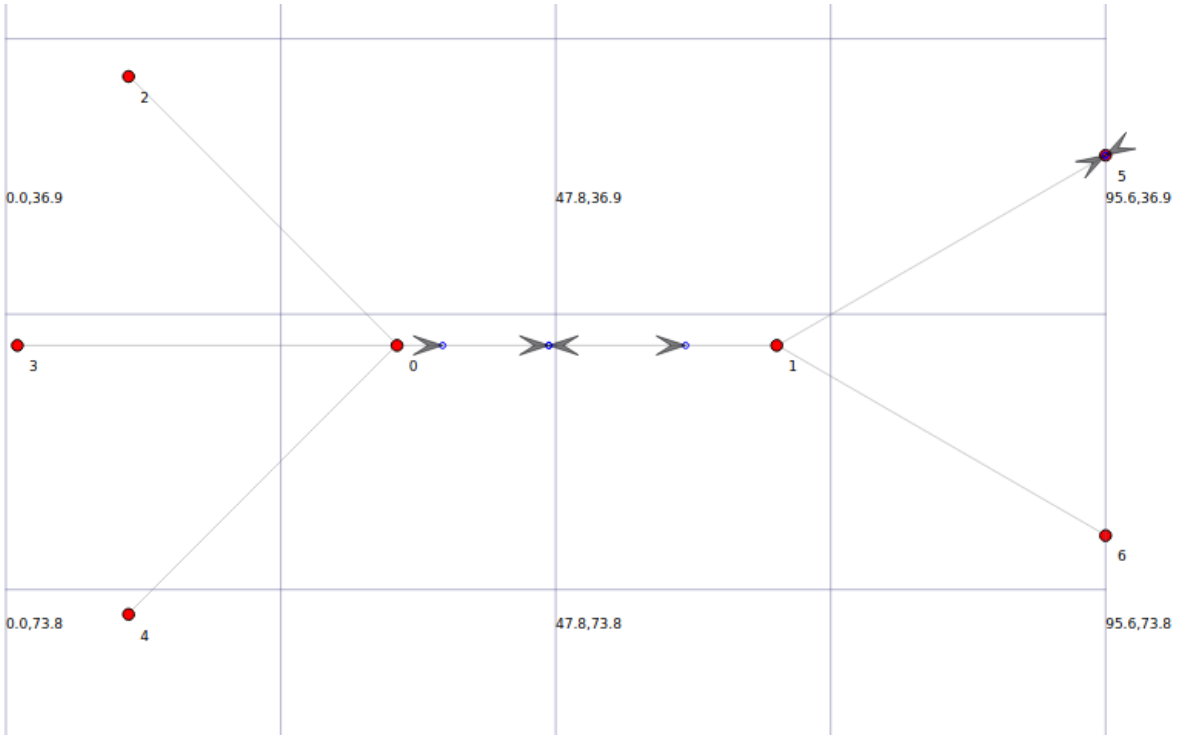
Slika 7: Dumbbell topologija

Svi čvorovi sa leve strane moraju da komuniciraju sa desnom stranom samo preko jednog kanala koji ograničava protok i kašnjenje paketa.

```
int main(int argc, char **argv)
{

  // globalna promenjiva, za računanje kontrolne sume, koja omogućava da wireshark
      prepozna pakete kao ispravne
  GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));
  // podrazumevana promenjiva za klasu ns3::TCPL4Protocol
  Config::SetDefault("ns3::TCPL4Protocol::SocketType", StringValue("ns3::TCPVegas"));

  // prikazivanje svih vremena u milisekundama
  Time::SetResolution (Time::MS);

  PointToPointHelper PTPRouter;
  PointToPointHelper PTPLeaf;

  // atributi za instanciranje uređaja na glavnom kanalu (između levih i desnih
      čvorova)
  PTPRouter.SetDeviceAttribute("DataRate", StringValue("20Mbps"));
  PTPRouter.SetChannelAttribute("Delay", StringValue("25ms"));

  // atributi za instanciranje novih PointToPoint uređaja
  PTPLeaf.SetDeviceAttribute("DataRate", StringValue("100Mbps"));
  PTPLeaf.SetChannelAttribute("Delay", StringValue("1ms"));

  // PointToPointDumbbellHelper vrši instanciranje čvorova (oni se automacki ubacuju
      u globalni singleton NodeList)
  // pravi se 3 čvora sa leve strane, 2 čvora sa desne strane i oni dobijaju atribute
      koji su prethodno definisani za PTPLeaf
  // PTPRouter dobija druge atribute, recimo manju brzinu protoka, veće kašnjenje
  PointToPointDumbbellHelper dumbbell(3, PTPLeaf, 2, PTPLeaf, PTPRouter);

  // instaliranje raznih internet protokola na prethodno instanciranim čvorovima
      dumbbell topologije
  InternetStackHelper stack;
```

```
dumbbell.InstallStack(stack);

// dodavanje IP adrese mreža u topologiji
dumbbell.AssignIpv4Addresses(Ipv4AddressHelper("10.1.1.0", "255.255.255.0"),
                  Ipv4AddressHelper("10.2.1.0", "255.255.255.0"),
                  Ipv4AddressHelper("10.3.1.0", "255.255.255.0"));


Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

// ispis dodeljenih ip adresa
std::cout << "Left leaves: " << std::endl;
std::cout << "\tUpper: " << dumbbell.GetLeftIpv4Address(0) << std::endl;
std::cout << "\tLower: " << dumbbell.GetLeftIpv4Address(1) << std::endl;

std::cout << "Right leaves: " << std::endl;
std::cout << "\tUpper: " << dumbbell.GetRightIpv4Address(0) << std::endl;
std::cout << "\tLower: " << dumbbell.GetRightIpv4Address(1) << std::endl;

// dodeljivanje naziva levim čvorovima
for(uint i=0; i < dumbbell.LeftCount(); i++) {
   Names::Add(std::string("Left(")+std::to_string(i)+")", dumbbell.GetLeft(i));
}

// dodeljivanje naziva desnim čvorovima
for(uint i=0; i < dumbbell.RightCount(); i++) {
   Names::Add(std::string("Right(")+std::to_string(i)+")", dumbbell.GetRight(i));
}

// kontejner za aplikacije preko čega je moguce lakše pristupati svim aplikacijama
    u njima istovremeno
ApplicationContainer clientApps;
ApplicationContainer serverApps;

// TCP Clients
OnOffHelper clientHelperTCP("ns3::TCPSocketFactory", Address());
// OffTime - koliko dugo aplikacija ne šalje pakete
// OnTime - koliko dugo aplikacija šalje pakete, nakon čega ponovo prelazi u stanje
    Off i čeka OffTime
// pre nego što ponovo uđe u stanje On (kad šalje pakete)
clientHelperTCP.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[
    Constant=1]"));
clientHelperTCP.SetAttribute("OffTime", StringValue("ns3::ConstantRandomVariable[
    Constant=0.2]"));
clientHelperTCP.SetAttribute("PacketSize", UintegerValue(512));

// na koju adresu klijenti treba da šalju pakete, u ovom primeru svi levi čvorovi
    šalju prvom desnom čvoru
AddressValue remote(InetSocketAddress(dumbbell.GetRightIpv4Address(0), 50000));
clientHelperTCP.SetAttribute("Remote", remote);

clientApps.Add(clientHelperTCP.Install(dumbbell.GetLeft(0)));
clientApps.Add(clientHelperTCP.Install(dumbbell.GetLeft(1)));

// Pomoćna klasa (Helper) za definisanje PacketSink aplikacije za TCP (koristi
    ns3::TCPSocketFactory, za UDP se koristi ns3::UDPSocketFactory) servere
PacketSinkHelper serverHelper("ns3::TCPSocketFactory", Address());
```

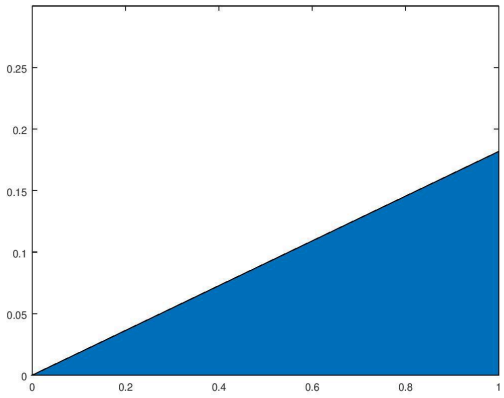
```
    // postavka lokalne adrese i porta na kojoj sluša server
    AddressValue local(InetSocketAddress(dumbbell.GetRightIpv4Address(0), 50000));
    serverHelper.SetAttribute("Local", local);

    // instaliranje serverske aplikacije koja samo prima pakete (i potvrđuje primljeni
        paket)
    serverApps.Add(serverHelper.Install(dumbbell.GetRight(0)));
    serverApps.Add(serverHelper.Install(dumbbell.GetRight(1)));

    // jako je lako uključiti ASCII ispis u fajl
    AsciiTraceHelper ascii;
    PTPRouter.EnableAsciiAll (ascii.CreateFileStream ("dumbbell-tcp.tr"));

    // postavka kad aplikacija počinje da radi i kad prestaje da radi
    serverApps.Start(Seconds(0.0));
    serverApps.Stop(Seconds(50.0));

    clientApps.Start(Seconds(0.0));
    clientApps.Stop(Seconds(50.0));

    // uključivanje generisanja Pcap fajlova sa saobraćajem kroz određeni čvor
    // generisani fajlovi će biti nazvani kao PTPLeaf-Left(0)-0.pcap,
        PTPLeaf-Left(0)-1.pcap, ...,
    // pravilo dodeljivanja naziva ovih fajlova je:
    // <naziv pcap fajla>-<ime čvora ukoliko postoji>(<jedinstveni identitet
        čvora>)-<indeks uređaja na čvoru>.pcap
    PTPRouter.EnablePcapAll("PTPRouter");
    PTPLeaf.EnablePcapAll("PTPLeaf");

    // veoma je lako generisati animaciju koja se može otvoriti sa NetAnim programom
        koji je dat uz ns3 program
    // veličina prozora za animaciju dumbbell topologije
    dumbbell.BoundingBox (1, 1, 100, 100);
    // animacija se generiše u fajlu dumbbell-tcp-animation.xml
    AnimationInterface anim ("dumbbell-tcp-animation.xml");

    // pokretanje simulacije
    Simulator::Run();
    Simulator::Destroy();
    return 0;
}
```



Slika 8: Primer animacije u NetAnim programu iz ns3 programa

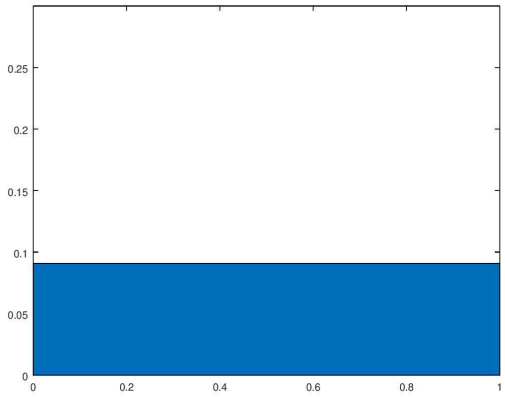
2.5 Entropija

Entropija je mera raspoređenosti dešavanja određenih pojava. Recimo ukoliko se iz skupa mogućih događaja, neki događaji javljaju mnogo češće od drugih, entropija će biti manja.



Slika 9: Primer histograma male entropije

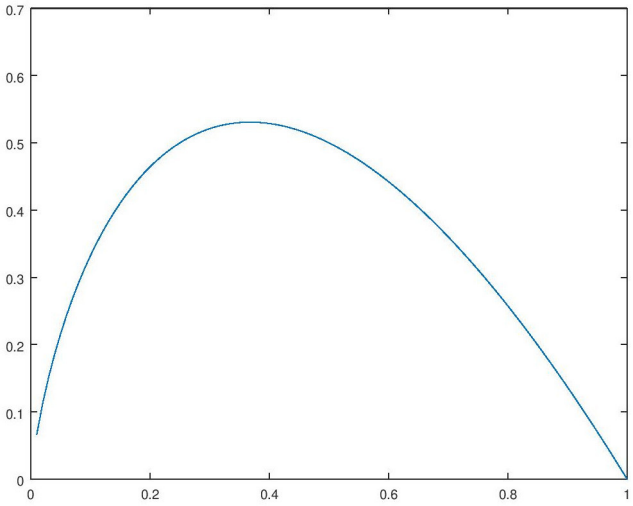
Ukoliko se svi događaji javljaju u istoj količini (tj. raspodela je uniformna), entropija će biti najveća moguća.



Slika 10: Primer histograma uniformne respodele sa velikom entropijom

Formula za računanje Shannon entropije je: $H(P) = -\sum_{i=1}^{N} p_i log(p_i)$

P u ovoj formuli predstavlja skup svih verovatnoća i njihova suma mora biti jednaka jedinici.



Slika 11: Izgled funkcije f(p) = -p*log(p)

Postoji više definicija entropije koje se koriste za različite svrhe:

1. U fizici

   U termodinamici entropija predstavlja meru stanja gasa (količina kompresije) i agregatnih stanja, na primer čvrsto agregatno stanje ima malu entropiju jer je struktura veoma gusto raspoređena pa je čvrstog stanja. Takođe se koristi u srodnim oblastima za statističku mehaniku i kvantnu mehaniku.

2. U teoriji informacija

Često se entropija koristi za ocenu kvaliteta kompresije podataka bez gubitaka. Mala entropija znači da se informacija može dosta kompresovati i preneti sa malom količinom informacija. Velika entropija znači da se informacija ne može kompresovati i mora se cela preneti.

Vidi se da je značaj entropije veliki i postoji mnogo varijanti entropije kod kojih je određena varijanta pogodnija za određene situacije.

Takođe postoji još jedna varijanta entropije koja se zove Tsalis entropija za koju neki autori smatraju da je, zbog nekih karakteristika mrežnog saobraćaja, ova verzija entropije pogodnija i daje bolje rezultate od prethodne, Shannon entropije.

Formula za Tsalis entropiju je:

$$S_q(p) = \frac{1 - \sum_{i=1}^{N} p_i^q}{q - 1}$$

3. Koncept rešenja

3.1 Ideja za implementaciju TcpSynFlood agenta

Na osnovu načina funkcionisanja aplikacije na čvoru, može se videti da svaka aplikacija kao argument uzima neki SocketFactory i adresu.

```
// Pomoćna klasa (helper) za aplikaciju OnOffApplication
OnOffHelper clientHelper("ns3::TcpSocketFactory", Address());
```

Aplikacija koristi SocketFactory da postavi atribute za instanciranje klase TcpSocket. Ideja je da se napravi novi socket factory kao i novi socket koji se ponaša kao UDP tj. socket bez ostvarivanja veze, nego samo šalje SYN paket pri zahtevu za slanje paketa.

3.2 Algoritam za detekciju napada preko entropije

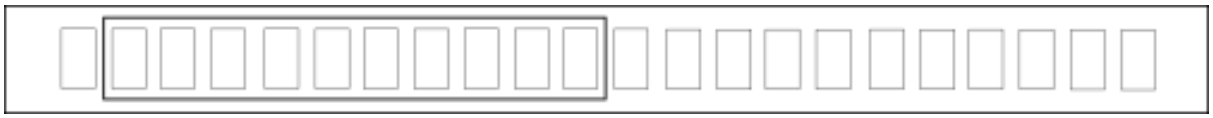
3.2.1 Klasifikacija paketa u intervale

Paketi se čitaju iz PCAP datoteke i svrstavaju se u odbirke. Niz odbiraka koji su odabrani u regularnim intervalima po vremenu koje je zabeleženo na svakom paketu (ts_sec, ts_usec - timestamp). Paketi se podele u grupe i svaki paket se smešta u sebi najbliži vremenski odbirak. Svaki odbirak može da sadrži više paketa, pa se svaki odbirak obrađuje tako što se zabeleži:

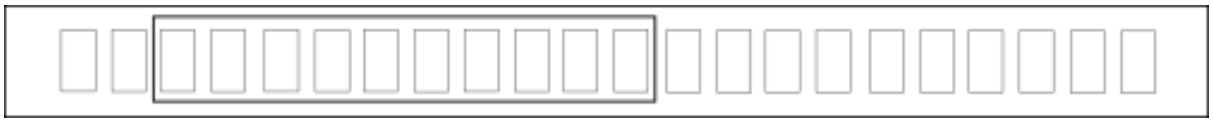
1. broj paketa
2. broj bajtova
3. broj pojavljivanja izvorišnih IP adresa
4. broj pojavljivanja izvorišnih portova
5. broj pojavljivanja odredišnih IP adresa
6. broj pojavljivanja odredišnih portova

3.2.2 Računanje entropije

Razlog za korišćenje detekcije pomoću računanja entropije je u činjenici da entropija u normalnoj mreži varira samo u malom opsegu, i mnogi napadi rezultuju u promeni raspodele adresa, portova i drugih osobina. Kroz ceo niz odbiraka se prolazi klizećim prozorom veličine od 10 odbiraka.



Slika 12: Izgled prozora



Slika 13: Prozor pomeren za 1 interval

Za svaki prozor se računa entropija za: izvorišne IP adrese, odredišne IP adrese, izvorišne portove, i odredišne portove. Tako obrađeni podaci se šalju dalje na obradu sa CUSUM algoritmom koji detektuje promene u entropiji i intervali u kojima se pronađu značajnije promene iznad normalnih granica (koje su adaptivne) se smatraju pod DOS napadom.

3.2.3 Algoritam za detekciju promena (CUSUM)

CUSUM (eng. Cumulative Sum) je algoritam koji pretražuje tačke u kojima se vrednost menja. Ovaj algoritam se koristi u kontroli kvaliteta.

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

μ_n - predstavlja srednju vrednost promenjive y_n

σ_n - predstavlja disperziju tj. standardnu devijaciju promenjive y_n

Kad je $d_n > H$, detektovana je promena vrednosti koja nadmašuje dozvoljene granice i ukoliko posmatramo entropiju paketa, velika je verovatnoća da se radi o DOS napadu. Da bi se promenjiva d_n povećala, potrebno je da vrednost y_n pređe vrednost srednje vrednosti prethodnih y_n vrednosti kao i dodatnu težinu zadatu sa promenjivom K. Dakle ako promenjiva d_n veća od promenjive H to znači da je promena y_n dovoljno dugo

bila povišena da nadmaši skaliranu vrednost disperzije promenjive y_n, drugim rečima y_n je dovoljno rasla da nadmaši njegovu varijaciju pri normalnim uslovima koja postoji kad nema DOS napada, što znači da postoji abnormalno povećanje entropije tj. velika je verovatnoća da se radi o DOS napadu.

Veličina normalnog saobraćaja se menja tokom vremena, ali ta promena je prilično spora za razliku od naleta DOS napada. Za svrhe normalizacije promenjivog saobraćaja koristi se koncept težinske promene tokom vremena, tj. postoji parametar ($\beta 1$ i $\beta 2$) koji određuje brzinu usvajanja nove vrednosti u odnosu na staru. To je ustvari interpolacija ka novoj vrednosti sa parametrom iz opsega vrednosti [0,1], i označava koliki procenat nove vrednosti treba uzeti i koliki procenat stare vrednosti. Što znači da se normalno stanje ne može brzo promeniti. Parametri $\beta 1$ i $\beta 2$ označavaju brzinu promene srednje vrednosti i disperzije. Ovakav koncept sporije promene vrednosti se još naziva i eksponencijalno težinski kretanje srednje vrednosti (eng. Exponentially weighted moving average - EWMA), i predstavlja jednostavan filter sa beskonačnim impulsnim odzvom (eng. Infinite impulse response - IIR) u niskopropusnom režimu. U nekim implementacijama CUSUM algoritma za neke druge svrhe koristi se i promenjiv parametar K, ali je tokom eksperimenata pokazano da varijacija parametra K ne doprinosi puno ka boljim rezultatima algoritma.

4. Programsko rešenje

4.1 Implementacija TCP Syn Flood agenta na NS3 simulatoru

Pošto u samom socketu nema potrebe da se čuva neko specijalno stanje, osim stanja atributa, nije bilo potrebno koristiti većinu apstraktnih funkcija, nego ih samo implementirati tako da vraćaju neke podrazumevane vrednosti. Definisana je klasa TcpSynFloodSocket koja nasleđuje apstraktnu klasu Socket nakon čega je potrebno definisati i implementirati sve virtualne metode koje su definisane kao apstraktne u klasi Socket. Izgled klase TcpSynFloodSocket:

```
class TcpSynFloodSocket : public Socket {
  public:
   static TypeId GetTypeId (void);
   TcpSynFloodSocket (void);

   virtual ~TcpSynFloodSocket (void);
   virtual enum Socket::SocketErrno GetErrno (void) const;
   virtual enum Socket::SocketType GetSocketType (void) const;
   virtual Ptr<Node> GetNode (void) const;
   void SetNode (Ptr<Node> node);
   int SetupEndpoint();
   virtual int Bind (const Address &address);
   virtual int Bind ();
   virtual int Bind6 ();
   virtual int Close (void);
   virtual int ShutdownSend (void);
   virtual int ShutdownRecv (void);
   virtual int Connect (const Address &address);
   virtual int Listen (void);
   virtual uint32_t GetTxAvailable (void) const;
   virtual int Send (Ptr<Packet> p, uint32_t flags);
   virtual int SendTo (Ptr<Packet> p, uint32_t flags, const Address &toAddress);
   virtual uint32_t GetRxAvailable (void) const;
   virtual Ptr<Packet> Recv (uint32_t maxSize, uint32_t flags);
   virtual Ptr<Packet> RecvFrom (uint32_t maxSize, uint32_t flags, Address &
       fromAddress);
   virtual int GetSockName (Address &address) const;
   virtual int GetPeerName (Address &address) const;
```

```
  virtual bool SetAllowBroadcast (bool allowBroadcast);
  virtual bool GetAllowBroadcast () const;

  void SetTcp(Ptr<TcpL4Protocol> tcp);

  private:
    Ptr<RandomVariableStream> m_randomAddress;
    Ptr<RandomVariableStream> m_randomPort;
    SequenceNumber32 m_seq;
    Ipv4EndPoint* m_endPoint;
    Ptr<Node> m_node;
    Ptr<TcpL4Protocol> m_tcp;
    bool m_randomizeSourceAddress;
};
```

Deo koda koji vrši registraciju konstruktora klase, klasu koju nasleđuje i atribute koje definiše je sledeći:

```
NS_LOG_COMPONENT_DEFINE ("TcpSynFloodSocket");

NS_OBJECT_ENSURE_REGISTERED (TcpSynFloodSocket);

TypeId TcpSynFloodSocket::GetTypeId (void)
{
  static TypeId tid = TypeId ("ns3::TcpSynFloodSocket")
    .SetParent<Socket> ()
    .AddConstructor<TcpSynFloodSocket> ()
    .SetGroupName("Test")
    .AddAttribute("RandomizeSourceAddress", "RandomizeSourceAddress",
      BooleanValue(false),
      MakeBooleanAccessor(&TcpSynFloodSocket::m_randomizeSourceAddress),
      MakeBooleanChecker())
   ;

  return tid;
}
```

Makro NS_OBJECT_ENSURE_REGISTERED vrši registraciju klase u dinamički sistem za tipiziranje. Makro koristi definisanje i odma deklaraciju strukture sa konstruktorom da pozove kod pri inicijalnom startovanju programa, pre pozivanja početne main funkcije.

```
#define NS_OBJECT_ENSURE_REGISTERED(type)       \
  static struct Object ## type ## RegistrationClass \
  {                                             \
    Object ## type ## RegistrationClass () {    \
      ns3::TypeId tid = type::GetTypeId ();     \
      tid.SetSize (sizeof (type));              \
      tid.GetParent ();                         \
    }                                           \
  } Object ## type ## RegistrationVariable
```

Rezultat ovog makroa bi bila deklarisana promenjiva kao static (vidljiva samo u jednom modulu u kome je deklarisana) koja se inače nigde ne koristi, ali se samo pozove

konstruktor.

Većina metoda nisu korišćene u ovoj implementaciji, ali su implementirane tako da se svaki njihov poziv može pratiti. To je postignuto sa NS_LOG_FUNCTION makroom koji je dat u okviru veoma fleksibilnog NS3 logging sistema. Sve metode koje nisu korišćene vraćaju neke podrazumevane fiksne vrednosti:

```
Ptr<Node> TcpSynFloodSocket::GetNode (void) const {
   NS_LOG_FUNCTION (this);
   return m_node;
}

int TcpSynFloodSocket::GetSockName (Address &address) const {
   NS_LOG_FUNCTION (this);
   return 0;
}

int TcpSynFloodSocket::GetPeerName (Address &address) const {
   NS_LOG_FUNCTION (this);
   return 0;
}

bool TcpSynFloodSocket::SetAllowBroadcast (bool allowBroadcast) {
   NS_LOG_FUNCTION (this);
   return false;
}
bool TcpSynFloodSocket::GetAllowBroadcast () const {
   NS_LOG_FUNCTION (this);
   return false;
}
```

Neke korišćene funkcije:

```
void TcpSynFloodSocket::SetTcp(Ptr<TcpL4Protocol> tcp) {
   m_tcp = tcp;
}
void TcpSynFloodSocket::SetNode (Ptr<Node> node) {
   NS_LOG_FUNCTION (this);
   m_node = node;
}
int TcpSynFloodSocket::Bind () {
   NS_LOG_FUNCTION (this);
   m_endPoint = m_tcp->Allocate();
   return 0;
}
```

Pri pravljenju socket-a pozivaju se SetTcp i SetNode funkcije. SetTcp zapisuje instancu klase TcpL4Protocol koja se kasnije koristi za slanje TCP paketa. Klasa TcpL4Protocol iz internet modula i služi za slanje i primanje TCP paketa. TcpL4Protocol sadrži u sebi niz otvorenih socket konekcija, svaki od njih ima dodeljen port, na osnovu čega prosleđuje primljene pakete na određeni socket. Kod koji vrši uspostavu ”veze”:

```
int TcpSynFloodSocket::Connect (const Address &address) {
```

```
  NS_LOG_FUNCTION (this);
  InetSocketAddress transport = InetSocketAddress::ConvertFrom (address);
   Ipv4Address ipv4 = transport.GetIpv4 ();
  m_endPoint->SetPeer(ipv4, transport.GetPort());
  SetupEndpoint();
  NotifyConnectionSucceeded();
  return 0;
}
```

Aplikacija koja koristi TCP socket uglavnom ne koristi SendTo funkciju, nego prvo poziva Connect sa čime vrši konekciju, pa onda poziva Send funkciju za slanje paketa na adresu datu pri pozivu Connect funkcije. Sa obzirom da ova implementacija ne vrši uspostavu veze, Connect funkcija samo beleži adresu na koju Send funkcija posle treba da šalje SYN paket. Kod koji vrši slanje SYN paketa:

```
int TcpSynFloodSocket::Send (Ptr<Packet> p, uint32_t flags) {

  TcpHeader tcphdr;
  tcphdr.SetSourcePort(m_endPoint->GetLocalPort());
  tcphdr.SetDestinationPort(m_endPoint->GetPeerPort());
  tcphdr.SetFlags(TcpHeader::SYN);

  tcphdr.SetSequenceNumber(++m_seq);
  tcphdr.SetSourcePort(m_randomPort->GetInteger());

  Ipv4Address s_addr;
  if(m_randomizeSourceAddress) {
    s_addr = Ipv4Address(m_randomAddress->GetInteger());
  } else {

    s_addr = m_endPoint->GetLocalAddress();
  }

  m_tcp->SendPacket(p, tcphdr, s_addr, m_endPoint->GetPeerAddress(), m_boundnetdevice
      );
  int size = p->GetSize() + tcphdr.GetLength();

  NotifyDataSent(size);
  NotifySend(1500);
  return size;
}
```

4.1.1 Postavka simulacije sa TCP flood socket-om

U odnosu na postavku simulacije koja je detaljno objašnjena u teorijskom delu, potrebno je samo instalirati neku aplikaciju sa ns3::TcpSynFloodFactory fabrikom socket-a. U ovom slučaju to je aplikacija OnOffApplication koja se olakšano pravi sa OnOffHelper pomoćnom klasom. Pre instalacije je potrebno instalirati ns3::TcpSynFloodFactory na čvorove na kojima će se ns3::TcpSynFloodFactory koristiti, a to se postiže sa TcpSyn-FloodHelper pomoćnom klasom.

```
// generisanje slučajne izvorne adrese (opcioni atribut)
Config::SetDefault("ns3::TcpSynFloodSocket::RandomizeSourceAddress", BooleanValue(true
    ));
TcpSynFloodHelper synflood;
synflood.Install(dumbbell.GetLeft(0));
synflood.Install(dumbbell.GetLeft(1));

// OnOffHelper pomoćna klasa koja vrši instalaciju OnOffApplication aplikacije sa
    ns3::TcpSynFloodFactory protokolom
OnOffHelper clientHelperAttack("ns3::TcpSynFloodFactory", Address());
// instalacija SYN flood napadačke aplikacije na čvor
clientApps.Add(clientHelperAttack.Install(dumbbell.GetLeft(0)));
```

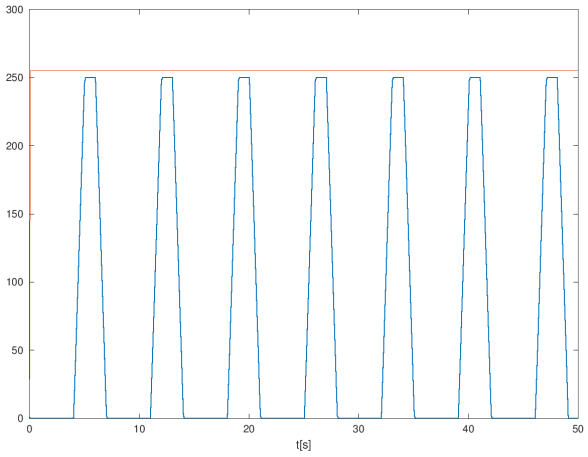
4.1.2 Testiranje simulacije

Rezultati simulacije se mogu prikazati u wireshark-u ili u dijagramu koji pokazuje broj poslatih SYN paketa.

4.769000	10.2.1.1	10.1.2.1	TCP	50000 → 49153 [ACK] Seq=1 Ack=253953 Win=131072 Len=0
4.776000	10.1.2.1	10.2.1.1	TCP	49153 → 50000 [ACK] Seq=253953 Ack=1 Win=131072 Len=5
4.785000	10.1.2.1	10.2.1.1	TCP	49153 → 50000 [ACK] Seq=254465 Ack=1 Win=131072 Len=5
4.785000	10.2.1.1	10.1.2.1	TCP	50000 → 49153 [ACK] Seq=1 Ack=254977 Win=131072 Len=0
4.792000	10.1.2.1	10.2.1.1	TCP	49153 → 50000 [ACK] Seq=254977 Ack=1 Win=131072 Len=5
4.992000	10.2.1.1	10.1.2.1	TCP	50000 → 49153 [ACK] Seq=1 Ack=255489 Win=131072 Len=0
5.001000	10.1.2.1	10.2.1.1	TCP	49153 → 50000 [ACK] Seq=255489 Ack=1 Win=131072 Len=5
5.004000	2.43.220.197	10.2.1.1	TCP	29800 → 50000 [SYN] Seq=0 Win=65535 Len=50
5.004000	10.2.1.1	2.43.220.197	TCP	50000 → 29800 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
5.008000	20.80.53.175	10.2.1.1	TCP	44895 → 50000 [SYN] Seq=0 Win=65535 Len=50
5.008000	10.2.1.1	20.80.53.175	TCP	50000 → 44895 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
5.009000	10.1.2.1	10.2.1.1	TCP	49153 → 50000 [ACK] Seq=256001 Ack=1 Win=131072 Len=5
5.009000	10.2.1.1	10.1.2.1	TCP	50000 → 49153 [ACK] Seq=1 Ack=256513 Win=131072 Len=0
5.012000	82.234.80.48	10.2.1.1	TCP	28080 → 50000 [SYN] Seq=0 Win=65535 Len=50
5.012000	10.2.1.1	82.234.80.48	TCP	50000 → 28080 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
5.016000	82.103.127.232	10.2.1.1	TCP	22493 → 50000 [SYN] Seq=0 Win=65535 Len=50
5.016000	10.2.1.1	82.103.127.23	TCP	50000 → 22493 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0

Slika 14: Primer u wiresharku

U wireshark-u se vidi da nakon 5. sekunde napad počinje sa namerno proizvoljno postavljenih IP adresa. Takođe se vidi da server šalje SYN,ACK odgovor na tu IP adresu.



Slika 15: Broj syn paketa

4.2 Implementacija algoritma za detekciju napada preko promene entropije

Za ceo algoritam je definisana struktura Interval, koja se koristi za držanje informacija o paketima koji pripadaju određenom vremenskom intervalu. U istoj strukturi su takođe definisani podaci koji će se odnositi na određeni prozor koji traje 1 sekundu i obuhvata sve intervale u toj sekundi počev od ovog intervala pa na dalje.

```
struct Interval {
   int num_packets;  // broj paketa u intervalu
   int num_syn;    // broj tcp syn paketa u intervalu
   int num_bytes;    // ukupan broj bajtova svih paketa u intervalu

   // za broj pojavljivanja određenih portova i ip adresa
   std::vector<int> num_src_ports;
   std::vector<int> num_dst_ports;
   std::vector<int> num_src_ips;
   std::vector<int> num_dst_ips;

   // podaci koji se računaju pri prozorskoj obradi entropije i odnose se na prozor
       koji počinje ovim intervalom
   double ent_pktnum;
   double ent_bytenum;
   double ent_srcport;
   double ent_dstport;
   double ent_srcip;
   double ent_dstip;

   int tot_pktnum;
   int tot_syn;
};
```

Definisan je vektor intervala. Svaki interval predstavlja pakete u vremenskom rastojanju od 100 milisekundi (ukoliko se izabere 10 podintervala tj. 10 intervala u 1 sekundi).

```
    std::vector<Interval> intervals;
```

Kod za inicijalizaciju vektora intervala:

```
int num_intervals; // računa se pri inicijalizaciji
int num_subintervals = 10;
int max_time = 800; // 800 sekundi, sve nakon toga se ignoriše
int num_ports = 1000; // broj različitih portova koji se mogu zapisati (portovi se
    odsecaju po modulu 1000)
void init_vectors() {
   num_intervals = max_time*num_subintervals;
   intervals.resize(num_intervals+1);
   for(auto &s : intervals) {
      s.num_src_ports.resize(num_ports);
      s.num_dst_ports.resize(num_ports);
      s.num_src_ips.resize(num_ports);
      s.num_dst_ips.resize(num_ports);
   }
}
```

Kod za čitanje pcap datoteke i početno svrstavanje paketa u intervale:

```
int parse_pcap(std::string filename) {
   double time;
   int i, j, sec, sub_int;
   int pkt_size;
   int src_addr=0, src_port=0;
   int dst_addr=0, dst_port=0;
   char errbuff[80];
   struct pcap_pkthdr pcap_hdr;

   pcap_t* p = pcap_open_offline(filename.c_str(), errbuff);
   if(!p) { return -1; }

   const u_char *pkt_data;
   bool sec_first = true;
   double sec_offs = 0;
   int last_sec = 0;
   while (pkt_data = pcap_next(p, &pcap_hdr)) {
      tcp_packet* pkt = (tcp_packet*)pkt_data;
      ptp_tcp_packet* ptp_pkt = (ptp_tcp_packet*)pkt_data;
      pkt_size = pcap_hdr.len;
      tcp_header* tcp = &pkt->tcp;
      ip_header* ip = &pkt->ip;

      // ukoliko je paket enkapsuliran u PointToPoint protokolu (kao npr. iz ns3
          simulatora)
      if(ptp_pkt->ptp.type == PROTO_PTP) {
         tcp = &ptp_pkt->tcp;
         ip = &ptp_pkt->ip;
      }

      if(sec_first) {
         sec_first = false;
         sec_offs = pcap_hdr.ts.tv_sec + pcap_hdr.ts.tv_usec * 1e-6 - 0.01;
      }

      // informacije o vremenu slanja ili pristizanja paketa (iz timestamp-a)
      time = (double)(pcap_hdr.ts.tv_sec + pcap_hdr.ts.tv_usec * 1e-6) - sec_offs;
      sec = (int)time;
      sub_int = (int)(fmod(time, 1.0)*num_subintervals);

      // obrada se vrsi najvise max_time sekundi
      if(sec >= max_time) {
         return 0;
      }

      const int i = sec*num_subintervals+sub_int;
      auto& interval = intervals[i];

      // dodavanje broj syn paketa
      if(ip->proto == PROTO_L4_TCP && tcp->flags == tcp_flags::SYN) {
         interval.num_syn++;
      }

      // n_bytes, n_packets, n_pkts[sport], n_pkts[dport]
      interval.num_bytes += pkt_size;
      interval.num_packets++;
```

```
    // odsecanje ip adrese i portovi po modulu broja portova
    src_port = ntohs(tcp->sport) % num_ports;
    dst_port = ntohs(tcp->dport) % num_ports;
    src_addr = ip->saddr.ip % num_ports;
    dst_addr = ip->daddr.ip % num_ports;

    if(use_byte_entropy) {
       interval.num_src_ports[src_port] += pkt_size;
       interval.num_dst_ports[dst_port] += pkt_size;
       interval.num_src_ips[src_addr] += pkt_size;
       interval.num_dst_ips[dst_addr] += pkt_size;
    } else {
       interval.num_src_ports[src_port]++;
       interval.num_dst_ports[dst_port]++;
       interval.num_src_ips[src_addr]++;
       interval.num_dst_ips[dst_addr]++;
    }
  }
  return 0;
}
```

Kod za računanje entropije:

```
void process_entropy() {
  int i, j;

  int total_packets = 0;
  int total_bytes = 0;
  int total_syn = 0;

  // računanje ukupnog broja paketa, syn paketa i bajtova za prvi prozor
  // ovo je neophodno za računanje entropije
  for(int i=0; i < num_subintervals; i++) {
    total_packets += intervals[i].num_packets;
    total_syn += intervals[i].num_syn;
    total_bytes += intervals[i].num_bytes;
  }

  t_total_bytes = total_bytes;

  for (j=0; j < num_intervals - num_subintervals; j++) {
    const int j1 = j + num_subintervals;

    std::unique_ptr<Entropy> pktnum_entropy (new Entropy());
    std::unique_ptr<Entropy> bytenum_entropy (new Entropy());
    std::unique_ptr<Entropy> srcport_entropy(new Entropy());
    std::unique_ptr<Entropy> dstport_entropy(new Entropy());
    std::unique_ptr<Entropy> srcip_entropy(new Entropy());
    std::unique_ptr<Entropy> dstip_entropy(new Entropy());

    if(use_tsalis) {
       srcport_entropy = std::make_unique<TsalisEntropy>(Q);
       dstport_entropy = std::make_unique<TsalisEntropy>(Q);
       srcip_entropy = std::make_unique<TsalisEntropy>(Q);
       dstip_entropy = std::make_unique<TsalisEntropy>(Q);
    }
```

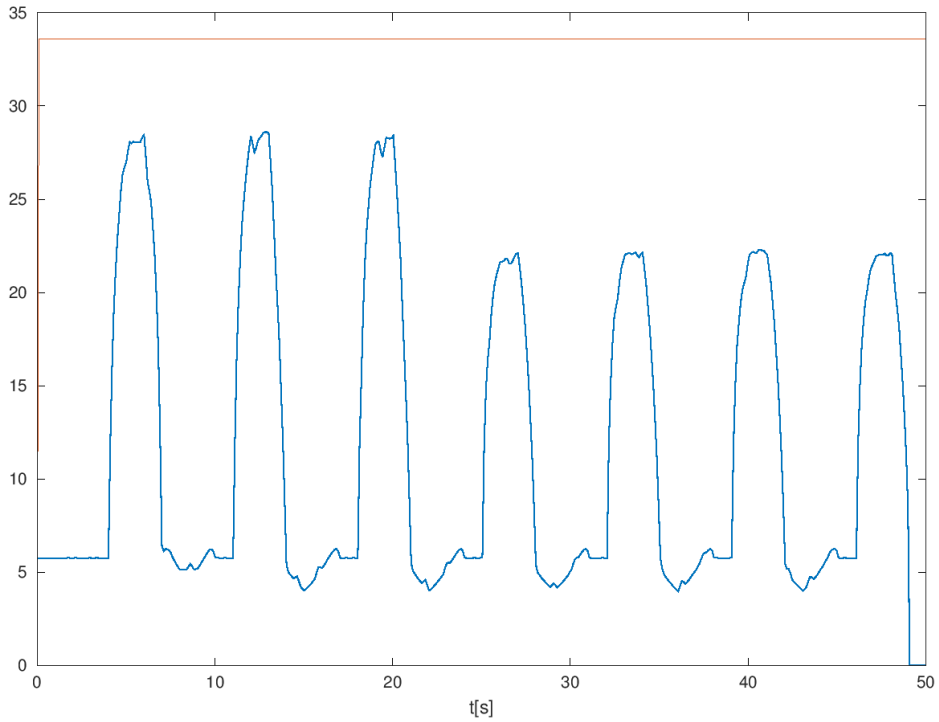
```
    for (i=0; i < num_subintervals; i++) {
       if (intervals[j+i].num_packets != 0) {
          pktnum_entropy->Add( intervals[j+i].num_packets / (double)total_packets );
          bytenum_entropy->Add( intervals[j+i].num_bytes / (double)total_bytes );
       }
    }

    intervals[j].ent_pktnum = pktnum_entropy->GetValue();
    intervals[j].ent_bytenum = bytenum_entropy->GetValue();
    intervals[j].tot_pktnum = total_packets;
    intervals[j].tot_syn = total_syn;

    for (i=0; i < num_ports; i++) {
       int k, srcp=0, dstp=0;
       int srcip=0, dstip=0;
       for (k=0; k < num_subintervals; k++) {
          auto &interval = intervals[j+k];
          srcp += interval.num_src_ports[i];
          dstp += interval.num_dst_ports[i];
          srcip += interval.num_src_ips[i];
          dstip += interval.num_dst_ips[i];
       }

       double total_bytes = use_byte_entropy ? total_bytes : total_packets;
       if (srcp != 0) srcport_entropy->Add(srcp / total_bytes);
       if(srcip != 0) srcip_entropy->Add(srcip / total_bytes);
       if (dstp != 0) dstport_entropy->Add(dstp / total_bytes);
       if(dstip != 0) dstip_entropy->Add(dstip / total_bytes);
    }

    srcport_entropy->SetCount(UINT16_MAX);
    dstport_entropy->SetCount(UINT16_MAX);
    srcip_entropy->SetCount(UINT16_MAX);
    dstip_entropy->SetCount(UINT16_MAX);

    intervals[j].ent_srcport = srcport_entropy->GetValue();
    intervals[j].ent_dstport = dstport_entropy->GetValue();
    intervals[j].ent_srcip = srcip_entropy->GetValue();
    intervals[j].ent_dstip = dstip_entropy->GetValue();

    // za pomeranje prozora potrebno je samo zameniti vrednosti 2 intervala
    // na primer:
    // 1 2 [ 3 4 5 ] 6 7
    // 1 2 3 [ 4 5 6 ] 7
    // izbaciti interval 3 podatke iz prozora, dodati podatke intervala 6
    total_packets += (intervals[j1].num_packets - intervals[j].num_packets);
    total_syn += (intervals[j1].num_syn - intervals[j].num_syn);
    total_bytes += (intervals[j1].num_bytes - intervals[j].num_bytes);

    t_total_bytes += intervals[j1].num_bytes;
  }
}
```

Klasa Entropy korišćena u prethodnom kodu:

```
class Entropy {
  public:
    Entropy();
```

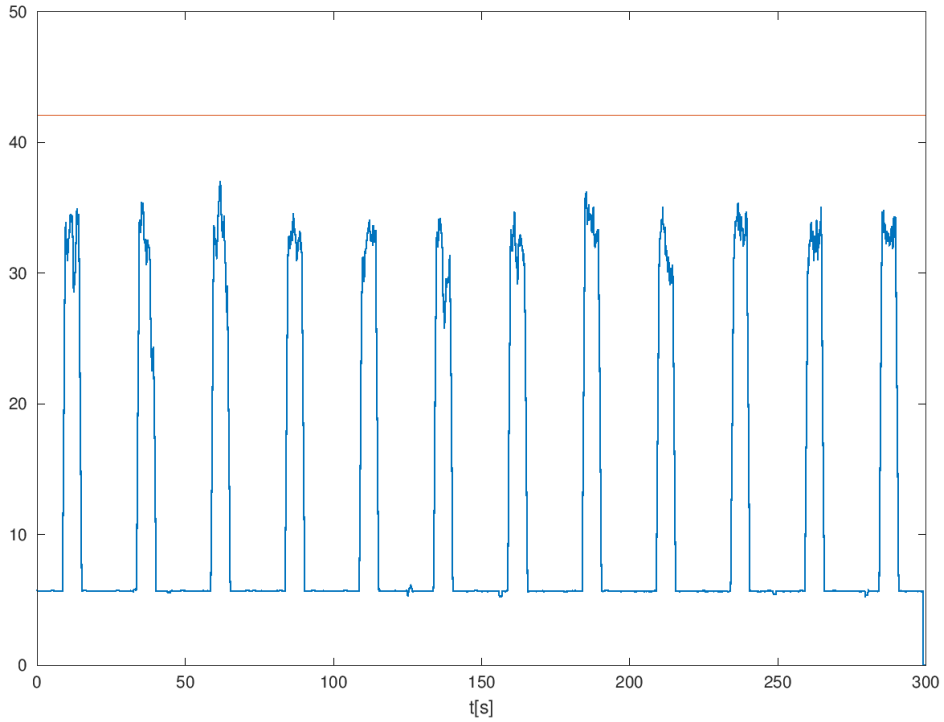
```
      virtual void Add(double p);
      void SetCount(int count);
      virtual double GetValue(bool normalized=true);
   protected:
      double m_value;
      int m_count;
};

Entropy::Entropy() : m_value(0.0), m_count(0) {}

void Entropy::Add(double p) {
   if(p == 0) {
      m_value = 0;
   } else {
      m_value += p * (-log2(p));
   }
   m_count++;
}

void Entropy::SetCount(int count) {
   m_count = count;
}

double Entropy::GetValue(bool normalized) {
   if(normalized) {
      if(m_count <= 1) {
         return m_value;
      } else {
         // entropija se normalizuje na vrednosti od 0 do 100
         return m_value * 100.0 / log2((double)m_count);
      }
   } else {
      return m_value;
   }
}
```

4.2.1 Testiranje algoritma za računanje entropije na primeru iz ns3 simulatora



Slika 16: Izgled entropije portova tokom periodičnog napada u ns3 simulatoru

Na grafiku se vidi da se pri napadu entropija naglo povećava iz razloga što se paketi šalju sa različitih portova i time se nenormalno povećava entropija i detektuje DOS napad. Ovaj grafik pokazuje da algoritam za računanje entropije radi ispravno.

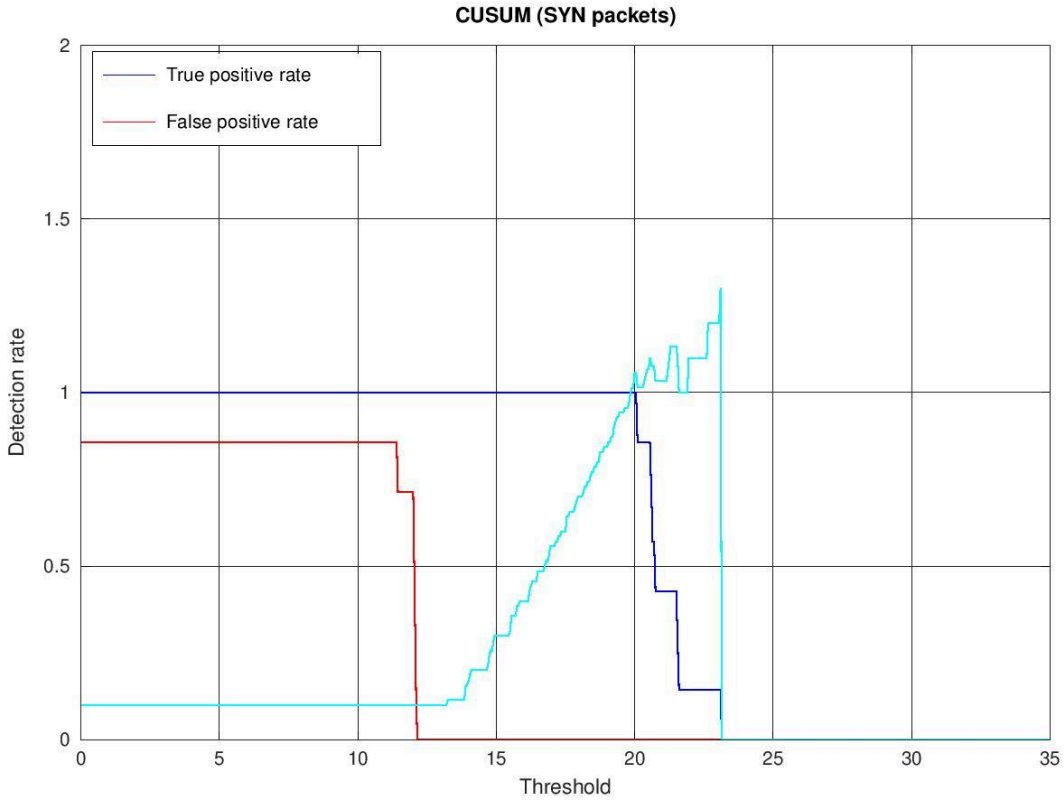


Slika 17: Izgled entropije portova iz obrade napada u realnom sistemu

Na slici 17 prikazan je realan UDP flood napad sa raspberry pi 3 računara na laptop. Ovde se takođe veoma dobro izdvaja napad koji je namerno pokrenut periodično tako da 5 sekundi traje napad, a 20 sekundi traje normalan saobraćaj gde se vrši prenos podataka sa laptopa na desktop računar. Napad se prepoznaje skokovima u entropiji.

5. Rezultati

Za analizu tačnosti algoritma potrebno je imati tačne podatke o vremenu kad počinje napad i kad se završava kako bi se mogao odrediti parametar praga detekcije DOS napada.

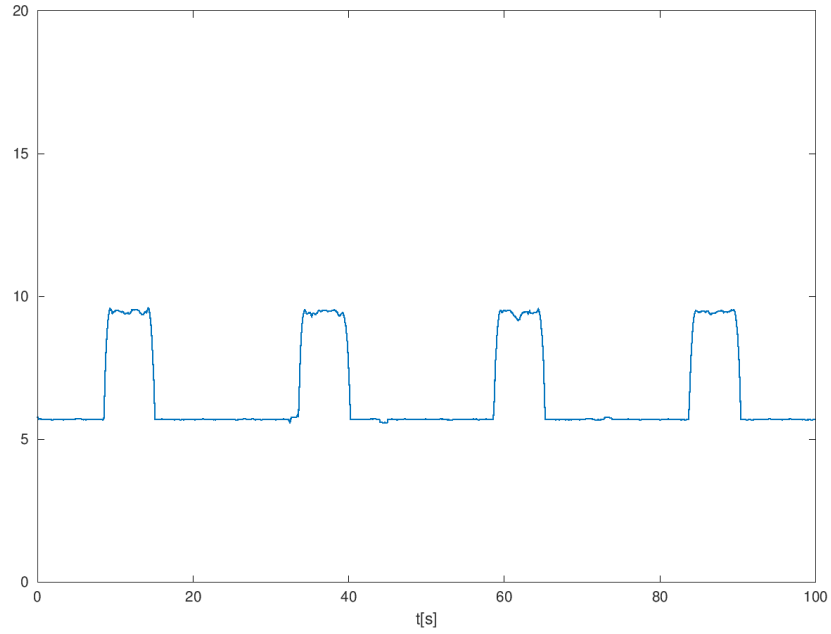


Slika 18: Analiza detekcije i kašnjenja u odnosu na izabrani prag detekcije CUSUM funkcije

Iz slike 18 se vidi da pri menjanju praga detekcije h CUSUM algoritma, dobija se određeni stepen tačnosti i određeno kašnjenje pri detekciji DOS napada. Svetlo plava linija predstavlja kašnjenje detekcije DOS napada, dok tamno plava i crvena predstavljaju tačnu i lažno tačnu detekciju.

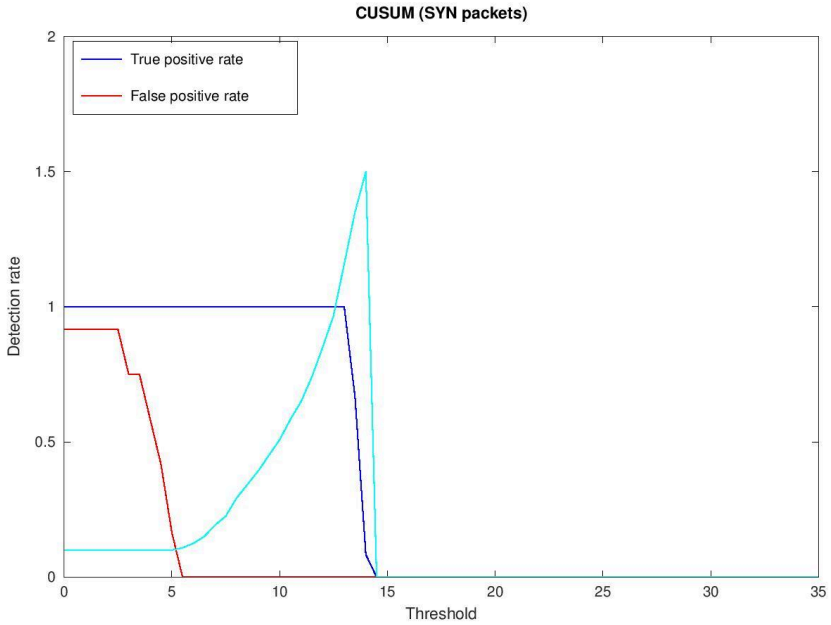
5.1 Primer napada sa raspberry pi 3 na laptop

Za ovaj napad je pokrenut napad UDP flood sa raspberry pi 3 računara na laptop koji je trajao 5 sekunde, a zatim 20 sekundi pauze odnosno normalnog saobraćaja koji je u ovom slučaju bio prenos nekog velikog fajla sa desktop računara na laptop.



Slika 19: Izgled entropije za IP adrese prilikom napada

Sa slike 19 se vidi da prilikom napada dolazi do skakanja entropije.



Slika 20: Analiza napada sa raspberry pi3

6. Zaključak

Često se za testiranje DOS napada koriste fizički uređaji za koje treba vremena da se podese i prilikom testiranja je potrebno prikupljati podatke sa svih uređaja i obrađivati ih pojedinačno. Simulator omogućava dosta praktičan način za testiranje raznih algoritama za detekciju DOS napada kroz simulaciju različitih ograničenja brzine protoka kao i različite scenarije koje inače nije lako testirati sa emulacijom tj. testiranje u realnim uslovima. Emulacija složenih mreža je previše nepraktična i sama obrada podataka uglavnom mora da se obavi nakon završetka napada, dok je u simulaciji moguće obrađivati podatke i menjati parametre tokom simulacije napada.

Prilikom istraživanja zaključio sam da ovaj simulator pruža mnoge mogućnosti za testiranje i istraživanje efikasnosti protokola, njihove mane kao i način da se uspešno detektuje i spreči DOS napad.

Postoji još mnogo mogućnosti za unapređenje ns3 simulatora u svrhe istraživanja DOS napada. Jedna od ideja bi bila uključivanje algoritma za detekciju DOS napada u sam NS3 simulator.

Simulator je takođe jedan veoma zanimljiv alat za pisanje i testiranje novih protokola pre korišćenja u realnim sistemima. Moguće je testirati čitav niz testova sa različitim scenarijima za veoma kratko vreme i sa veoma preciznim povratnim informacijama koje je zahvaljujući statističkim i animacionim modulima u ovom simulatoru jako lako pratiti i prikazati. Smatram da je ovo jako dobar test za simulaciju DOS napada koji bi u budućnosti mogao da bude izuzetno koristan.

7. Literatura

[1] 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)

[2] 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)

[3] 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)

[4] pcapng file format `https://github.com/pcapng/pcapng`

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

[6] NS3 simulator `https://www.nsnam.org`

[7] James Gross, Mesut Güneş, Klaus Wehrle: Modeling and Tools for Network Simulation

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

[10] DDoS-Scripts `https://github.com/vbooter/DDoS-Scripts`