IMPLEMENTASI LAYANAN INTRUSION PREVENTION SYSTEM UNTUK MENGATASI DENIAL OF SERVICE SYN FLOOD PADA CONTROLLER SDN

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Tatag Winaryo
NIM: 125150207111009

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2016

PENGESAHAN

IMPLEMENTASI LAYANAN INTRUSION PREVENTION SYSTEM UNTUK MENGATASI DENIAL OF SERVICE SYN FLOOD PADA CONTROLLER SDN

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Tatag Winaryo
NIM: 125150207111009

Skripsi ini telah diuji dan dinyatakan lulus pada
23 Agustus 2016
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Rakhmadhany Primananda, S.T, M.Kom	Adhitya Bhawiyuga, S.Kom, M.S
NIK: -	NIK: 201405 890720 1 001

Mengetahui
Ketua Program Studi Informatika/Ilmu Komputer

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 23 Agustus 2016

Tatag Winaryo

NIM: 125150207111009

KATA PENGANTAR

Puji dan rasa syukur yang mendalam penulis panjatkan kehadirat Allah SWT, karena berkat limpahan rahmat, hidayah, dan inayah-Nya maka skripsi ini dapat diselesaikan dengan baik. Salam dan salawat semoga selalu tercurah pada baginda Rasulullah Muhammad SAW.

Penulis mengucapkan rasa terimakasih yang sebesar-besarnya atas semua bantuan yang telah diberikan sehingga skripsi ini dapat terselesaikan, baik secara langsung maupun tidak langsung. Secara khusus rasa terimakasih tersebut kami sampaikan kepada :

- Bapak Rakhmadhany Primananda, S.T, M.Kom selaku dosen pembimbing I yang telah memberikan bimbingan, bantuan dan menyemangati dalam meneliti topik yang dibahas serta memberi saran-saran yang luar biasa.
- Bapak Adhitya Bhawiyuga, S.Kom, M.S selaku dosen pembimbing II yang telah membantu dalam penulisan skripsi hingga selesai serta memberi saran-saran yang luar biasa.
- Bapak Issa Arwani, S.Kom, M.Sc selaku Kepala Prodi Informatika/Ilmu Komputer.
- Seluruh dosen dan karyawan Fakultas Ilmu Komputer, Universitas Brawijaya atas ilmu, bimbingan dan bantuannya hingga penulis selesai menyusun skripsi ini.
- Rekan-rekan LKI-AMD dan kelas H semester 1 yang telah menginspirasi dan berbagi pengalaman bersama.
- Orang tua yang selalu mendukung, karena tanpa dukungannya penulis tidak akan bisa menempuh masa-masa perkuliahan ini.
- Calon istri yang menjadi salah satu alasan untuk segera menyelesaikan skripsi ini.

Penulis menyadari skripsi ini tidaklah sempurna, baik dari materi maupun penyajiannya. Untuk itu saran dan kritik yang membangun dan penelitian lebih lanjut sangat diharapkan untuk menutupi kekurangan pada skripsi ini..

Malang, 23 Agustus 2016

Penulis

tatagwinaryo@gmail.com

ABSTRAK

Penelitian ini dilatar belakangi oleh adanya arsitektur jaringan terkini yang dikenal dengan *software defined networking* atau disingkat SDN. Pada SDN, lalu lintas jaringan dapat diprogram pada sebuah *controller* sehingga dapat dikendalikan dengan mudah, ditambah lagi dengan adanya beberapa bahasa pemrograman yang mendukung SDN. Pada penelitian ini, salah satu teknik keamanan jaringan *intrusion prevention system* atau disingkat IPS yang dibangun menggunakan bahasa pemrograman python bertujuan untuk meminimalisir serangan yaitu *denial of service* atau disingkat DoS. DoS sendiri bekerja dengan menghabiskan *resource* korban. Pada penelitian ini DoS yang diteliti adalah s*yn flood*, yaitu DoS yang menghabiskan resource korban dengan mengirim paket *syn* secara terus menerus kepada korban. IPS tersebut diimplementasikan pada *controller* jaringan SDN. Untuk membangun IPS, penulis akan melakukan pem-blokiran *host* penyerang dengan menganalisa paket-paket yang dikirim. Adapun informasi yang di analisa dari paket-paket tersebut adalah *Mac Address* dan *IP Address* serta jumlah paket TCP SYN yang dikirim. Penelitian ini dikerjakan dengan merancang dan mengimplementasikan IPS pada arsitektur jaringan SDN, dimana keberhasilan IPS tersebut ditentukan dengan DoS *syn flood* yang dicobakan agar dapat diatasi oleh IPS yang diimplementasikan. Penelitian ini membahas bagaimana mengimplementasikan IPS pada SDN dengan controller pyretic. Hasil yang diperoleh pada penelitian ini cukup memuaskan, karena sesuai dengan harapan yaitu dapat mengurangi efek dari DoS. Dengan melakukan beberapa percobaan yaitu menyerang korban beberapa kali dengan jumlah penyerang yang berbeda disetiap percobaannya, dan dengan merata-ratakan hasil dari masing-masing percobaan tersebut diperoleh rata-rata keefektifan dari sistem yang telah diimplementasikan yaitu mencapai 95%, walaupun terdapat beberapa hal yang perlu diteliti lebih lanjut seperti membangun IPS pada SDN dengan *load balancing*, menambah keefektifan sistem dengan menganalisa *source code*-nya ataupun menganalisa desain sistem IPS yang telah dibangun agar lebih mudah digunakan.

Kata kunci: SDN, IPS, pyretic, DoS, keamanan, Jaringan

ABSTRACT

This research is motivated by current network architecture, known as software-defined networking or SDN. On SDN, network traffic can be programmed in a controller so that it can be controlled with ease, coupled with the presence of some programming language that supports SDN. In this study, one of the techniques of network security intrusion prevention system or IPS built using python programming language that is aimed at minimizing the risk of a denial of service or DoS. DoS works by victims spend resources. In this study examined are DoS syn flood, ie DoS that spend resources on victim by sending SYN packets continuously to the victim. IPS is implemented at the network controller SDN. To build the IPS, the author will block the attacker host by analyzing packets sent. The information analyzed from such packages are Mac Address and IP Address and the number of TCP SYN packets sent. The research was done by designing and implementing IPS on SDN network architecture, which is determined by the success of IPS DoS syn flood that would be attempted to be resolved by implementing IPS. This study will discuss how to implement the IPS in SDN with pyretic controller. The results obtained in this study is quite satisfactory, because as expected that it can reduce the effects of DoS. By doing some experiments that attacked the victim several times with a different number of attackers in each experiment, obtained an average effectiveness of the system has been implemented which reached 95%. Although there are some things that need to be further investigated as IPS building on SDN with load balancing, increase the effectiveness of the system by analyzing its source code or analyze the IPS system design that has been built to make it easier to use.

Kata kunci: SDN, IPS, pyretic, DoS, security, networking

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR ALGORITMA

BAB 1 PENDAHULUAN

1.1 Latar belakang

Software Defined Networking atau yang sering dikenal dengan singkatan SDN adalah arsitektur baru yang dinamis, dapat dikelola(diprogram), hemat biaya, dan mudah beradaptasi, sehingga ideal untuk trafik yang tinggi[1]. SDN memiliki *controller* sebagai pusat pengendali keseluruhan aktivitas seperti *routing*, *forwarding* dan lainnya. *Controller* terpisah dari *switch-switch* dan untuk mengendalikan semuanya, *controller* dapat diprogram. Letak *controller* pun terdapat pada PC atau *hardware* tersendiri.

Tidak seperti arsitektur jaringan lainnya, SDN bersifat *open vendor*. Namun, SDN yang bersifat open vendor ini masih sangat baru sehingga memiliki banyak celah keamanan atau *vulnerability*. Celah keamanan SDN masih sangat terbuka, bahkan untuk *common threat* seperti yang sering terjadi pada jaringan pada umumnya yaitu *Denial of Service* atau DoS. DoS adalah upaya untuk menghabiskan *resource* target dengan tujuan agar target tidak dapat menjalankan tugasnya dengan benar. Salah satu jenis DoS adalah *SYN Flood*, yaitu pengiriman paket palsu TCP SYN yang diarahkan korban[3]. Serangan ini mengekploitasi mekanisme *three way handshake,* sehingga terjadi gangguan pada korban karena kehabisan *resource*. Pada umumnya, *server* akan membalas dengan paket SYN/ACK setelah menerima paket SYN dari *client*, lalu *client* akan mengirim lagi paket ACK untuk menjalin koneksi kepada *server.* Namun pada serangan ini, *client* tidak mengirim paket ACK sehingga *server* akan menunggu. Jika aktivitas ini dilakukan secara terus-menerus dalam waktu singkat, maka akan menyebabkan gangguan pada kinerja *server*.

Dalam penelitian ini penulis berusaha mengatasi masalah diatas terkait serangan umum DoS pada arsitektur SDN dengan menggunakan *Intrusion Prevention System* atau sering disingkat dengan IPS pada arsitektur SDN tersebut. IPS sendiri adalah sistem yang melakukan proses deteksi ancaman dan mencoba untuk menghentikan insiden yang terdeteksi sebisa mungkin[4]. IPS yang akan dibuat akan memanfaatkan kemampuan atau sifat *programmable* pada *controller* SDN dengan menganalisa paket-paket yang dikirim oleh penyerang seperti *IP Address, Mac Address* serta jumlah paket-paket TCP SYN yang dikirimkan dalam waktu tertentu. Pada dasarnya setiap *host* memiliki satu *Mac Address*, ketika sebuah *host* dideteksi memiliki *IP Address* lebih dari 5 atau mengirim paket TCP SYN lebih dari 50 dalam waktu singkat maka *host* tersebut akan dianggap sebuah ancaman lalu melakukan pemblokiran terhadap *host* tersebut berdasarkan *Mac Address*-nya. Hal tersebut didukung oleh hasil uji yang penulis lakukan pada pengujian yaitu BAB 5.2.4.

Dengan penerapan IPS pada arsitektur SDN, diharapkan dapat mencegah *common threat* atau ancaman umum khususnya dalam penelitian ini adalah *syn flood attack*. Juga diharapkan dengan menjadikan IPS menjadi sebuah layanan,

lalu lintas jaringan akan lebih efisien karena yang melakukan pencegahan adalah *controller* sehingga dapat memblokir langsung terhadap *host* penyerang.

1.2 Rumusan masalah

Adapun beberapa rumusan masalah pada penelitian ini adalah sebagai berikut:

1. Bagaimana konsep atau rancangan IDS yang dibangun pada *controller* SDN agar dapat mendeteksi *DoS Syn Flood*?
2. Bagaimana cara mencegah *DoS Syn Flood* sehingga tercipta IPS pada *controller* SDN?
3. Berapa besar persentase keefektifan sistem, CPU *usage* dan *memory usage* pada *controller*?

1.3 Tujuan

Adapun beberapa tujuan pada penelitian ini adalah sebagai berikut :

1. Memahami konsep atau rancangan IDS yang dibangun pada *controller* SDN agar dapat mendeteksi *DoS Syn Flood*.
2. Mengetahui cara mencegah *DoS Syn Flood* sehingga tercipta IPS pada *controller* SDN.
3. Mengetahui seberapa besar persentase keefektifan, CPU *usage* dan *memory usage* pada *controller*.

1.4 Manfaat

Dengan adanya penelitian ini penulis berharap semoga penelitian ini dapat bermanfaat khususnya bagi para pengembang *Software Defined Networking* atau SDN terutama dalam bidang keamanan atau *security*. Penelitian ini bermanfaat agar dapat meminimalisir dampak kerusakan akibat serangan-serangan yang umum yang sangat mungkin terjadi yang dalam penelitian ini adalah serangan *DoS Syn Flood*.

Adapun manfaat bagi Fakultas Ilmu Komputer adalah agar dapat diteliti lebih lanjut mengenai SDN terutama dalam bidang keamanan jaringan, sehingga ilmu mengenai SDN dapat berkembang dan dapat di implementasikan di Indonesia. Selain itu dengan adanya materi pada penelitian dapat dijadikan materi pada pembelajaran pada mata kuliah arsitektur jaringan terkini dan keamanan jaringan.

1.5 Batasan masalah

Masalah dalam penelitian ini akan sangat luas jika diteliti secara menyeluruh. Oleh karena itu penulis memberi batasan agar masalah tidak melebar. Adapun batasan-batasan masalah adalah sebagai berikut :

1. Serangan umum yang akan dicobakan hanya *DoS Syn Flood* terhadap *host* pada arsitektur SDN.

2. Penulis hanya meneliti dan melakukan percobaan *Intrusion Prevention System* pada arsitektur SDN.
3. *Intrusion Prevention System* yang dibuat melibatkan *controller* SDN.
4. Penelitian *Intrusion Prevention System* pada *controller* SDN hanya sebatas konsep dan implementasi *virtual* atau tanpa implementasi didunia nyata pada kasus sebenarnya.
5. *Controller* SDN menggunakan pyretic.

1.6 Sistematika pembahasan

Struktur skripsi atau sistematika pembahasan yang telah penulis susun menggunakan kerangka atau susunan sebagai berikut :

a) BAB I PENDAHULUAN

Pada BAB I memuat latar belakang, rumusan masalah, tujuan, manfaat, batasan masalah, dan sistematika penulisan.

b) BAB II LANDASAN KEPUSTAKAAN

Pada BAB II akan menguraikan kajian pustaka dan dasar teori yang relevan dengan *topic* penelitian ini seperti *DoS SYN Flood*, *Software Defined Networking* dan *Intusion Prevention System*.

c) BAB III METODOLOGI

Pada BAB III akan membahas metodologi yang akan digunakan pada penelitian ini. Akan dijelaskan pula bagaimana langkah langkah-langkah yang akan dilakukan selama pengujian.

d) BAB IV PERANCANGAN DAN IMPLEMENTASI

Pada BAB IV akan membahas bagaimana sistem dirancang dan bagaimana mengimplementasi hasil rancangan tersebut.

e) BAB V PENGUJIAN

Pada BAB III akan membahas metode pengujian yang akan dilakukan pada rancangan dan implementasi yang telah dilakukan. Juga akan menguji sistem tersebut dan menganalisanya.

f) BAB VI PENUTUP

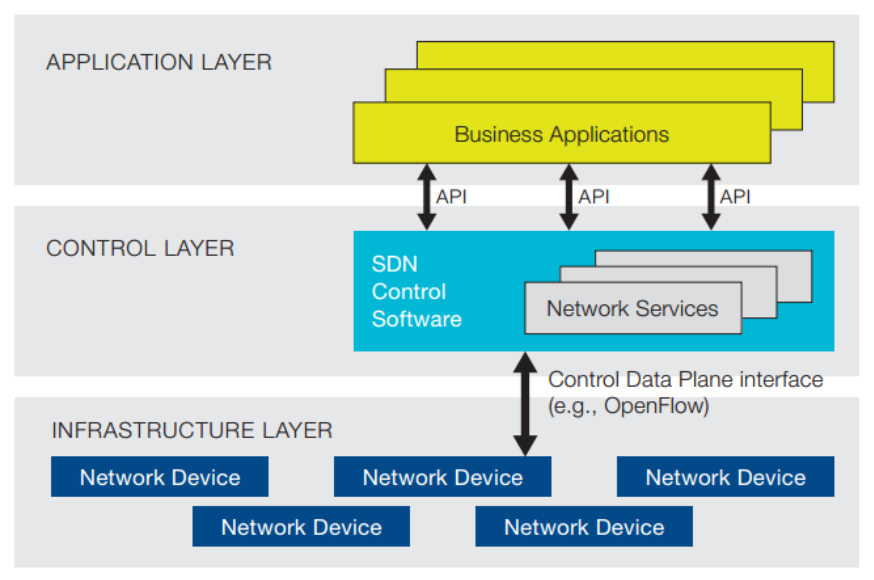
Pada BAB III berisi kesimpulan dari penelitian ini sekaligus saran yang harapannya dapat bermanfaat sebagai penelitian lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

Pada bab landasan kepustakaan ini penulis mencoba untuk membahas dan menguraikan tentang teori, konsep maupun system dari literature ilmiah yang berkaitan dengan penelitian yang penulis lakukan.

2.1 Software Defined Networking (SDN)

Software defined networking atau yang disingkat dengan SDN adalah sebuah arsitektur jaringan terbaru. SDN adalah arsitektur jaringan yang dinamis dan fleksibel. Dengan SDN, jaringan statis saat ini dapat berkembang menjadi sebuah platform yang mampu merespon dengan cepat perubahan kebutuhan bisnis, *end user*, dan pasar[2].



Gambar 2.1 Arsitektur SDN

Sumber : [2]

SDN sendiri didukung oleh beberapa bahasa pemrograman seperti *python* dan *java*. Dengan adanya beberapa bahasa pemrograman yang mendukung SDN sehingga mempermudah pengembang untuk menggunakan dan mengembangkannya. Pemrograman SDN dilakukan pada *controller*, sehingga tidak perlu melakukan akses secara langsung pada *switch* agar setiap *host* dapat terkoneksi.

Bagi penulis, SDN memiliki fitur yang sangat penting yaitu SDN merupakan arsitektur jaringan yang dapat diprogram. SDN memiliki *controller*, yang merupakan pusat dari arsitektur ini. Dalam *controller* inilah, kode-kode *program* akan ditulis. Adapun *controller* yang akan digunakan pada penelitian ini adalah *pyretic*.

2.1.1 Keamanan Pada SDN

Seperti yang telah kita ketahui, SDN merupakan arsitektur jaringan baru sehingga untuk perkembangan dimasa depan masih sangat luas. Selain itu juga, SDN bersifat *open vendor*. Namun tak dapat dipungkiri bahwa celah kemanan pada setiap arsitektur jaringan pastilah ada. Dengan asumsi jika memiliki akses ke jaringan kontrol *OpenFlow*, maka akan sangat mungkin untuk melakukan serangan seperti *ARP Poisoning* dan *TCP SYN flood*[8].

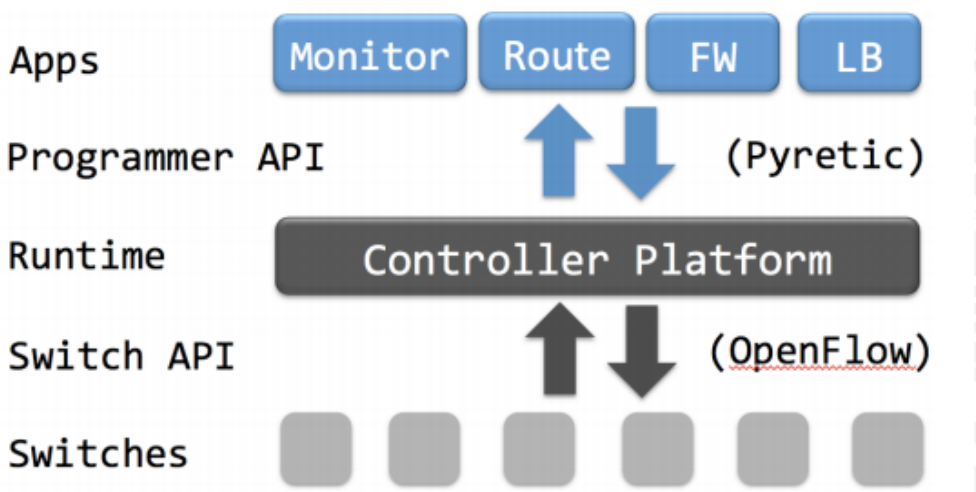
Namun, celah keamanan tersebut dapat diatasi dengan keunggulan-keunggulan SDN karena sifatnya sebagai arsitektur yang dapat diprogram. Sehingga inovasi untuk mengembangkan dan meningkatkan kemanan masih terbuka lebar. Pengembang atau *administrator* jaringan dapat menulis *program* menggunakan teknik *data mining* dan *machine learning* untuk memungkinkan identifikasi cerdas terhadap ancaman[13].

2.1.2 Controller SDN (Pyretic)

SDN *controllers* menyediakan visibilitas dan *control* terhadap jaringan, mereka dapat memastikan bahwa kontrol akses, rekayasa lalu lintas, kualitas layanan, keamanan, dan kebijakan lainnya yang diberlakukan secara konsisten di seluruh infrastruktur jaringan kabel dan nirkabel[2]. *Controller* SDN merupakan pusat dari arsitektur SDN. Sehingga, segala aktivitas atau *behavior* pada jaringan akan dikontrol oleh sebuah *controller*. Kita dapat memprogram jaringan pada SDN melalui kode-kode yang kita tuliskan pada modul dalam *controller*. *Controller* sendiri memiliki banyak versi Bahasa pemrograman. Salah satunya adalah *pyretic* yang berbasiskan Bahasa *python*.

Fungsi *controller* pada SDN adalah untuk memanajemen jaringan dan didukung oleh sifat *programmable*-nya, sehingga dapat tercipta jaringan cerdas (*intelligent networking*). *Controller* akan mengendalikan semua *switch* yang terhubung agar *switch* tahu kemana paket-paket yang datang akan diteruskan atau dikirim. Tidak hanya itu, *controller* juga dapat melakukan *drop* paket dari *host* tertentu bahkan memblokir *host* tertentu.

Dengan kemampuan *programmable* pada SDN, *controller* dapat dimanfaatkan sebagai perlindungan pertama dari serangan-serangan umum. Pada *pyretic*, memiliki *support* untuk instalasi *rule*, yang akan mengatur *switch* sebelum mereka dibutuhkan, untuk menghindari *switch-controller* latency yang tidak perlu[5]. Dari pernyataan ini, dapat dikatakan bahwa kita dapat membuat *rule* pada *controller* dan lalu lintas pada *switch* dapat diatur sehingga mengamankan jaringan melalui *controller*-pun seharusnya dapat dilakukan.



Gambar 2.2 Peran controller SDN

Sumber : [5]

Adapun jenis-jenis perintah pada *controller pyretic* ada beberapa macam, perintah-perintah tersebut dapat dilihat pada tabel berikut :

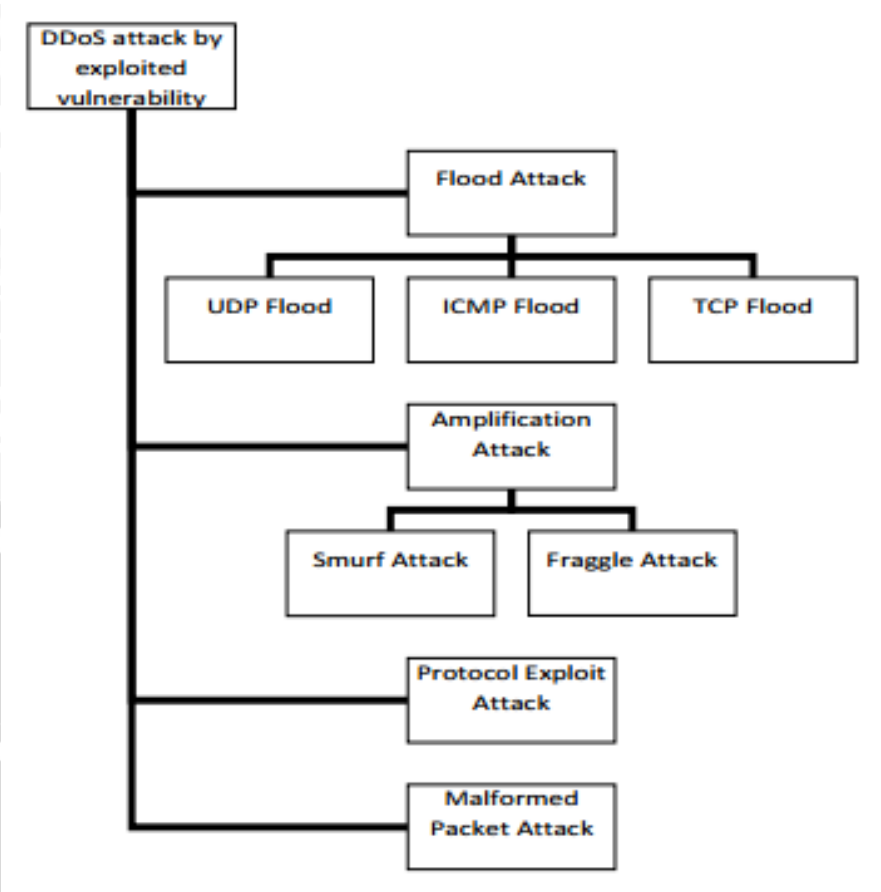
Tabel 2.1 Perintah pada pyretic

Syntax	Keterangan
identity	Return original packet
none	Return empty set
match(f=v)	identity if field f matches v, none otherwise
modify(f=v)	returns packet with field f set to v
fwd(a)	modify(port=a)
flood()	returns one packet for each local port on the network spanning tree

Sumber: [5]

2.2 Denial Of Service (DoS)

Denial Of Service adalah sebuah serangan yang sangat memungkinkan dalam dunia internet saat ini[12]. DoS adalah jenis serangan yang menghabiskan *resource* server sehingga *server* tidak dapat lagi diakses oleh *user* normal dikarenakan *server down* atau tidak berfungsi sebagaimana mestinya.



Gambar 2.3 Klasifikasi DDoS attack

Sumber : [12]

Adapun klasifikasi DoS pada umumnya adalah sebagai berikut :

a. Flood Attack

Flood attack adalah sebuah serangan yang menghabiskan *resource* target dengan membanjirinya dengan paket-paket yang sekedar dikirim namun tidak ditindak lanjuti oleh penyerang, sehingga target kerepotan menangani paket yang jumlahnya sangat banyak. *Flood attack* dibagi menjadi tiga, yaitu *UDP flood, ICMP flood dan TCP flood*.

b. Amplification Attack

Amplification attack adalah jenis serangan dimana penyerang me-*request* pada target sehingga target merespon dengan respon yang jauh lebih besar. Biasanya serangan seperti ini memanfaatkan kemampuan *broadcast*, lalu penyerang me-*request* pada *host* disuatu jaringan secara acak namun mengirim balasan ke *host* target. Contohnya adalah *smurf attack*.

c. Protocol Exploit Attack

Serangan ini memanfaatkan kelemahan *protocol* yang digunakan. Penyerang akan memanfaatkan kelemahan *protocol* atau dapat berupa bug dan menyalahgunakan hal tersebut untuk menyerang.

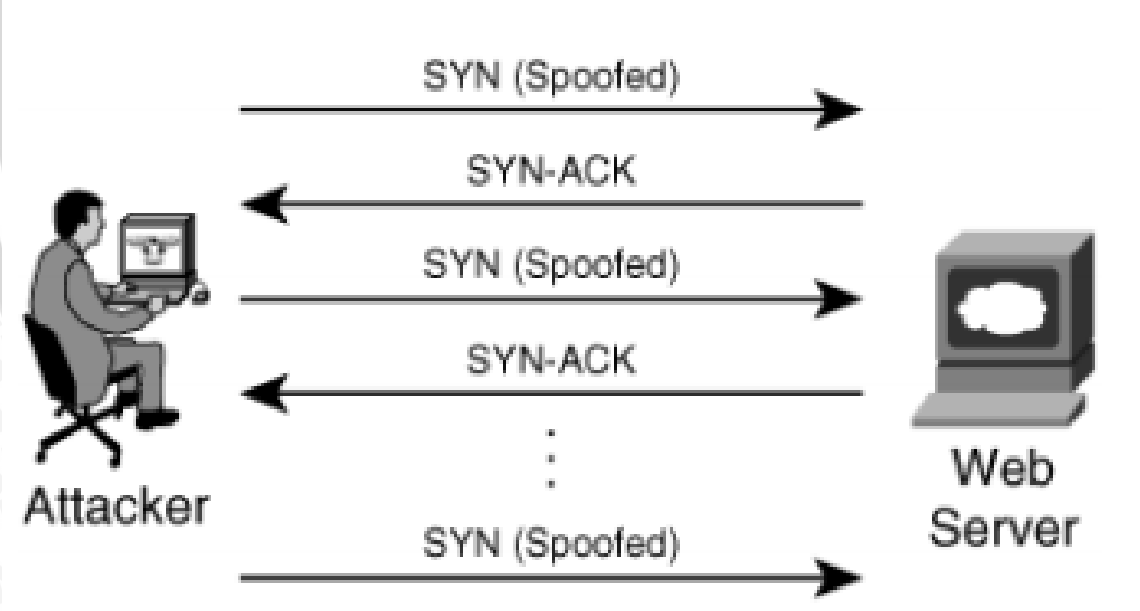
d. Malformed Packet Attack

Serangan yang memanfaatkan *errors* pada TCP/IP di system target dengan mengirimkan format paket yang tidak dikenali oleh target.

Pada penelitian yang penulis lakukan, akan berfokus pada *Flood attack* sebagai teknik serangan yang akan dicegah pada arsitektur jaringan SDN. Lebih tepatnya adalah *TCP Flood* yaitu *Syn Flood Attack*.

2.2.1 Syn Flood Attack

Syn flood attack merupakan serangan yang mengekploitasi *three-way handshake* untuk menghabiskan *resource* target. Caranya adalah dengan mengirim paket paket palsu secara terus-menerus sehingga sumberdaya target akan terkuras dan lalu target menolak untuk melakukan koneksi *TCP* lagi dari *user normal*[15]. Untuk memperparah keadaan, penyerang biasanya menggunakan IP palsu (*spoofed IP*) sehingga target akan sedikit kesulitan menganalisa paket untuk mendeteksi serangan *syn flood* dan mencegahnya.



Gambar 2.4 Syn flood attack

Sumber : [12]

Dengan begitu *syn flood attack* terbagi menjadi dua, yang pertama *direct attack* yaitu *syn flood attack* yang langsung menyerang korban dengan IP asli penyerang dan yang kedua adalah dengan *spoofed IP* yaitu dimana penyerang memalsukan IPnya terlebih dahulu sebelum nenyerang agar lebih sulit terdeteksi.

Terkait cara kerja *syn flood attack* yang lebih detail, penyerang mengirimkan paket *SYN* kepada *server*. Ketika *server* menerima paket *SYN* tersebut, sesuai skema *three way handshake* maka *server* akan membalas dengan paket *SYN/ACK*. *Server* akan menunggu balasan dari penyerang berupa paket *ACK*. Namun penyerang tidak membalasnya dan membiarkan proses tersebut berjalan sampai terjadi *TCP connection timeout* dan *three way handshake* tidak akan terpenuhi. *TCP connection timeout* akan terjadi pada umumnya dalam waktu 75 detik[3]. Penyerang terus mengirimkan paket *SYN* untuk mengulang proses tersebut sehingga tercipta keadaan dimana *server* sibuk melayani penyerang dan tidak berjalan sebagaimana mestinya.

Terdapat sebuah cara sederhana untuk mendeteksi atau mengklasifikasi paket *SYN*. *SYN flood* dapat dideteksi dengan me-*monitoring TCP state*, pada *linux* dan *windows* terdapat *command netstat* yang menampilkan status koneksi pada sebuah *host*[15]. Jika sebuah *server* memperlihatkan jumlah *connection request* "SYN_RECV" maka *server* dapat diduga sedang diserang dengan metode *Syn Flood Attack*[14]. Jika serangan langsung dengan dengan jumlah *SYN_RECV* yang cukup banyak dari *IP* yang sama, maka akan dengan mudah menghentikan serangan tersebut dengan menambahkan IP penyerang pada *firewall*[14]. Dalam kasus pada penelitian skripsi ini, penulis akan menggunakan *Mac Address* sebagai acuan alamat sebuah *host* yang melakukan serangan. Jadi, jika sebuah *host* dengan *Mac Address* tertentu mengirimkan paket *TCP SYN* dengan jumlah besar, maka *host* tersebut akan ditangani oleh sistem. Adapun jumlah paket TCP SYN yang dapat diterima oleh server korban dapat mencapai 2.000 sampai 7.000 paket dalam satu detik[15].

2.3 Intrusion Prevention System (IPS)

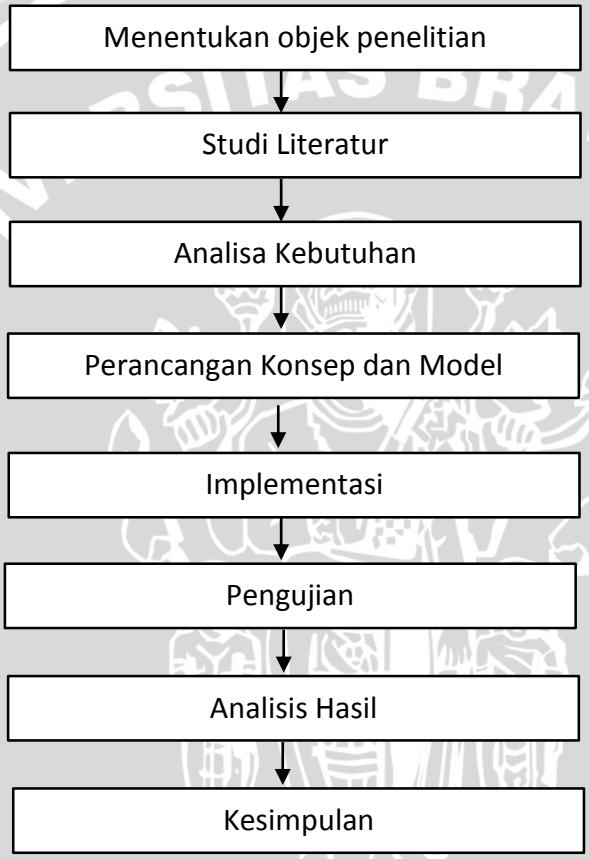
Intrusion Detection dan *prevention system* telah menjadi konsep yang dikenal di dunia keamanan TI dan telah menjadi *tool* standar dalam keamanan jaringan dan sistem komputer entah itu ancaman eksternal maupun internal[4].

Intrusion prevention system atau yang disingkat IPS adalah *tool* ataupun dapat berupa layanan yang melakukan pencegahan dini pada ancaman keamanan jaringan. Terdapat beberapa aplikasi yang menyediakan layanan IPS, salah satu contohnya saja *snort*. Namun selain *tools* yang tersedia, IPS dapat juga dibangun sesuai kebutuhan jaringan yang diinginkan.

Setiap IPS pasti juga menerapkan IDS (*intrusion detection system*), dimana IDS akan berperan sebagai sistem pendeteksi ancaman. IDS sendiri terbagi menjadi tiga jenis, yaitu *signature based detection*, *statistical anomaly based detection* dan *stateful protocol analysis detection*. Pada *signature based detection,* sistem mendeteksi dengan cara memonitor paket dalam jaringan lalu dicocokkan dengan ketentuan yang telah ditetapkan sebelumnya. Pada *statistical anomaly based detection*, sistem mendeteksi dengan cara memonitor dan menganalisa perilaku paket pada jaringan. Sedangkan *stateful protocol analysis detection* hampir sama dengan *statistical anomaly based detection,* tapi pada metode ini perilaku atau status untuk mengenali intrusion ditentukan atau dikembangkan oleh *vendor*.

BAB 3 METODOLOGI

Metode penelitian ini akan membahas tentang konsep *Intrusion Prevention System* yang dibagun pada SDN dengan melibatkan *controller* dimana nantinya akan mencegah intrusi, dalam hal ini adalah DoS berupa *SYN Flood*. Terdapat beberapa tahapan yang ada pada penelitian ini sebagai metodologi penelitian yaitu menentukan objek penelitian, studi literatur, analisa kebutuhan, perancangan konsep dan model, implementasi, pengujian, analisis hasil dan kesimpulan. Berikut diagram alir tahapan metodologi penelitian yang digunakan :



Gambar 3.1 Aliran tahap metode penelitian

3.1 Menentukan Objek Penelitian

Objek penelitian yang menjadi bahan penelitian ini adalah keamanan pada arsitektur jaringan terkini yaitu *Software Defined Networking* terhadap serangan umum yang pada kasus ini adalah serangan berupa *denial of service syn flood*. Dengan membangun *controller* berbasiskan *intrusion prevention system*, dan menjadikannya sebuah layanan sehingga *user* atau *host* yang berada pada jangkauan *controller* dapat terlindungi dari *syn flood attack* atau setidaknya dapat mengurangi serangan tersebut.

Untuk *controller*, akan menggunakan *pyretic* sebagai Bahasa pemrograman SDN. *Pyretic* sendiri dibangun dengan menggunakan Bahasa pemrograman python.

3.2 Studi Literatur

Studi literatur pada penelitian ini membahas dan mempelajari tentang *intrusion prevention system* dalam pencegahan terhadap beberapa serangan seperti DDoS. Adapun beberapa panduan ataupun referensi yang digunakan atau mendukung penelitian ini berupa beberapa jurnal atau paper yang erat kaitannya dengan *intrusion prevention system, Denial of Service,* juga SDN dan lainnya yang sejenis.

Beberapa jurnal yang erat kaitannya dengan hal diatas yang menjadi referensi penelitian ini seperti jurnal yang ditulis oleh Jason Alexander yang menjelaskan secara umum tentang IPS dengan judul jurnalnya "*Intrusion Detection and Prevention System (IDS/IPS)*", jurnal yang ditulis oleh Haining Wang yang membaghas bagaimana mendeteksi serangan SYN Flood dengan judul "*Detecting SYN Flooding Attacks*" dan juga jurnal yang dikeluarkan oleh Open Networking Foundation atau ONF yang memperkenalkan SDN pada jurnalnya yang berjudul "*Software-Defined Networking : The New Norm For Networks*".

3.3 Analisa Kebutuhan

Adapun beberapa kebutuhan pada penelitian ini yang penulis bagi menjadi kebutuhan fungsional dan kebutuhan non-fungsional :

a. Kebutuhan fungsional
   - Mendeteksi ancaman umum yang akan dicobakan yaitu mendeteksi *syn flood attack*
   - Pengiriman notifikasi dari modul pendeteksi ancaman ke modul *controller* setelah mendeteksi ancaman yang dilakukan di *server controller*
   - Memblokir paket-paket dari pelaku SYN Flood oleh *controller* sehingga pelaku tidak dapat mengakses jaringan

b. Kebutuhan non-fungsional
   - Laptop
   - *Mininet*
   - *Pyretic*
   - Mesin virtual seperti virtualbox
   - Aplikasi monitoring pada SDN seperti *sflow*

- Python dengan *library webservice* seperti *XMLRPC*
- Aplikasi kalkulasi seperti *Microsoft Excel* (jika dibutuhkan)
- Dan juga *linux Ubuntu* yang menjadi OS pada mesin virtual

3.4 Perancangan Konsep Dan Model

Perancangan konsep dan model akan menggambarkan bagaimana *system* akan dibagun dan berjalan pada penelitian ini. Gambaran inilah yang akan menjadi acuan pada system yang akan dikembangkan. Terdapat setidaknya dua modul pada *system* ini, yaitu modul *controller* dan modul *pendeteksi*.

- Modul *controller*

  Modul *controller* adalah modul yang akan mengatur semua *routing host* yang terkoneksi. Pada modul ini juga mampu melakukan *drop host*, sehingga dapat dimanfaatkan sebagai komponen IPS.

- Modul *pendeteksi*

  Sedangkan pada modul pendeteksi adalah modul yang akan memonitor dan mendeteksi paket-paket yang melintasi *switch* tertentu. Dengan begitu modul pendeteksi harus dapat bekerjasama dengan modul *controller* dengan cara memberitahu modul *controller* mengenai *host* yang akan di-*drop*.

Dengan adanya dua modul tersebut diharapkan *system* dapat berjalan dengan baik. Untuk mendukung *system* terdistribusi untuk kedua modul tersebut, membetuhkan sebuat *library* pada *python* yang dikenal sebagai *XMLRPC*. Dengan adanya *XMLRPC*, modul *controller* dan modul pendeteksi dapat berkomunikasi.

3.5 Implementasi

Implementasi akan dilakukan setelah perancangan *system* selesai dilakukan, karena implementasi sistem dalam penelitian ini akan mengacu pada perancangan tersebut. Namun dengan keterbatasan biaya, tenaga dan waktu, maka sistem hanya dapat diimplementasikan secara *virtual* tanpa adanya implementasi di dunia nyata. Implementasi akan dijelaskan secara rinci pada bab perancangan dan implementasi.

3.6 Pengujian

Pengujian akan dilakukan dengan 2 kasus uji, kasus pertama adalah pengujian dengan menyerang *server* yang dilakukan oleh *client* dengan *syn flood* dan kasus kedua adalah pengujian dengan *client* mencoba melakukan koneksi TCP biasa pada *server*. Akan diuji apakah *client* pada kedua kasus tersebut *didrop* atau tidak.

3.7 Analisis Hasil

Analisa hasil dilakukan setelah pengujian, tepatnya setelah data-data pengujian tersajikan secara lengkap. Analisis hasil dilakukan untuk menyelidiki

sabab akibat hasil didapatkan. Hal ini juga akan berguna untuk menarik kesimpulan dan saran untuk penelitian kedepannya.

3.8 Kesimpulan

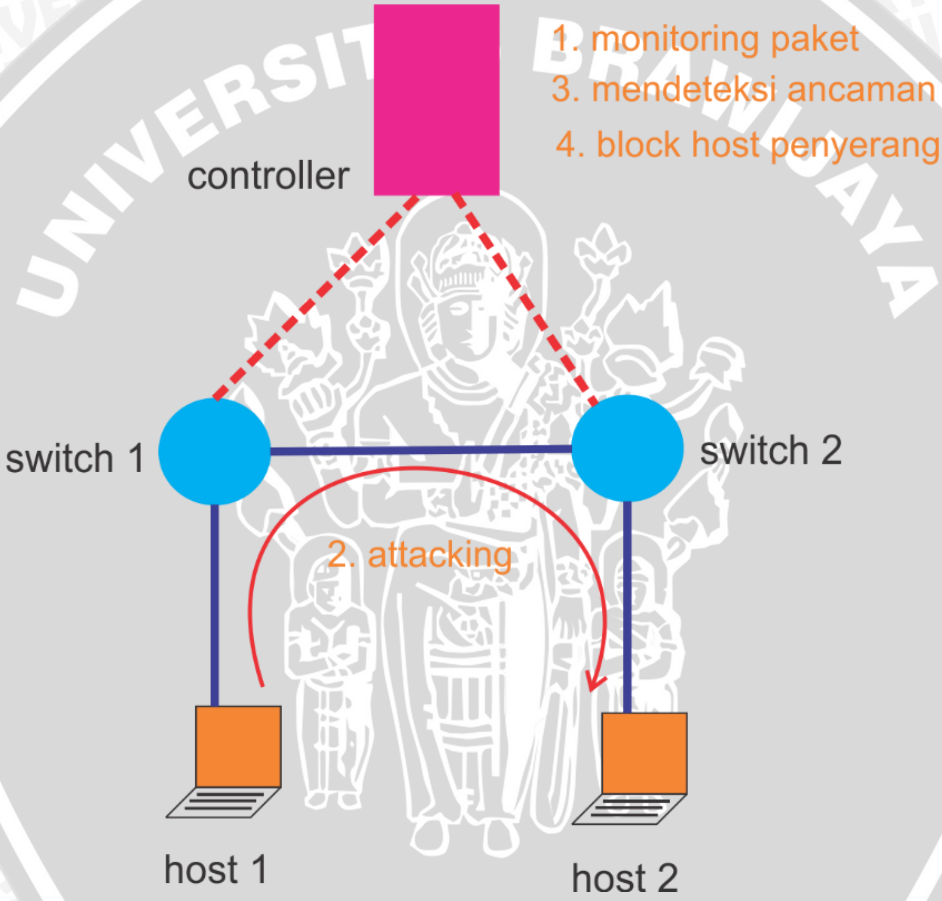
Penarikan kesimpulan hanya dapat dilakukan setelah menganalisa hasil yang diperoleh. Kesimpulan akan memuat apakah penelitian ini berhasil atau tidak dengan melihat data-data analisis hasil.

BAB 4 PERANCANGAN DAN IMPLEMENTASI

Pada bab ini penulis akan menjelaskan segala hal mengenai sistem yang dikembangkan dari perancangan sampai bagaimana implementasinya sehingga dapat berjalan.

4.1 Rancangan Arsitektur

Konsep dan model akan dibuat agar dapat tercapainya tujuan pada bab 1.3, dimana perancangan secara umum dapat dilihat pada Gambar 4.1 dibawah:



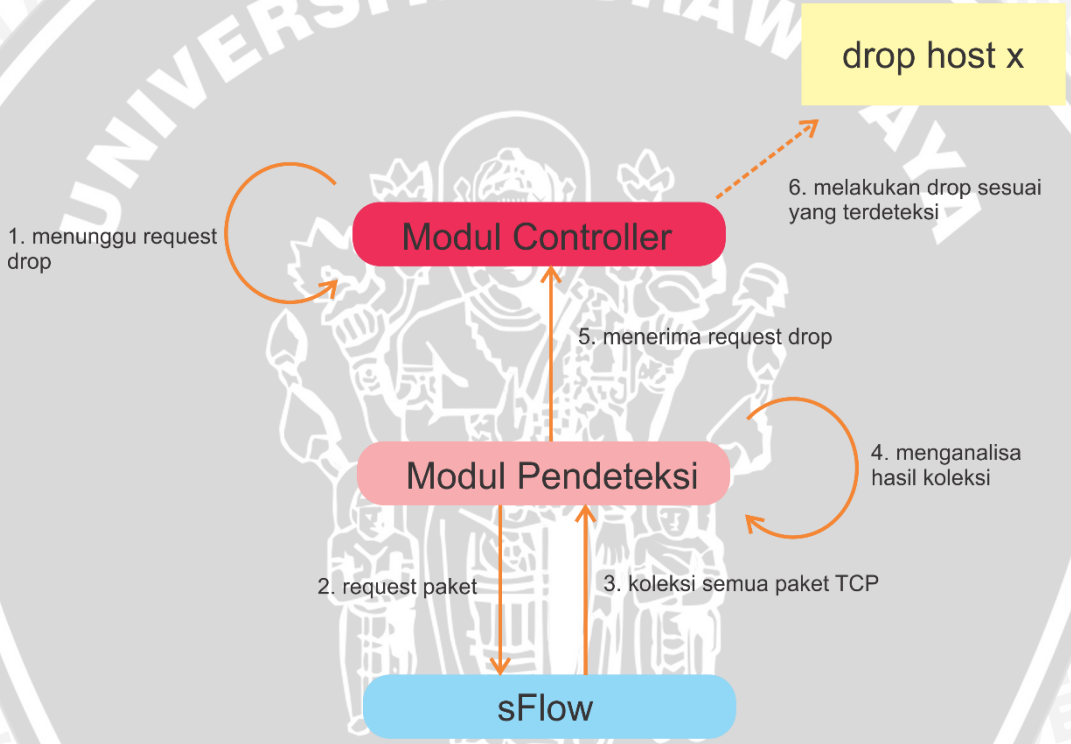
Gambar 4.1 Perancangan umum sistem

Dapat dilihat pada nomor 1 dari Gambar 4.1, *controller* mulai me-*monitoring* semua paket yang melintas pada *switch-switch*. Aplikasi *monitoring* yang dipakai pada penelitian ini adalah *sflow*. Dengan memonitor lalu-lintas jaringan, paket-paket yang melintas dapat dimonitor dan direquest oleh modul yang dibuat secara manual agar dapat dianalisa dan ditentukan apakah *host* melakukan serangan atau tidak.

Pada nomor 2 di Gambar 4.1, *host* 1 melakukan serangan pada *host* 2. Namun *controller* telah melakukan *monitoring* dan menyeleksi paket-paket yang melintas. Setelah terdeteksi, yaitu pada nomor 3 maka *host* yang terdeteksi akan ditindak lanjuti ke nomor 4 yaitu melakukan *blocking* pada *host* 1 sehingga *host* 1 tidak dapat melakukan akses keluar dari *switch* 1.

Pada SDN untuk melakukan *block* terhadap suatu *host*, maka cukup dengan melakukan update pada *dynamic policy*. Dimana *dynamic policy* untuk routing *host* tertentu akan di-*drop* (dengan pyretic cukup memanggil perintah *drop*) sehingga *host* yang di *block* tidak dapat melakukan *routing*.

Adapun rancangan yang lebih mendetail pada *controller* adalah sebagai berikut:



Gambar 4.2 Rancangan pada controller SDN

Gambar 4.2 diatas menggambarkan bagaimana *controller* dari *memonitoring* paket-paket sampai dengan melakukan *block*. Pada nomor 1, modul *controller* sudah berjalan sekaligus menunggu *request drop* dari modul pendeteksi.

Modul *controller* sebenarnya memiliki 2 *thread* dan satu proses inti. Proses inti yang dimaksud adalah proses *routing* dasar atau penetapan *dynamic policy* dasar. Sedangkan *thread* yang pertama merupakan *thread* yang menjalankan *XMLRPC* dimana *XMLRPC* ini akan dijalankan selamanya atau sampai modul *controller* dimantikan. *XMLRPC* ini memiliki *method* yang siap menerima *request* untuk melakukan perintah *drop* terhadap host tertentu pada *dynamic policy*-nya.

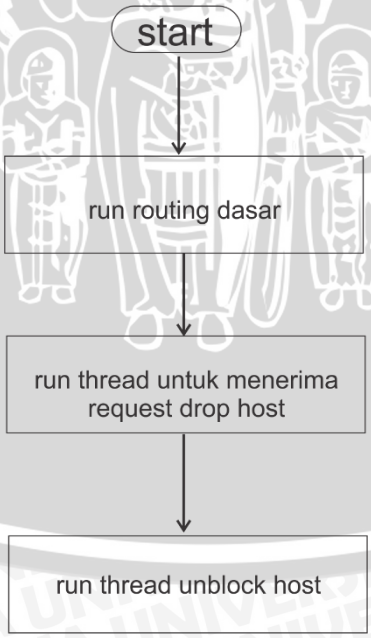
Sedangkan *thread* yang kedua merupakan fungsi yang berjalan setiap beberapa menit sekali, yaitu mereset semua *host* yang di-*block.* Sehingga dalam beberapa menit sekali *host* yang di-*block* dapat mengakses jaringan kembali.

Selanjutnya pada nomor 2 dari gambar 4.2, modul pendeteksi meminta atau *merequest* jenis paket-paket yang dibutuhkan agar dapat di koleksi. Pada nomor 3 adalah tahap dimana semua paket *TCP SYN* dikoleksi oleh modul pendeteksi dari *sFlow*. Paket-paket yang terkoleksi inilah yang akan di analisa. Untuk menganalisanya terdapat pada nomor 4. Tahap penganalisaan akan dilakukan setiap beberapa detik sekali, karena sedikit sulit untuk menganalisa jika dilakukan secara langsung.

Setelah dianalisa dan jika terdeteksi suatu *host* melakukan serangan *syn flood,* maka akan lanjut ke tahap selanjutnya yaitu pada nomor 5, modul *controller* menerima *request* untuk melakukan *drop* dari modul pendeteksi. Seperti yang telah dijelaskan sebelumnya, modul *controller* memiliki *thread* yang memanfaatkan *XMLRPC* yang didalamnya terdapat fungsi atau *method* untuk menerima *request* perintah *drop*. Ketika *request* telah diterima maka akan lanjut ketahap selanjutnya yaitu nomor 6, perintah *drop* akan dipanggil oleh modul *controller* dan mengupdate *dynamic policy*-nya.

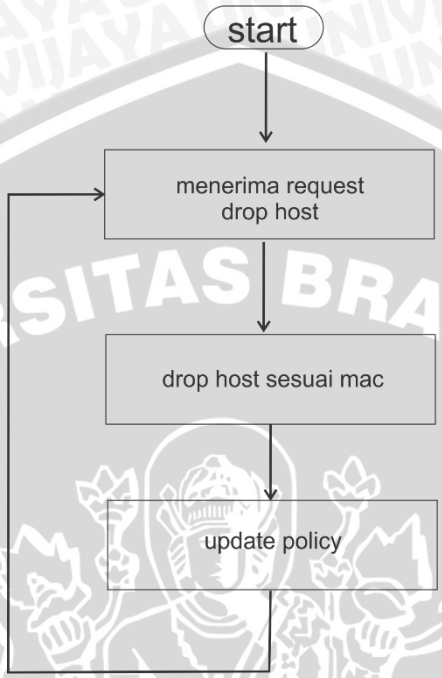
4.1.1 Modul Controller

Telah dijelaskan sebelumnya bahwa terdapat modul *controller* pada *system* ini, berikut adalah *flowchart* modul *controller* :



Gambar 4.3 Proses modul controller (proses inti)

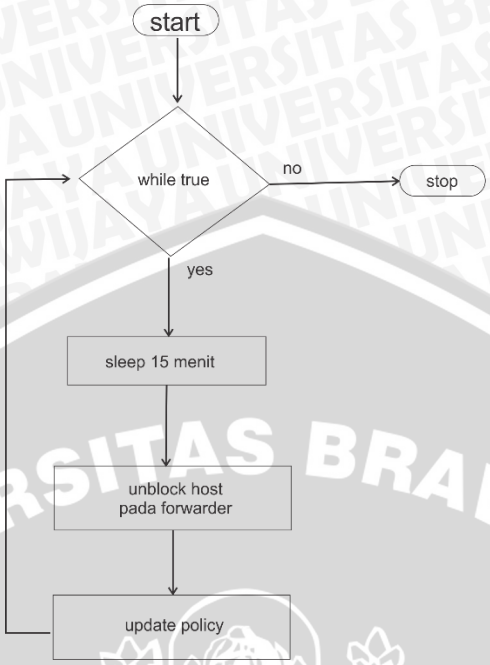
Pada Gambar 4.3 meruapkan proses inti yang terdapat pada modul *controller*. Dapat dilihat gambar tersebut terdapat *routing* dasar yang merupakan *default* dari *pyretic*, lalu terdapat 2 *thread* yang merupakan *thread* untuk menerima *request drop host* dan *thread unblock host*.



Gambar 4.4 Proses modul controller

(thread untuk menerima request drop host)

Pada Gambar 4.4 merupakan *flowchart* bagaimana proses modul *controller* bekerja saat menerima *request drop host*. Dapat dilihat ketika menerima *request drop host*, modul *controller* akan langsung melakukan *drop*. Dalam penelitian ini, hal tersebut dilakukan dengan menggunakan perintah *drop* yang disediakan oleh *pyretic*. Setelah melakukan *drop*, harus dilakukan update *policy*.

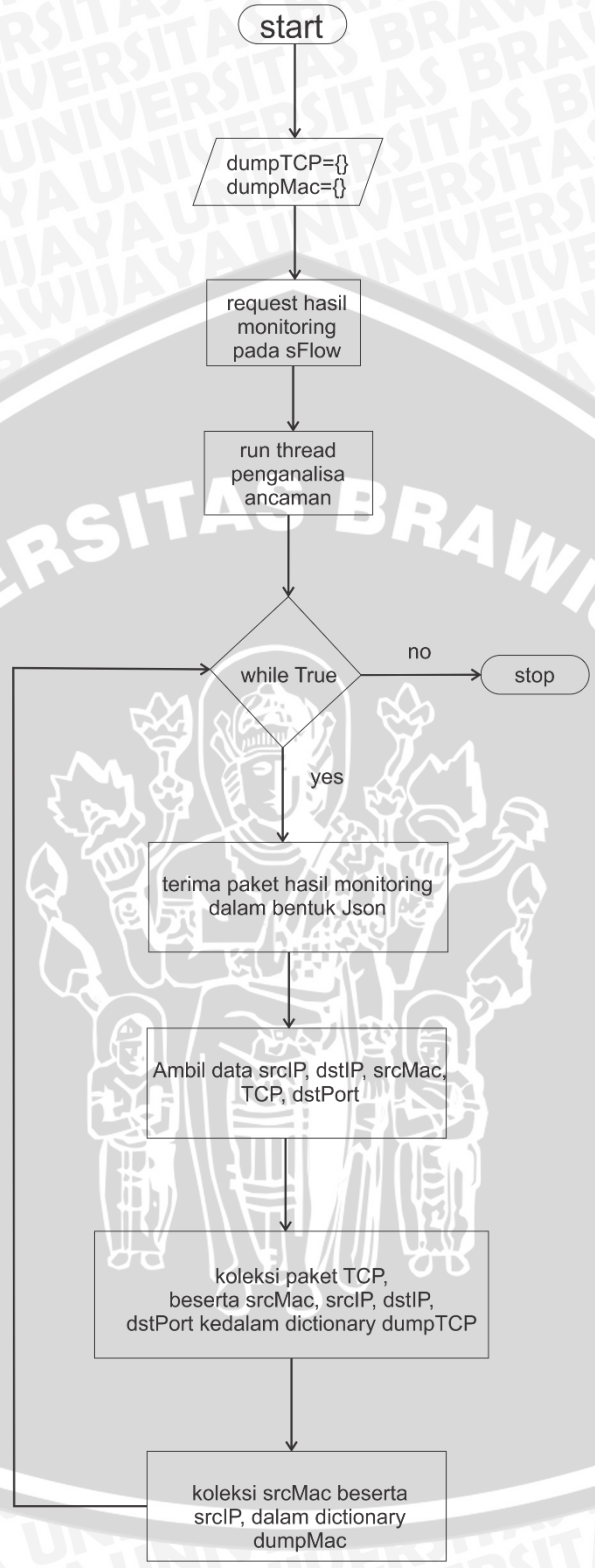


Gambar 4.5 Proses modul controller (thread unblock host)

Pada Gambar 4.5, yaitu pada *thread* penganalisa ancaman, proses akan berjalan setiap dua detik. Artinya untuk menganalisa dibutuhkan koleksi paket-paket yang *direquest* sebelumnya. Sebenarnya *interval* dua detik ini tidak baku, namun penulis mempertimbangkan suatu masalah yaitu jika *interval* terlalu lama maka target akan menghadapi masalah sebelum penyerang terblokir, namun jika terlalu cepat koleksi paket yang dilancarkan oleh penyerang tidak cukup memenuhi syarat untuk menjadi penentu pemblokiran penyerang. Seperti yang kita ketahui, pengiriman paket untuk melakukan *DoS* dapat mencapai 2000 paket per detik. Agar lebih fleksibel, interval dapat berubah. Artinya tidak baku dua detik, dan dalam penelitian ini interval tersebut dapat di ubah.

4.1.2 Modul Pendeteksi

Selain modul *controller*, juga terdapat modul pendeteksi. Seperti yang telah dijelaskan bahwa modul pendeteksi bekerja seperti IDS (*Intrusion Detection System*) yaitu mendeteksi ancaman sebisa mungkin. Berikut adalah *flowchart* modul pendeteksi :



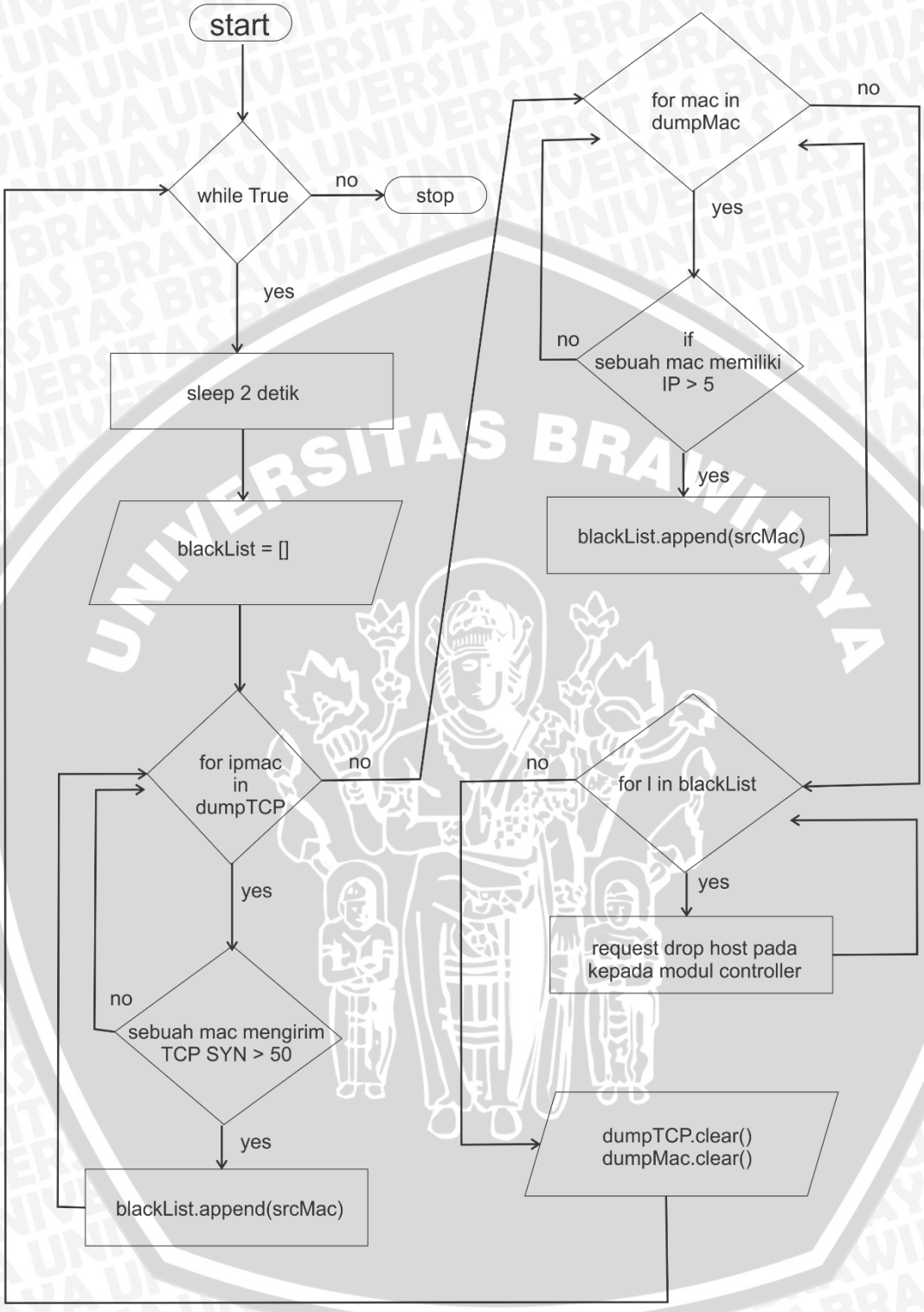
Gambar 4.6 Proses modul pendeteksi (proses inti)

Gambar 4.6 merupakan *flowchart* yang menjelaskan bagaimana proses inti pada modul pendeteksi. Proses diawali dengan menginisialisasi *variable*

dictionary dumpTCP dan *dictionary* dumpMac. *Variable dumpTCP* merupakan *variable* yang menampung jumlah paket TCP SYN yang dikirimkan oleh sebuah *host* berdasarkan *MAC Address*-nya, sedangkan *variable dumpMac* merupakan *variable* yang menampung jumlah IP *Address* yang digunakan oleh sebuah *host* berdasarkan MAC *Address*-nya.

Proses selanjutnya adalah melakukan *request* hasil *monitoring* pada *sFlow*. Proses ini akan meminta hasil *monitoring* pada *sFlow*, dimana *sFlow* berperan sebagai aplikasi *monitoring* yang memonitor trafik pada sebuah *switch*. Lalu selanjutnya adalah menjalankan *thread* penganalisa ancaman.

Setelah persiapan selesai, maka akan ada proses yang dilakukan secara berulang degnan cara melakukan *loop* tanpa batas. Proses dalam *loop* tersebut adalah menerima paket hasil *monitoring* yang dilakukan oleh *sFlow*. Lalu mengambil data berupa *srcIP, dstIP, srcMac, TCP SYN* dan *dstPort* lalu mengoleksi paket *TCP SYN, srcMac, srcIP, dstIP, dstPort* kedalam *dictionary dumpTCP* dan mengoleksi paket *srcMac*, *srcIP* kedalam *dictionary dumpMac* .



Gambar 4.7 Proses modul pendeteksi (thread penganalisa ancaman)

Jika di lihat pada Gambar 4.6 yang merupakan proses inti, pada *flowchart* tersebut terdapat proses yang menjalankan *thread* penganalisa ancaman. Pada *thread* itulah proses untuk menganalisa apakan sebuah *host* melakukan serangan atau tidak. Seiring berjalannya *thread* penganalisa ancaman, berjalan pula proses

inti yang melakukan *looping*. Dalam *loop* tersebut terdapat proses menampung atau *dump* TCP dan *source* MAC.

Ketika *source* Mac dan TCP SYN sudah ditampung, maka akan dianalisa oleh *thread* penganalisa ancaman dalam beberapa detik sekali. Perhatikan pada Gambar 4.7, proses analisa akan dilakukan setiap 2 detik. Proses analisa penentuan ancaman DoS dilakukan dengan memperhitungkan banyaknya TCP SYN yang dikirim dan banyaknya IP yang dipakai dalam 2 detik tersebut dimana hal ini didukung oleh pengujian pada BAB 5.2.4. Yang pertama berdasarkan paket TCP SYN yang dikirim, yaitu jika satu *host* dengan satu *MAC address* mengirim jumlah paket TCP SYN lebih besar dari 50, maka akan dimasukkan kedalam *blacklist*. Pada dasarnya sebuah *host* dapat mengirim paket TCP sebanyak 10905 dalam 2,3 detik (lihat Gambar 4.8), hal ini menjadi dasar bahwa untuk mendeteksi sebuah ancaman *DoS syn flood* adalah dengan melihat jumlah paket TCP yang yang dikirim sebanyak kurang dari 10905. Namun ketika parameter ini terlalu kecil maka akan semakin mudah sebuah *host* terdeteksi sebagai ancaman, sehingga semakin besar terjadi kemungkinan *host* yang mengirim paket TCP biasa juga terdeteksi sebagai ancaman, hal ini dapat dilihat pada Tabel 5.6 tentang hasil uji TCP normal yang menunjukan bahwa sebuah *host* dengan mengirim paket TCP biasa sebanyak 30 dalam 2 detik sudah terdeteksi sebagai ancaman. Kemungkinan kedua, yaitu jika satu *host* dengan satu *MAC address* namun menggunakan lebih dari 5 *IP Address*, maka akan dimasukkan kedalam *blacklist*.

Gambar 4.8 Jumlah paket TCP yang dikirim dalam 2,3 detik

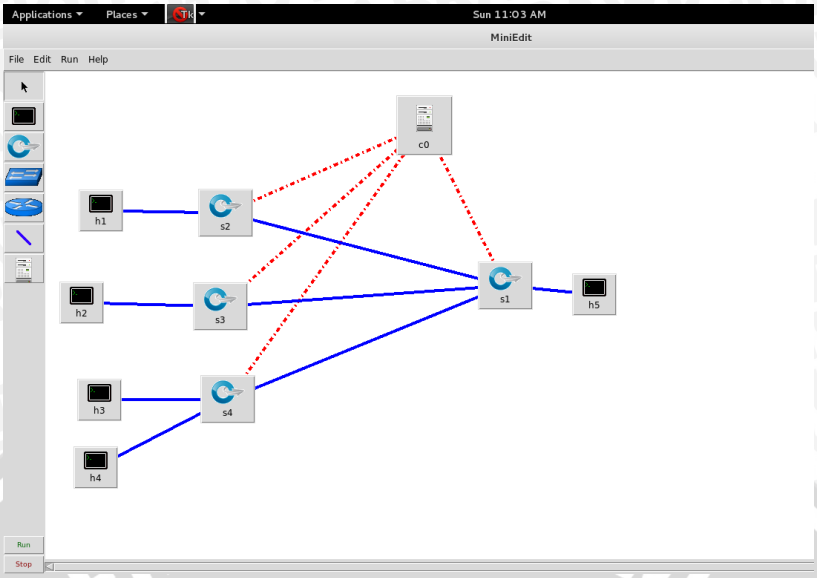
Blacklist adalah *list host* yang dianggap sebagai ancaman. Ketika analisa sudah selesai, maka tahap selanjutnya adalah melakukan *request drop host* kepada modul *controller*. Setelah selesai, *dumpTCP* dan *dumpMac* harus dikosongkan.

4.2 Implementasi

Pada sub-bab ini, penulis akan menjelaskan mengenai implementasi dari konsep yang telah dirancang sebelumnya. Adapun yang akan dibahas adalah topologi yang digunakan, modul *controller*, modul pendeteksi dan bagaimana mendistribusikan modul pendeteksi dengan modul *controller* agar dapat bekerjasama.

4.2.1 Topologi

Adapun *topologi* yang digunakan pada penelitian ini terdiri dari 1 *controller*, 4 *switch* dan 5 *host*. *Topologi* tersebut dapat dilihat pada Gambar 4.9.



Gambar 4.9 Topologi yang digunakan

Pada Gambar 4.9, merupakan dasar dari topologi pada penelitian ini. Namun, untuk menunjang penelitian ini terutama saat pengujian, *host* dapat bertambah sesuai kebutuhan. Bertambahnya *host* hanya pada *switch* s2, s3 dan s4, karena pada switch 1 merupakan switch *monitoring*.

4.2.2 Implementasi Modul Controller

Seperti yang telah dijelaskan pada rancangan arsitektur bahwa modul *controller* memiliki 2 *thread* yaitu *thread* untuk menerima request *drop host* dan *thread unblock host* serta 1 proses inti yaitu proses *routing* dasar.

Pada dasarnya, dalam *routing* dasar memiliki kode *forwarder* sebagai berikut :

```
1  self.forward = if_(match(srcmac=pkt['srcmac']),
2                          self.flood,
3                          self.forward)
4
```

Algoritma 4.1 Forwarder pada pyretic

Format kode *forwarder* tersebut akan digunakan sebgai perintah *drop* dengan mengubah bagian tertentu dengan memanggil fungsi *drop*, dan juga menentukan *host* yang di *drop* dengan memasukan informasi *mac address* kedalam *forwarder* tadi. Lalu dari dengan berubahnya forwarding tadi, maka policy juga harus di update. Jika *request drop* dilakukan maka *forwarder* suatu *host* akan menjadi seperti pada kode berikut :

```
1  self.forward = if_(match(srcmac=EthAddr(macFix)),
2                     match(srcmac=EthAddr(macFix))>>self.drop,
3                     self.forward)
```

Algoritma 4.2 Perintah drop pada forwarder

Untuk *thread* penerima *request drop*, berdasarkan diagram *flowchart* yang telah dibuat, maka kode *program* akan menjadi seperti dibawah ini :

```
1   def blockAct(self,mac):
2           temp=list(mac)
3           macFix=temp[0].lower()+temp[1].lower()+":"+\
4                  temp[2].lower()+temp[3].lower()+":"+temp[4].lower()+\
5                  temp[5].lower()+":"+\
6                  temp[6].lower()+temp[7].lower()+":"+\
7                  temp[8].lower()+temp[9].lower()+":"+\
8                  temp[10].lower()+temp[11].lower()
9           print "blocking host with mac : ",macFix
10          self.forward = if_(match(srcmac=EthAddr(macFix)),
11                             match(srcmac=EthAddr(macFix))>>self.drop,
12                             self.forward)
13          self.update_policy()
14
```

Algoritma 4.3 Thread penerima request drop

Sedangkan *thread* kedua, yaitu *thread* untuk *unblock host* yang sebelumnya telah di *block* oleh *system* akan mengubah lagi *forwardernya* dengan mengupdate *forward* menjadi *flood*. Kode dapat dilihat sebagai berikut :

```
1  def unblockAct(self):
2          while True:
3              #900 = 15 menit
4              time.sleep(900)
5              self.forward=self.flood
6              self.update_policy()
```

Algoritma 4.4 Thread unblock host

Dapat dilihat pada baris 7, untuk melakukan *unblock* cukup menggantinya dengan *flood*.

4.2.3 Implementasi Modul Pendeteksi

Pada modul pendeteksi, terdapat proses inti dengan memiliki satu *thread* untuk menganalisa paket. Proses inti akan melakukan *request* pada *sFlow* untuk meminta paket-paket yang telah dimonitor oleh *sFlow* agar dapat dianalisa. Untuk meminta paket pada *sFlow* perlu menginisiasi *dictionary flow*, lihat kode dibawah :

```
flow = {'keys':'ipsource,ipdestination,tcpsourceport
                ,tcpdestinationport,macsource,stack',
        'value':'bytes',
        'filter':'tcpflags~.......1.'
        'log':True}
```

Algoritma 4.5 Meminta data monitoring pada sFlow

Dapat dilihat bahwa yang diminta atau di *request* oleh proses inti adalah *ipsource*, *ipdestination*, *tcpsourceport*, *tcpdestinationport*, *macsource* dan *stack* (pada stack terdapat informasi bahwa paket merupakan paket tcp). Namun untuk menghubungkan modul pendeteksi dengan *sFlow*, penulis menyarankan untuk membacanya di artikel lain.

Selanjutnya pada proses inti terdapat fungsi untuk mengoleksi paket. Hal ini dilakukan setelah menerima paket yang dimonitor oleh *sFlow*. Adapun *pseudocode* program adalah sebgai berikut :

```
    For f in flows do
            splitedFlowKey ← f['flowkeys'].split(",")
            srcIP ← splitedFlowKey[0]
            ipDst ← splitedFlowKey[1]
            srcMac ← splitedFlowKey[4]
            srcTCP ← splitedFlowKey[5].split(".")
            dstPort ← splitedFlowKey[3]

            dumpTcp.setdefault("{mac},{ip},{ipdst},{portdst}".format(mac=srcM
            ac,ip=srcIP,ipdst=ipDst,portdst=dstPort),[]).append(srcTCP[2])

            cekIP ← True
            If srcMac in dumpMac then
                    For l in dumpMac[srcMac] do:
                            If l == srcIP then
                                    cekIP=False #jika false maka ada yg sama
                    end For
            If cekIP == True then #jika tidk ada yg sama maka ditambah ke dump
                    dumpMac.setdefault(srcMac,[]).append(srcIP)
end For
```

Algoritma 4.6 Mengoleksi data paket hasil monitoring

Pseudocode diatas merupakan algoritma untuk mengoleksi paket yang dimonitor. Pertama, informasi sebuah paket tersimpan pada *variable flowkeys*. *Variable flowkeys* akan di-*split* berdasarkan tanda ","(koma) dan setiap *value*-nya disimpan pada *variable srcIP* (menyimpan *value source IP*), *ipDst* (menyimpan *value destination IP*), *srcMac* (menyimpan *value source Mac*), *srcTCP* (menyimpan informasi *TCP*) dan *dstPort* (menyimpan *value destination Port*).

Untuk mengoleksi *TCP* SYN yang dikirimkan sebuah *host*, beberapa *value* tersebut akan dibentuk menjadi *key* pada *dictionary* agar bersifat unik (menunjukan identitas sebuah *host*) sehingga dapat mengoleksi paket *TCP* SYN yang dikirim pada sebuah *host* tersebut dengan melakukan perintah *append*. Hal ini dilakukan untuk menghadapi serangan *Syn flood direct attack*. Sedangkan untuk mengoleksi IP yang digunakan oleh sebuah *host*, harus memeriksa dulu apakah IP tersebut telah terkoleksi atau belum. Hal ini dilakukan untuk menghadapi penyerangan *DoS Syn flood* dengan *spoofed IP*. Ketika sebuah *host* memiliki IP yang berbeda dari yang telah dikoleksi, maka IP baru tersebut akan ditambahkan dalam *dictionary dumpMac* dengan melakukan perintah *append*.

Setelah koleksi paket, maka sesuai diagram *flowchart* yang telah digambarkan, dalam beberapa detik tertentu akan melakukan penganalisaan terhadap paket yang telah dikumpulkan. Hal ini dilakukan untuk menentukan apakah suatu *host* melakukan serangan atau tidak, jika melakukan serangan maka akan meminta modul *controller* untuk melakukan *block* pada *host* tersebut berdasarkan *mac address*. Fungsi penganalisa inipun dijalankan dalam *thread* tersendiri. Adapun *pseudocode program* penganalisa adalah sebagai berikut :

```
1       While True
2           Time.sleep(2)
3           blackList ← []
4           For ipmac in dumpTcp do
5               parsedIpmac ← ipmac.split(",")
6               getMac ← parsedIpmac[0]
7
8               If len(dumpTcp[ipmac]) > 50 then
9                   blackList.append(getMac)
10          end For
11
12          For mac in dumpMac do
13              if len(dumpMac[mac])>5 then
14                  blackList.append(mac)
15          end For
16
17          For l in blacklist do
18              Request block l, kepada modul controller
19  End For
```

Algoritma 4.7 Analisa ancaman

Berdasarkan *pseudocode* diatas, hal pertama yang dilakukan adalah mempersiapkan *list blacklist* untuk mendaftarkan *host* (berdasarkan *Mac Address*), dimana *list* tersebut merupakan daftar *host* yang akan di-*block*. Setelah itu, melakukan analisa *TCP* dengan kondisi jika *tcp* SYN pada sebuah *key* (pada *dumpTcp*) lebih besar dari 50 maka *Mac Addreass* yang mengirim paket TCP tersebut didaftarkan pada *blackList*. Selanjutnya adalah menganalisa banyaknya IP yang digunakan oleh sebuah *host* (berdasarkan *Mac Address*). Jika sebuah ada lebih dari 5 IP dengan 1 *Mac Address*, maka *host* dengan *Mac Address* tersebut

didaftarkan pada *blackList*. Setelah selesai menganalisa, maka perintah yang dilakukan selanjutnya adalah meminta (me-*request*) *controller* untuk melakukan *block* sesuai *list* pada *blackList*.

4.2.4 Mendistribusikan Sistem

Untuk mendistribusikan *system*, maka dibutuhkan suatu *library* dalam *python* yaitu *XMLRPC*. Modul *controller* akan berperan sebagai *server* yang melakukan *serving* pada *localhost* dengan port 6666. Menjalankan XMLRPC pada modul *controller* berada dalam *thread* penerima *request drop host*. Berikut adalah kode untuk *serving* :

```
1   server=SimpleXMLRPCServer(("localhost",6666))
2   server.register_function(self.blockAct,'blockAct')#serving fungsi untuk blocking
3   server.serve_forever()
```

Algoritma 4.8 Serving penerima request drop dengan XMLRPC

Sedangkan pada modul pendeteksi, akan berperan sebagai *client* yang akan melakukan *request* kepada modul *controller* di *localhost* dengan *port* 6666, lihat kode dibawah:

```
1   prox=xmlrpclib.ServerProxy("http://localhost:6666/")
```

Algoritma 4.9 Koneksi oleh modul deteksi kepada modul controller

Fungsi *request* ini seharusnya berjalan dalam *thread* penganalisa ketika *loop* terhadap *blackLis*, dan mengirim setiap isi *blackList* kepada modul *controller*. Silahkan lihat kembali sub-bab 4.2.3 , pada *pseudocode* :

```
1   For l in blacklist do
2     Request block l, kepada modul controller
3   End For
```

Algoritma 4.10 Request drop host sebelum berfungsi

Pseudocode diatas akan diganti dengan *pseudocode* seperti dibawah :

```
1   For l in blacklist do
2                 prox.blockAct(l)
3   End For
```

Algoritma 4.11 Request drop host setelah memiliki fungsi

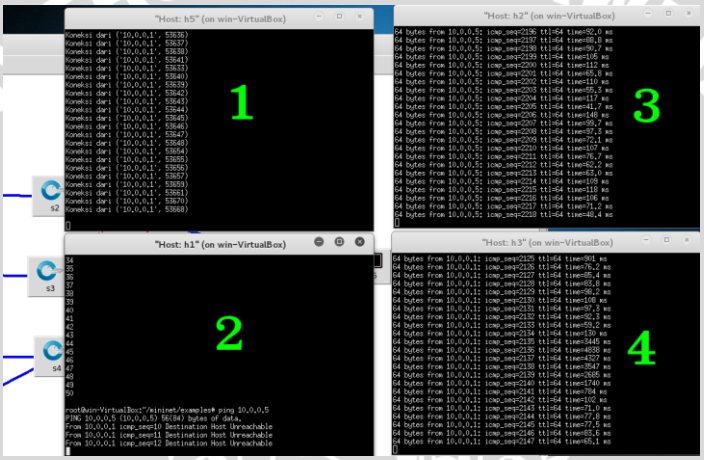
Dengan begitu modul pendeteksi dan modul *controller* dapat berkomunikasi dan *system* akan berjalan sesuai dengan apa yang telah dirancang.

BAB 5 PENGUJIAN

Pada bab ini akan membahas semua hal tentang pengujian sistem. Pembahasan akan menjelaskan secara penuh mengenai skenario, pengujian sistem dan analisis hasil.

5.1 Skenario Pengujian

Pengujian akan dilakukan berdasarkan metode uji yang telah dirancang penulis. Adapun metode untuk menentukan kebarhasilannya adalah dengan melakukan uji dengan *ping*. Jika sebuah host terblokir, maka seharusnya ping tidak dapat dilakukan oleh *host* tersebut, karena *host* penyerang tersebut telah di-*drop* oleh *system*. Berikut adalah contoh uji dengan ping jika *host* penyerang telah terblokir :



Gambar 5.1 Contoh uji dengan ping

Dari Gambar 5.1 diatas, terdapat empat *terminal*. *Terminal* 1, merupakan *terminal* yang akan berperan sebagai *host* yang menjadi *server* ataupun target penyerangan. *Terminal* 2, merupakan *client* yang akan menyerang *server* ataupun akan membuat koneksi TCP dengan *server*. Selain itu, *terminal* 2 juga akan melakukan uji ping terhadap *server* untuk mengetahui apakan dirinya masih bisa mengakses *server* atau tidak. *Terminal* 3, merupakan *terminal* yang akan melihat apakah *server down* atau tidak. Dan *terminal* 4, merupakan *terminal* yang akan melihat apakah *client* (terminal 2) terblokir atau tidak, namun untuk *terminal* 4 ini tidak terlalu penting dan hanya berlaku pada pengujian dengan satu penyerang. Untuk pengujian dengan lebih dari satu penyerang, mengecekan apakah penyerang berhasil *diblock* atau tidak dapat dilakukan langsung oleh penyerang seperti halnya yang dilakukan *terminal* 2.

Namun untuk menentukan keberhasilan tidak persis seperti yang dicontohkan pada Gambar 5.1. karena pada pengujian sebenarnya terdapat banyak terminal dimana *terminal-terminal* tersebut akan berlaku selayaknya *host-host* yang melakukan serangan.

Seperti yang telah dijelaskan pada bab 3, dalam tahap uji ini ada beberapa hal yang perlu diuji, berikut penjelasannya :

- Pengujian Tanpa IPS

  Pengujian ini akan dilakukan dengan melakukan serangan terhadap sebuah *host*. Pada kasus ini tidak ada sistem IPS yang dijalankan pada arsitektur jaringan yang ada.

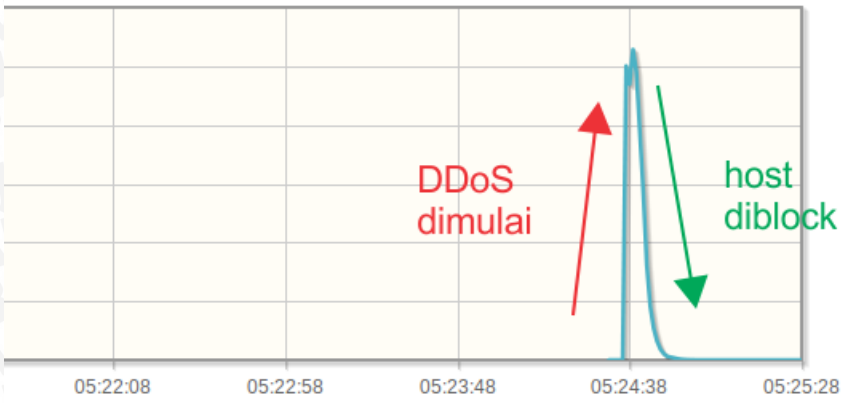
- Pengujian Dengan IPS

  Pada pengujian ini dilakukan dengan melakukan serangan *syn flood attack* yang akan dibagi menjadi dua, yaitu *syn flood direct attack* dan *syn flood* dengan *spoofed IP* . Pengujian dilakukan dengan melihat apakah *host* yang melakukan serangan berhasil di *drop* atau tidak oleh *system* yang telah dibuat.

  Pengujian akan dilakukan beberapa kali dengan jumlah *host* yang berbeda pada setiap pengujian. Jumlah host yang diujikan di setiap pengujian adalah 1 *host*, 3 *host*, 5 *host*, 8 *host* dan 10 *host*. Dari setiap pengujian juga akan dilihat berapa *host* yang berhasil *diblock* dari total *host* yang diujikan sehingga akan didapat keefektifan dari sistem.

  Pada setiap pengujiannya juga memiliki batas waktu maksimal 5 menit untuk modul pendeteksinya. Misal untuk pengujian 3 *host*, maka penyerangan dilakukan oleh 3 *host* yang dilakukan hampir bersamaan lalu menghidupkan modul pendeteksi selama kurang atau sama dengan 5 menit.

  Untuk melihat keberhasilan pada pengujian dengan *syn flood attack* dapat juga dilihat pada grafik yang memang fitur ini telah disediakan pada aplikasi *monitoring sFlow*. Grafik ini akan menggambarkan trafik *flow* pada sebuah *switch* yang dimonitor. Berikut contoh grafik ketika sebuah *host* melakukan serangan dan berhasil di *block* oleh sistem:



Gambar 5.2 Contoh grafik ketika block host berhasil

Selain itu grafik juga dapat digunakan untuk melihat trafik saat menggunakan IPS yang telah dibangun serta trafik saat tidak menggunakan IPS.

Pada pengujian ini juga akan dilakukan pengujian dengan melakukan koneksi TCP normal yang dilakukan beberapa kali dengan masing-masing pengujian terdapat jumlah koneksi TCP yang berbeda dalam waktu kurang dari 2 detik. Akan dilihat disetiap pengujian, apakah *host* pengirim di *drop* oleh *system* atau tidak. Jumlah *TCP* yang diujikan adalah 10, 11, 30, 31, 50, 51, 60 dan 61 TCP secara hampir bersamaan yaitu kurang dari 2 detik.

Adapun dalam pengujian TCP normal akan menggunakan Algoritma 5.1 sebagai server dan Algortma 5.2 sebagai client. Kedua algoritma tersebut merupakan algoritma untuk melakukan koneksi TCP sederhana pada umumnya.

```
1   #!/usr/bin/python
2
3   import socket,sys,thread
4
5   s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6   host = "10.0.0.2"
7   port = int(sys.argv[1])
8   s.bind((host, port))
9
10  s.listen(5)
11  def konek(con,addr):
12      print 'Koneksi dari ', addr
13      con.close()
14
15
16  while True:
17
18      c, addr = s.accept()
19      try:
20          thread.start_new_thread(konek,(c,addr))
21      except:
22          print "masalah"
```

Algoritma 5.1 server.py

```
1   #!/usr/bin/python
2
3   import socket,sys,thread
4
5   host = sys.argv[2]
6   port = int(sys.argv[3])
7   loop = int(sys.argv[1])
8   stoper=0
9
10  def konekto():
11      #print "masuk method konekto"
12      s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13      s.connect((host,port))
14      #print s.recv(1024)
15
16      s.close
17
18  while stoper < loop:
19      try:
20        thread.start_new_thread(konekto,())
21      except:
22        continue
23      stoper=stoper+1
24      print stoper
```

Algoritma 5.2 client.py

Untuk memahami Algoritma 5.1 dan Algoritma 5.2 silahkan mencarinya pada artikel ataupun buku lain yang membahas *fundamental* tentang *python.*

- Pengujian Skalabilitas

  Adapun pengujian skalabilitas sistem dengan mengukur *CPU usage* dan *memory usage*. Pada pengujian skalabilitas, jumlah *host* penyerang yang diujikan di setiap pengujian adalah 1 *host*, 3 *host*, 5 *host*, 8 *host* dan 10 *host*. Untuk *mengetahui CPU usage* dan *memory usage*, penulis menggunakan perintah *top* pada *linux*.

- Pengujian Rule Pada IPS

  Pengujian ini bertujuan sebagai pendukung atau dasar *rule* dimana IPS menggunakan parameter IP yang boleh digunakan oleh sebuah *host* adalah kurang dari 5 IP dan paket TCP SYN yang boleh dikirimkan sebuah *host* dalam waktu kurang dari 2 detik adalah 50 paket. Adapun pengujian yang dilakukan pada tahap ini dibagi menjadi dua yaitu pengujian parameter IP dan parameter paket TCP SYN dimana kedua pengujian tersebut akan dilihat apakah setiap pengujiannya, *client* akan terdeteksi sebagai ancaman atau tidak. Pada pengujian parameter IP, pengujian dilakukan 5 kali dengan masing-masing pengujian selama kurang dari 2 detik dan *client* menggunakan IP yang berbeda sebanyak 1, 3, 5, 7 dan 10. Pengujian

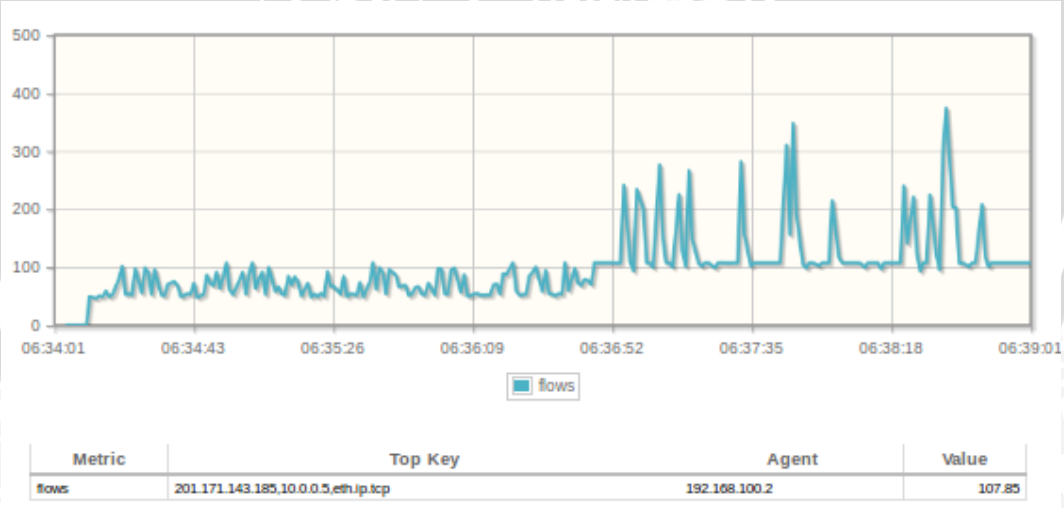
parameter IP dapat menggunakan *scapy* pada *python*. Pada pengujian parameter paket TCP SYN terbagi menjadi 2 yaitu uji jumlah paket TCP dan uji pengiriman file. Pada uji jumlah paket TCP SYN akan dilakukan sebanyak 5 kali dengan masing-masing pengujian selama kurang dari 2 detik dan *client* mengirim paket TCP SYN sebanyak 10, 30, 50, 70 dan 100. Uji jumlah paket *TCP SYN* ini juga dapat dilakukan dengan *scapy* pada *python*. Sedangkan uji pengiriman file akan dilakukan dengan mengirim sebuah file video berformat **.mkv* berukuran 73,5 MB dari host 1 dengan IP 10.0.0.1 ke host 2 dengan IP 10.0.0.2.

5.2 Pengujian

Seperti yang telah dijelaskan, pengujian akan dilakukan dengan beberapa kondisi dan setiap kondisinya akan dipisah. Kondisi-kondisi tersebut adalah pengujian tanpa IPS, pengujian dengan IPS, pengujian skalabilitas dan pengujian *rule* pada IPS.

5.2.1 Pengujian Tanpa IPS

Pengujian tanpa IPS akan memperlihatkan trafik ketika serangan dilakukan. Berikut adalah hasil uji dengan memperlihatkan trafik yang telah di dilakukan tanpa menggunakan IPS :



Gambar 5.3 trafik tanpa IPS

Pada Gambar 5.3 memperlihatkan trafik yang tinggi secara terus menerus ketika *syn flood attack* dijalankan. Hal ini terjadi karena tidak adanya sistem yang mencegah *syn flood*.

5.2.2 Pengujian Dengan IPS

Pengujian dengan IPS akan dilakukan dengan dua jenis serangan *syn flood* yaitu *direct attack* dan *spoofed IP*. Berikut ini adalah hasil uji dengan menerapkan IPS pada SDN.

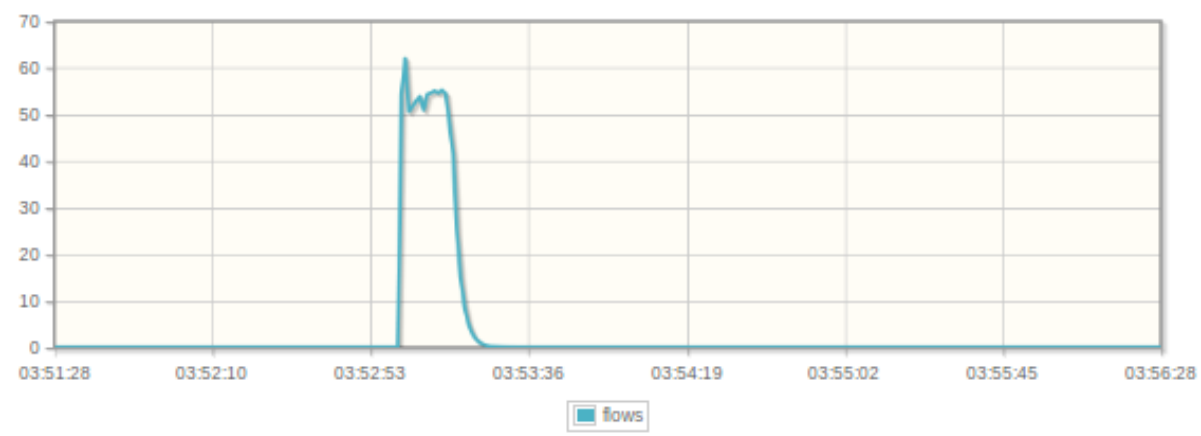
a. Uji Syn Flood

- 1 host

Tabel 5.1 Hasil uji dengan 1 host

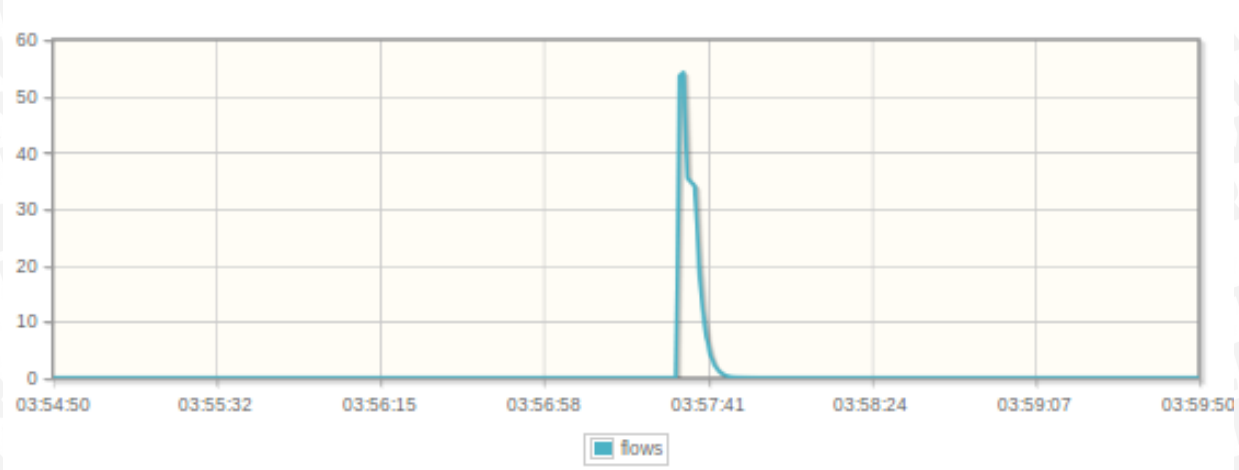
Host	Keberhasilan drop host	
	Direct Attack	Spoofed IP
1	berhasil	Berhasil

Tabel 5.1 menunjukan serangan dilakukan oleh 1 *host*. *Host* tersebut melakukan dua kali uji serangan yaitu *DoS syn flood direct attack* dan *DoS syn flood spoofed IP*. Pada uji serangan *DoS syn flood direct attack*, *host* penyerang berhasil di-*drop* dan pada uji serangan *DoS syn flood spoofed IP*, *host* penyerang juga berhasil di-*drop*.



Gambar 5.4 Trafik ketika menggunakan syn flood direct attack (1 host)

Pada Gambar 5.4, memperlihatkan bahwa *syn flood direct attack* dengan 1 *host* berhasil diatasi oleh IPS yang telah dibangun. Hal tersebut diperlihatkan dengan berhentinya trafik flow pada Gambar 5.4 ketika waktu menunjukan antara pukul 03:52:53 dan pukul 03:53:36.



Gambar 5.5 Trafik ketika menggunakan syn flood Spoofed IP (1 host)

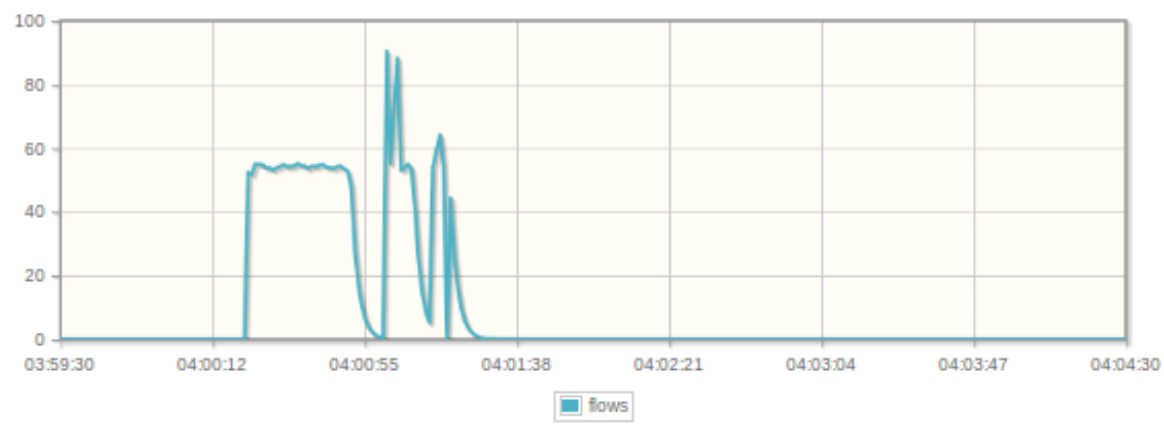
Pada Gambar 5.5, memperlihatkan bahwa *syn flood spoofed IP* dengan 1 *host* berhasil diatasi oleh IPS yang telah dibangun. Hal tersebut diperlihatkan dengan berhentinya trafik *flow* pada Gambar 5.5 ketika waktu menunjukan antara pukul 03:57:41 dan pukul 03:58:24.

- 3 host

Tabel 5.2 Hasil uji dengan 3 host

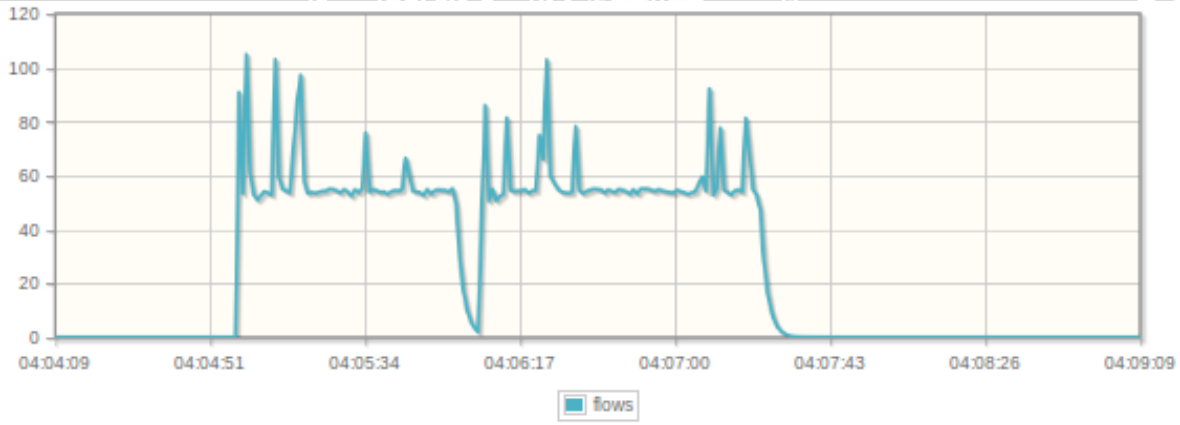
Host	Keberhasilan drop host	
	Direct Attack	Spoofed IP
1	Berhasil	Berhasil
2	Berhasil	Berhasil
3	Berhasil	Berhasil

Tabel 5.2 menunjukan serangan dilakukan oleh 3 *host*. *Host* tersebut melakukan dua kali uji serangan yaitu *DoS syn flood direct attack* dan *DoS syn flood spoofed IP*. Pada uji serangan *DoS syn flood direct attack*, ketiga *host* penyerang berhasil di-*drop* dan pada uji serangan *DoS syn flood spoofed IP*, ketiga *host* penyerang juga berhasil di-*drop*.



Gambar 5.6 Trafik ketika menggunakan syn flood direct attack (3 host)

Pada Gambar 5.6, memperlihatkan bahwa *syn flood direct attack* dengan 3 *host* berhasil diatasi oleh IPS yang telah dibangun. Hal tersebut diperlihatkan dengan berhentinya trafik *flow* pada Gambar 5.6 ketika waktu menunjukan antara pukul 04:00:55 dan pukul 04:01:38. Walaupun terdapat kenaikan trafik *flow* setelahnya, namun hal tersebut tak berlangsung lama karena dapat diatasi oleh sistem dan berhenti antara pukul 04:00:55 dan pukul 04:01:38.



Gambar 5.7 Trafik ketika menggunakan syn flood Spoofed IP (3 host)

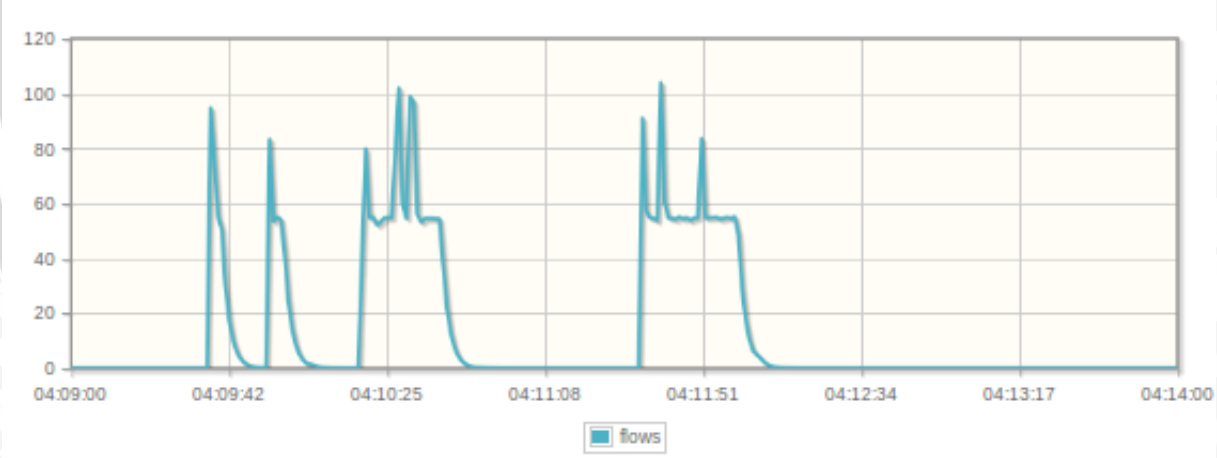
Pada Gambar 5.7, memperlihatkan bahwa *syn flood spoofed* IP 3 *host* berhasil diatasi oleh IPS yang telah dibangun. Hal tersebut diperlihatkan dengan berhentinya trafik *flow* pada Gambar 5.7 ketika waktu menunjukan antara pukul 04:05:34 dan pukul 04:06:17. Walaupun terdapat kenaikan trafik *flow* tepat setelahnya, namun hal tersebut dapat diatasi kembali oleh sistem dan berhenti antara pukul 04:07:00 dan pukul 04:07:43.

- 5 host

Tabel 5.3 Hasil uji dengan 5 host

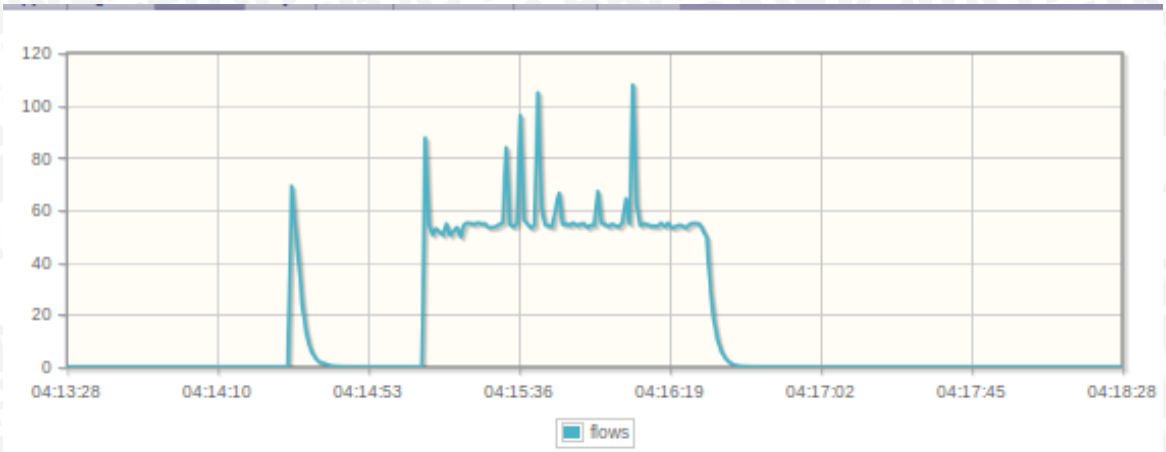
Host	Keberhasilan drop host	
	Direct Attack	Spoofed IP
1	Berhasil	Berhasil
2	Berhasil	Berhasil
3	Berhasil	Berhasil
4	Berhasil	Berhasil
5	Berhasil	Berhasil

Tabel 5.3 merupakan data serangan yang dilakukan oleh 5 *host*. Seperti yang telah direncanakan pada *scenario* uji, *host* tersebut melakukan dua kali uji serangan yaitu *DoS syn flood direct attack* dan *DoS syn flood spoofed IP*. Pada uji serangan *DoS syn flood direct attack*, kelima *host* penyerang berhasil di-*drop* dan pada uji serangan *DoS syn flood spoofed IP*, kelima *host* penyerang juga berhasil di-*drop*.



Gambar 5.8 Trafik ketika menggunakan syn flood direct attack (5 host)

Pada Gambar 5.8, memperlihatkan bahwa *syn flood direct attack* dengan 5 *host* berhasil diatasi oleh IPS yang telah dibangun. Namun ternyata terdapat beberapa jeda agar semua *host* dapat di-*drop* oleh sistem secara keseluruhan. Dalam jeda tersebut trafik terhenti dan kembali meningkat setelah beberapa saat. Semua serangan berhenti secara keseluruhan antara pukul 04:11:51 dan pukul 04:12:34.



Gambar 5.9 Trafik ketika menggunakan syn flood Spoofed IP (5 host)

Pada Gambar 5.9, memperlihatkan bahwa *syn flood spoofed* IP 5 *host* berhasil diatasi oleh IPS yang telah dibangun. Sama halnya dengan Gambar 5.8, bahwa terdapat jeda agar semua *host* dapat di-*drop*. Dalam jeda tersebut trafik terhenti dan kembali meningkat setelah beberapa saat dan berhasil dihentikan oleh sistem antara pukul 04:16:19 dan pukul 04:17:02.

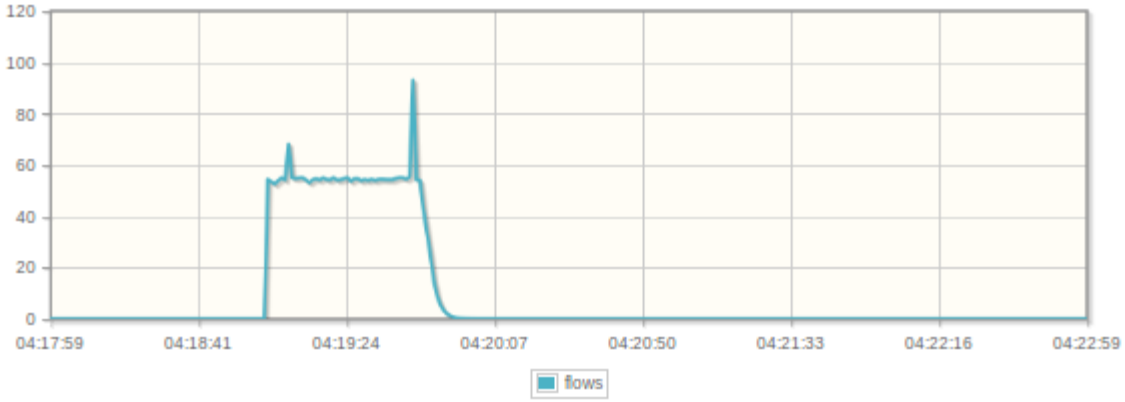
- 8 host

Tabel 5.4 Hasil uji dengan 8 host

Host	Keberhasilan drop host	
	Direct Attack	Spoofed IP
1	Berhasil	Berhasil
2	Berhasil	Berhasil
3	Berhasil	Berhasil
4	Berhasil	Berhasil
5	Berhasil	Berhasil
6	Berhasil	Berhasil
7	Berhasil	Berhasil
8	Berhasil	Berhasil

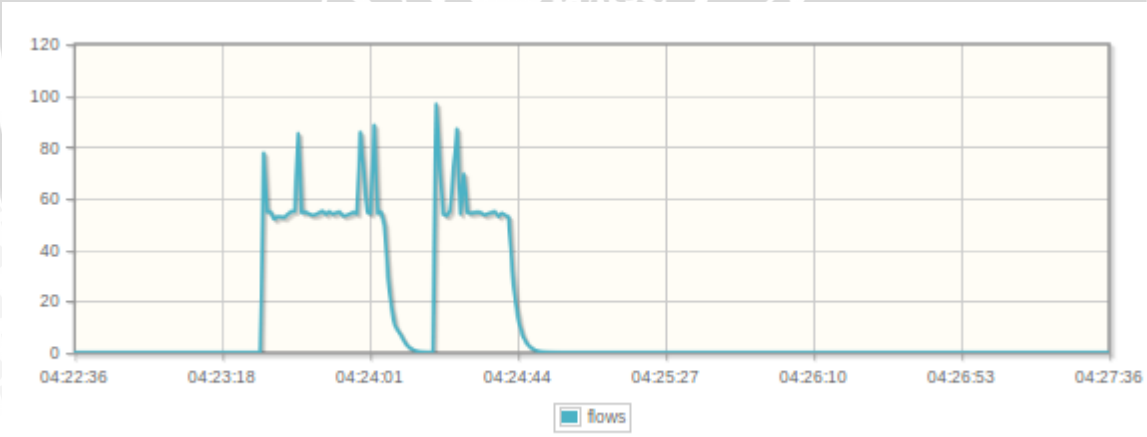
Tabel 5.4 merupakan data serangan yang dilakukan oleh 8 *host*. Seperti yang telah direncanakan pada scenario uji, *host* tersebut melakukan dua kali uji serangan yaitu *DoS syn flood direct attack* dan *DoS syn flood spoofed IP*. Pada uji serangan *DoS syn flood direct attack*, 8 *host*

penyerang berhasil di-*drop* dan pada uji serangan *DoS syn flood spoofed IP*, 8 *host* penyerang juga berhasil di-*drop*.



Gambar 5.10 Trafik ketika menggunakan syn flood direct attack (8 host)

Pada Gambar 5.10, memperlihatkan bahwa *syn flood direct attack* dengan 8 host berhasil diatasi oleh IPS yang telah dibangun. Dalam jangka waktu uji coba yang diberikan yaitu selama 5 menit, sistem berhasil melakukan *drop host* penyerang antara pukul 04:19:24 dan pukul 04:20:07.



Gambar 5.11 Trafik ketika menggunakan syn flood Spoofed IP (8 host)

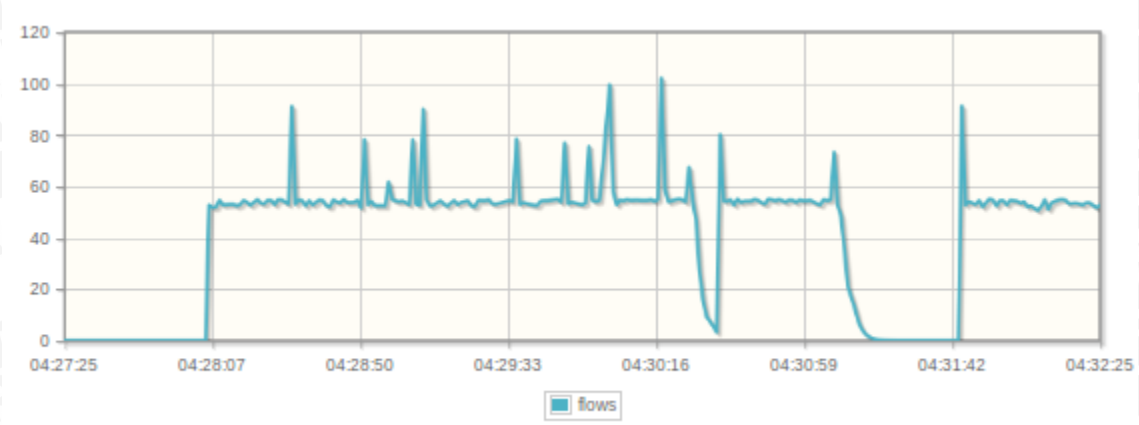
Pada Gambar 5.11, memperlihatkan bahwa *syn flood spoofed IP* dengan 8 *host* berhasil diatasi oleh IPS yang telah dibangun. Berbeda dengan sebelumnya pada Gambar 5.10, pada Gambar 5.11 menunjukan *host* tidak langsung dapat diatasi karena terdapat jeda setelah pukul 04:24:01. Serangan dapat di hentikan seara keseluruhan setelah pukul 04:24:44.

- 10 host

Tabel 5.5 Hasil uji dengan 10 host

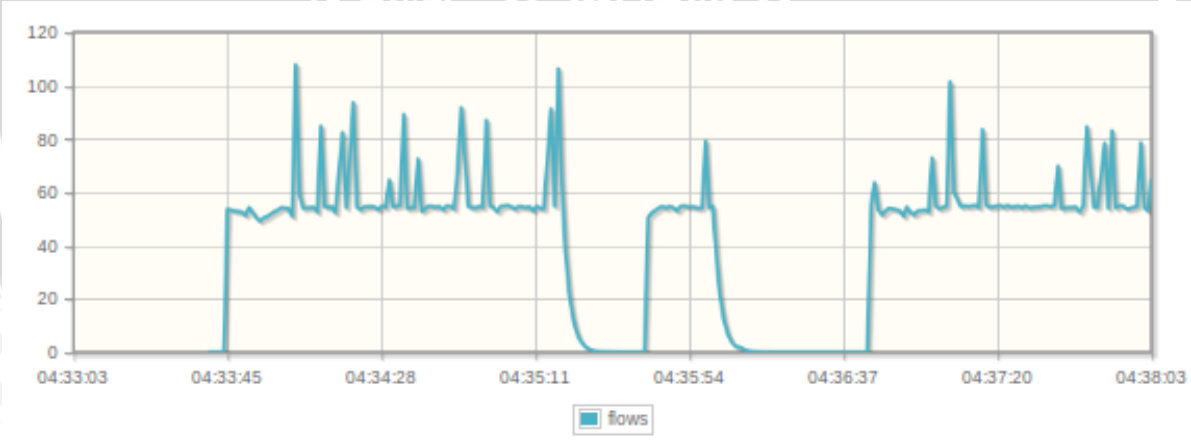
Host	Keberhasilan drop host	
	Direct Attack	Spoofed IP
1	Berhasil	Tidak
2	Berhasil	Berhasil
3	Tidak	Berhasil
4	Tidak	Berhasil
5	Berhasil	Berhasil
6	Tidak	Berhasil
7	Berhasil	Berhasil
8	Berhasil	Tidak
9	Berhasil	Berhasil
10	Tidak	Berhasil

Tabel 5.5 merupakan data serangan yang dilakukan oleh 10 *host*. Seperti yang telah direncanakan pada scenario uji, *host* tersebut melakukan dua kali uji serangan yaitu *DoS syn flood direct attack* dan *DoS syn flood spoofed IP*. Pada uji serangan *DoS syn flood direct attack*, terdapat 6 *host* penyerang berhasil di-*drop* dan 4 *host* penyerang yang gagal di-*drop*. Pada uji serangan *DoS syn flood spoofed IP*, 8 *host* penyerang tersebut berhasil di-*drop* dan terdapat 2 *host* penyerang gagal di-*drop*.



Gambar 5.12 Trafik ketika menggunakan syn flood direct attack (10 host)

Pada Gambar 5.12, memperlihatkan bahwa *syn flood direct attack* dengan 10 *host*, tidak semua *host* berhasil diatasi oleh IPS yang telah dibangun. Hal tersebut diperlihatkan dengan trafik *flow* yang tidak berhenti dalam waktu uji yang telah diberikan, yaitu selama 5 menit. dapat dilihat trafik *flow* tetap ada atau berlanjut saat pukul 04:32:25.



Gambar 5.13 Trafik ketika menggunakan syn flood Spoofed IP (10 host)

Pada Gambar 5.13, memperlihatkan bahwa *syn flood spoofed IP* dengan 10 *host* tidak semuanya berhasil diatasi oleh IPS yang telah dibangun. Hal tersebut diperlihatkan oleh trafik *flow* yang tidak berhenti setelah pukul 04:38:03.

b. Uji TCP normal

Tahap ini adalah pengujian dengan uji TCP biasa, dengan menggunakan 1 *host* yang mencoba beberapa kali percobaan dengan masing-masing jumlah koneksi TCP berbeda. Berikut hasilnya :

Tabel 5.6 Hasil uji TCP normal

Nomor	Jumlah koneksi TCP (dalam 2 detik)	Terdeteksi sebagai ancaman
1	1	Tidak
2	10	Tidak
3	11	Tidak
4	30	Tidak
5	31	Tidak
6	50	Ya
7	51	Ya
8	60	Ya
9	61	Ya

Pada Tabel 5.6 memperlihatkan data mengenai hasil uji TCP normal dengan sistem IPS yang sedang berjalan. Pada percobaan nomor 1 sampai 5, *host* tidak terdeteksi sebagai ancaman. Namun untuk percobaan nomor 6 sampai 9, *host* terdeteksi sebagai ancaman dimana *host* melakukan koneksi TCP dengan jumlah sama dengan atau lebih dari 50 koneksi dalam jangka waktu kurang dari 2 detik.

5.2.3 Pengujian Skalabilitas

Berikut ini adalah hasil uji skalabilitas dengan memperlihatkan *CPU usage* dan *memory usage*. Pengujian dilakukan dua kali berdasarkan jenis serangan, yaitu *syn flood direct attack* dan *syn flood* dengan *spoofed IP*.

a. CPU Usage

Tabel 5.7 Hasil uji CPU Usage

Nomor	Jumlah Host	CPU Usage (%)	
		Direct Attack	Spoofed IP
1	1	11.2	14.2
2	3	13.6	19.7
3	5	28.2	23.6
4	7	24.9	27.3
5	10	34.7	31.6

Pada Tabel 5.7 merupakan tabel data hasil uji mengenai *CPU usage*. Uji yang dilakukan adalah dengan melakukan serangan *DoS syn flood* secara berkala dan dengan jumlah *host* yang berbeda disetiap ujinya. Untuk mengetahui *CPU usage* disetiap ujinya, maka digunakan perintah "*top*" dan menyeleksi UID milik *pyretic* dengan perintah "*grep*" pada linux.

b. Memory Usage

Tabel 5.8 Hasil uji Memory Usage

Nomor	Jumlah Host	Memory Usage (%)	
		Direct Attack	Spoofed IP
1	1	0.9	0.9
2	3	1.5	1.5
3	5	1.5	1.5
4	7	1.5	1.4
5	10	1.5	1.5

Pada Tabel 5.8 merupakan tabel hasil uji mengenai *memory usage* dengan melakukan serangan *DoS syn flood* secara berkala dan dengan jumlah *host* penyerang berbeda disetiap ujinya. Untuk mengetahui *memory usage* di setiap ujinya, maka digunakan perintah "*top*" dan menyeleksi UID milik *pyretic* dengan perintah "*grep*" pada linux.

5.2.4 Pengujian Rule Pada IPS

Berikut ini adalah hasil dari pengujian *rule* yang diterapkan pada IPS yang telah dirancang. Pengujian dilakukan dua kali yaitu pengujian parameter IP dan parameter paket TCP SYN.

a. Parameter IP

Tabel 5.9 Hasil uji parameter IP

Nomor	Jumlah IP	Terdeteksi sebagai ancaman
1	1	Tidak
2	3	Tidak
3	5	Ya
4	7	Ya
5	10	Ya

Pada Tabel 5.9 dapat dilihat bahwa ketika jumlah IP yang digunakan oleh *host* penyerang sebanyak 5 IP dalam waktu kurang dari 2 detik, maka *host* tersebut akan terdeteksi sebagai penyerang. Begitu pula ketika *host* penyerang menggunakan IP sebanyak 7 dan 10 IP. Namun hal tersebut tidak terjadi ketika *host* penyerang hanya menggunakan IP sebanyak 1 IP atau 3 IP.

b. Parameter paket TCP SYN

- Uji jumlah paket TCP SYN

Tabel 5.10 Hasil uji parameter paket TCP

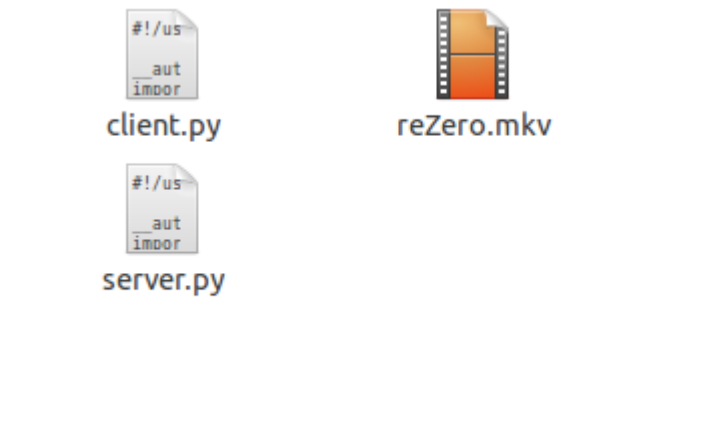
Nomor	Jumlah paket TCP	Terdeteksi sebagai ancaman
1	10	Tidak
2	30	Tidak
3	50	Ya
4	70	Ya
5	100	Ya

Pada Tabel 5.10 merupakan tabel hasil uji parameter paket TCP dan dapat dilihat bahwa ketika *host* penyerang mengirimkan paket TCP

sebanyak 10 atau 30, maka *host* penyerang tidak terdeteksi sebagai ancaman. Namun ketika *host* penyerang mengirimkan paket TCP sebanyak 50, 70 atau 100 kepada sebuah *host* korban, maka *host* penyerang akan terdeteksi sebagai ancaman.

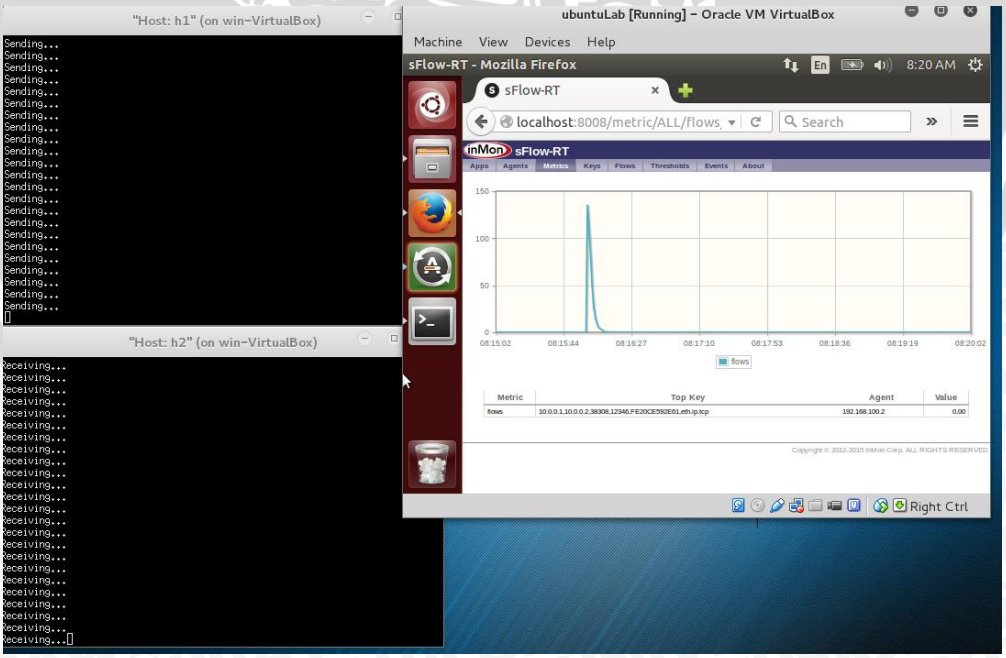
- Uji transfer file

Untuk melakukan uji ini penulis memiliki 3 file awal, yaitu *server.py*, *client*.py dan sebuah file yang dikirim dari *client.py* ke *server.py* bernama *reZero.mkv*. Perhatikan pada Gambar 5.14.



Gambar 5.14 File untuk uji transfer file

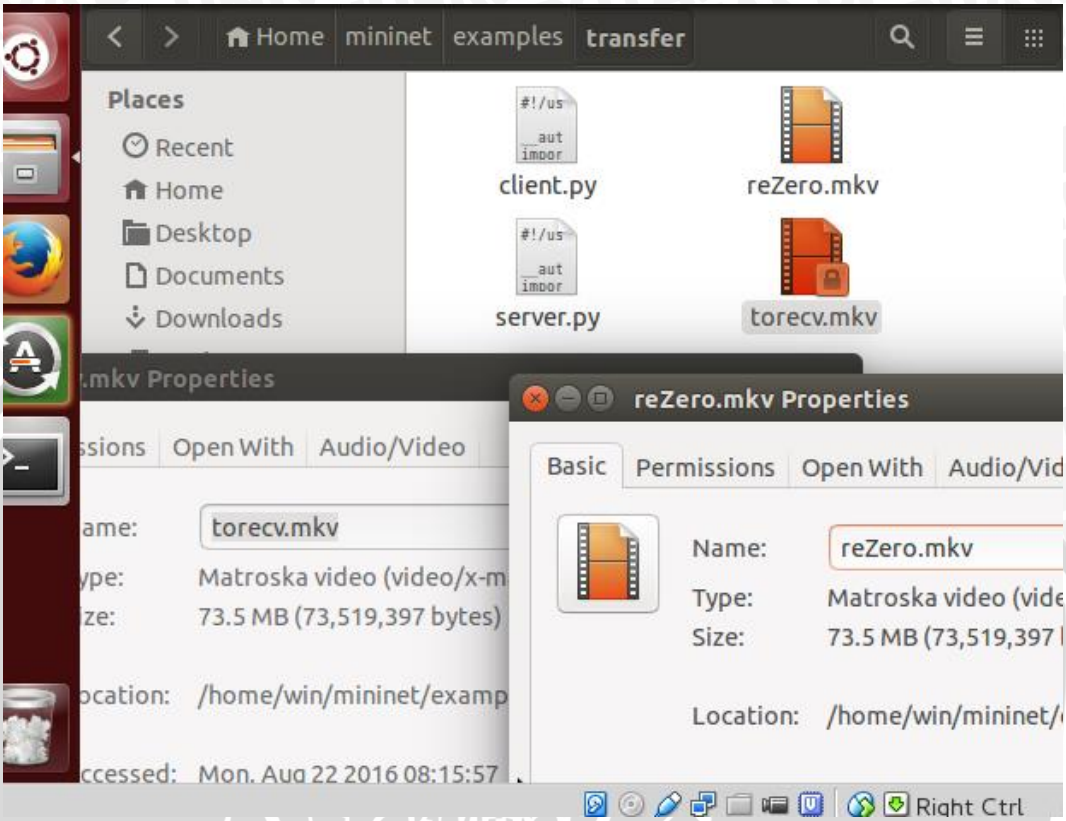
Server.py akan berjalan pada host 1 dan client.py berjalan pada host 2. Host 1 dan host 2 (mininet) akan berjalan menggunakan SSH. Perhatikan pada Gambar 5.15.



Gambar 5.15 Mengirim file dari host1 ke host2

Dapat dilihat pada Gambar 5.15, trafik flow hanya tinggi di satu waktu yaitu antara pukul 08:15:44 dan pukul 08:16:27. Hal tersebut menunjukan paket

TCP SYN yang dikirimkan oleh host 1. Dengan begitu, mengirimkan file atau melakukan pertukaran data yang cukup besar tidak dianggap sebagai SYN Flood oleh sistem.



Gambar 5.16 Bukti pengiriman file berhasil dilakukan

Paga Gambar 5.16 merupakan bukti bahwa file sudah dikirim. Dapat diperhatikan pada ukuran file yang sama besar.

5.3 Analisis Hasil

Pada bagian ini penulis akan menganalisa hasil berdasarkan data-data pengujian yang telah dilakukan sebelumnya. Ada dua sesi penjelasan analisis hasil, yang pertama adalah analisis hasil mengenai pengujian dengan melakukan serangan *DoS syn flood* dan yang kedua adalah analisis mengenai pengujian dengan melakukan koneksi TCP biasa.

5.3.1 Analisis Hasil Pengujian Dengan Syn Flood Attack

Adapun data-data mengenai pengujian dengan *syn flood attack*, sebelumnya telah dirangkum dalam tabel. Dari tabel-tabel tersebut didapatlah sebuah data berupa diagram hasil uji dengan *syn flood attack.* Dari data tersebut akan dirangkum lagi pada Tabel 5.9 yaitu mengenai hasil uji dengan *syn flood attack* sebagai berikut:

Tabel 5.11 Hasil uji dengan syn flood attack

Nomor	Jumlah penyerang	Host yang di-drop	
		Direct attack	Spoffed IP
1	1	1	1
2	3	3	3
3	5	5	5
4	8	8	8
5	10	6	8

Pada Tabel 5.9 dapat dilihat bahwa tidak semua *host* yang melakukan serangan dapat dicegah. Hal tersebut dapat dilihat ketika terdapat 10 *host* yang melakukan serangan *DoS syn flood* secara bersamaan. Terdapat 6 *host* yang dapat di-*drop*, yang artinya terdapat 4 *host* juga yang gagal teratasi oleh sistem pada *Direct attack*. Sedangkan pada *Spoffed* IP terdapat 8 *host* yang berhasil di-*drop* dan 2 gagal di-*drop*.

Dengan demikian dapat digambarkan dalam diagram hasil uji dengan *syn flood attack* sebagai berikut :

Gambar 5.17 Diagram hasil uji dengan syn flood attack

Dengan data-data yang telah diolah seperti diatas, dapat pula dianalisa seberapa efektifkah sistem yang telah dibuat. Untuk mencari keefektifan sistem dapat dicari dengan menghitungnya dengan persamaan sederhana berikut :

$$K = \frac{x}{y} 100\% \quad (5.1)$$

Dimana :

K = keefektifan sistem

x = Jumlah *host* yang berhasil diblock

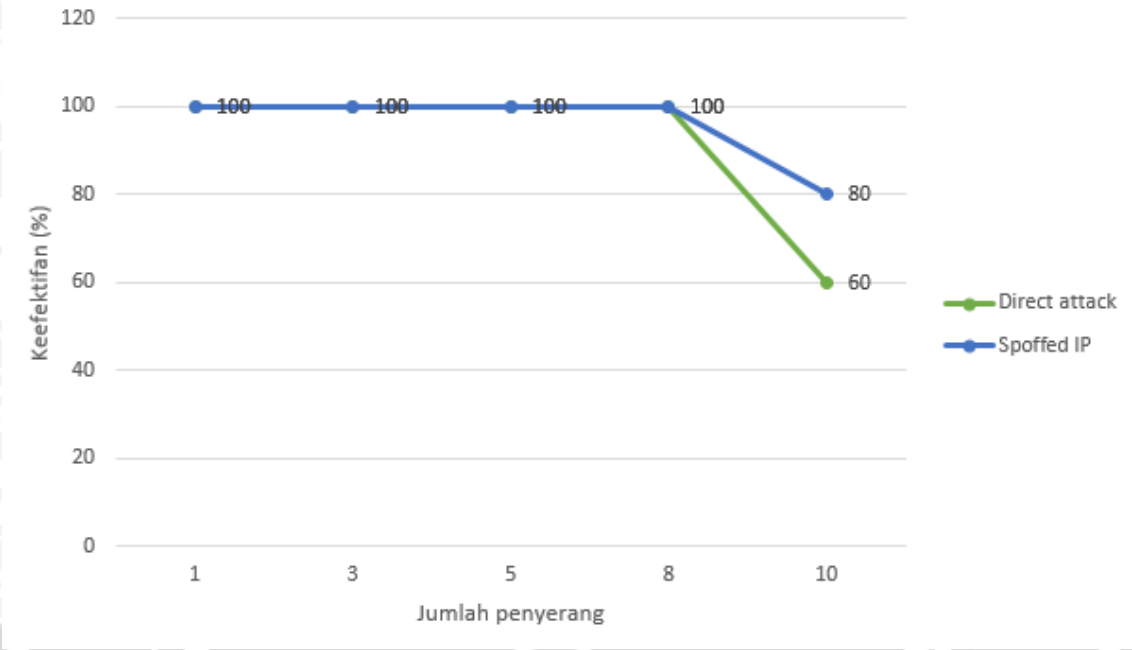
y = Jumlah *host* yang melakukan serangan

Dengan persamaan 5.1, didapatlah tabel dan diagram keefektifan sebagai berikut:

Tabel 5.12 Keefektifan sistem

Nomor	Jumlah penyerang	Keefektifan (%)		Rata – rata (%)
		Direct attack	Spoffed IP	
1	1	100	100	100
2	3	100	100	100
3	5	100	100	100
4	8	100	100	100
5	10	60	80	70
Rata – rata (%)		92	96	**94**

Pada Tabel 5.10 menunjukan keefektifan yang diperoleh dengan menerapkan sistem *IPS* yang telah dirancang mencapai 94%. Keefektifan tidak mencapai 100% dikarenakan tidak semua *host* penyerang dapat teratasi oleh sistem.



Gambar 5.18 Diagram keefektifan sistem

Adapun Gambar 5.15 menunjukan diagram keefektifan sistem yang didapat berdasarkan Tabel 5.10. Dapat dilihat keefektifan sistem akan semakin menurun pada kasus serangan *DoS syn flood direct attack* maupun *spoffed IP*.

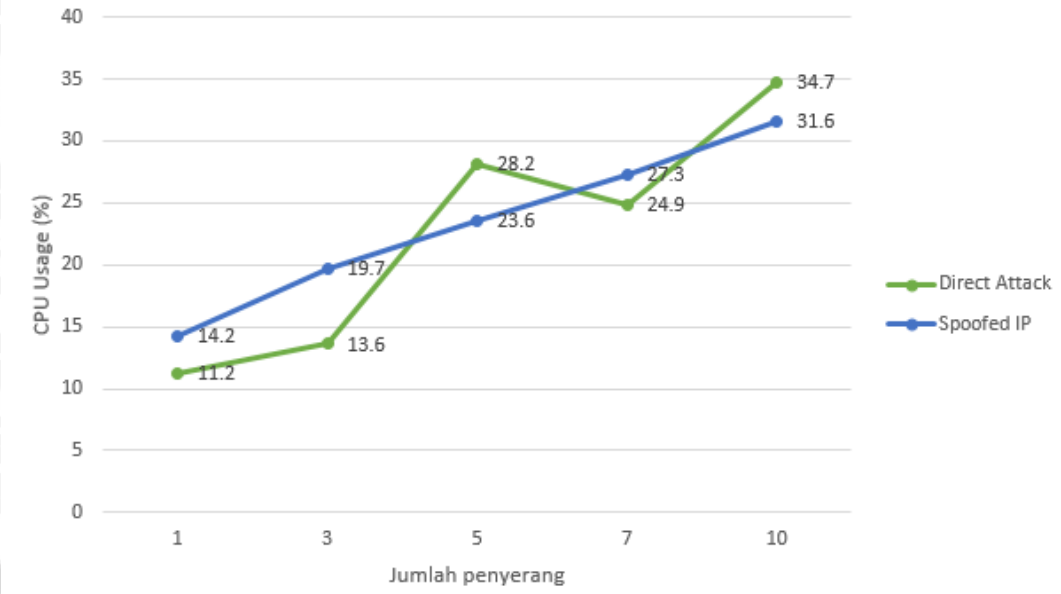
5.3.2 Analisis Hasil Pengujian Dengan TCP Normal

Pada pengujian dengan melakukan koneksi TCP biasa terdapat masalah ketika jumlah koneksi TCP sama atau lebih dari 50. Hal ini dikarenakan pada sistem IPS yang dibangun melakukan analisa setiap 2 detik. Dimana dalam 2 detik tersebut, setiap *host* hanya diperbolehkan melakukan koneksi TCP sebanyak 50.

Penulis belum mengetahui penyebabnya, namun jika diperhatikan pada teori *three way handshake*, maka hal ini wajar karena dapat dianalisa bahwa *host* yang melakukan koneksi TCP akan mengirim paket pada *server* berupa *syn* dan *ack*. Sehingga dalam 1 kali melakukan koneksi TCP, maka host tersebut mengirim 1 paket TCP syn dan terdeteksi sebagai syn flood ketika mengirimnya 50 kali. Itulah mengapa dengan melakukan koneksi TCP sebanyak 50 sebenarnya mengirim 50 paket *TCP SYN* dan dideteksi sebagai ancaman.

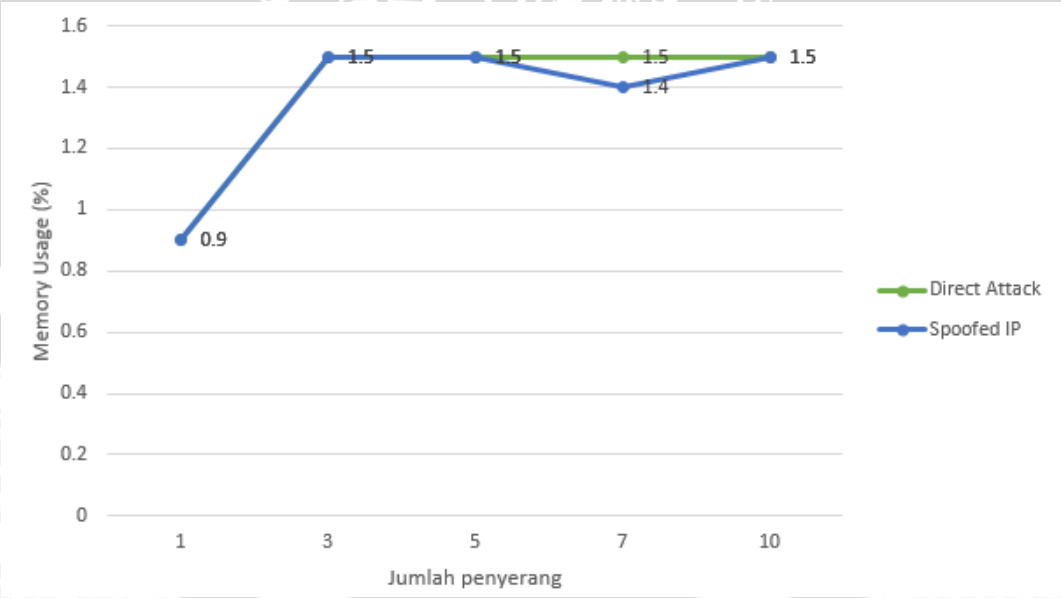
5.3.3 Analisis Hasil Pengujian Skalabilitas

Adapun data-data mengenai pengujian skalabilitas yang telah dilakukan sebelumnya telahdirangkum dalam Tabel 5.7 dan Tabel 5.8. Dari tabel-tabel tersebut dapat digambarkan dalam grafik sebagai berikut :



Gambar 5.19 Diagram uji CPU usage

Dari Gambar 5.16 dapat dilihat bahwa *CPU usage* akan semakin meningkat seiring meningkatnya jumlah *host* yang melakukan serangan. Namun terdapat penurunan *CPU usage* ketika ada 7 *host* penyerang. Namun secara keseluruhan, diagram uji *CPU usage* cenderung naik.



Gambar 5.20 Diagram uji memory usage

Pada Gambar 5.17 memperlihatkan bahwa terdapat peningkatan *memory usage* antara jumlah penyerang 1 *host* dan 3 *host*. Namun setelah itu, *memory usage* cenderung tetap.

BAB 6 PENUTUP

6.1 Kesimpulan

Berdasarkan penelitian yang telah dilakukan yaitu membangun IPS pada SDN untuk mencegah atau mengatasi *DoS Syn Flood,* dapat disimpulkan beberapa hal yaitu :

1. Sistem yang dibangun, berjalan seperti yang diharapkan. Sistem dapat mengatasi serangan yang dilakukan beberapa *host* sekaligus dengan keakuratan rata-rata mencapai 94%.
2. Berdasarkan hasil uji yang telah dilakukan dan dianalisa, sistem masih memiliki kekurangan. Jika dilihat dari grafik (ambil contoh Gambar 5.13) pada saat pengujian, grafik menunjukan flow yang *naik turun* atau tidak stabil. Dalam kasus itu semua host melakukan serangan tanpa ada yang melakukan koneksi TCP biasa. Namun *flow* yang turun setelah *drop host* berhasil dilakukan, hal ini menunjukan bahwa jaringan tidak dapat diakses dan dapat diakses kembali setelah beberapa detik kemudian. Ini menunjukan bahwa untuk melakukan *update* pada *policy* untuk tujuan *drop host,* dapat berdampak pada jaringan di beberapa detik kedepan, yaitu jaringan menjadi *down* untuk sementara.
3. Untuk mendeteksi *DoS Syn Flood* dapat dilakukan dengan mengambil data jumlah TCP, IP dan *mac address* lalu menganalisanya berdasarkan waktu. Seperti yang telah dijelaskan, penentuannya adalah dengan memperhitungkan 2 kemungkinan. Kemungkinan pertama, jika satu *host* mengirim jumlah paket *TCP SYN* lebih besar dari 50, maka akan dianggap ancaman. Kemungkinan kedua, yaitu jika satu *host* menggunakan lebih dari 5 IP *Address*, maka juga akan dianggap sebagai ancaman.
4. Untuk mencegah *Dos Syn Flood* yang telah terdeteksi bisa dilakukan dengan mengirimkan *request drop host* dari modul deteksi kepada modul *controller*. Pada modul *controller* akan melakukan *drop host* sesuai *format* yang disediakan oleh *pyretic*.
5. Berdasarkan analisa hasil, kefektifan sistem yang dibangun akan menurun berdasarkan jumlah *host* yang melakukan serangan. Dapat dilihat pada Gambar 5.15, penurunan keefektifan sebesar 6% terjadi ketika jumlah penyerang sebanyak 10 *host* dengan serangan *DoS Syn Flood Direct Attack*.
6. Ketika melakukan koneksi TCP biasa juga terdapat masalah ketika jumlah TCP biasa diatas 50. Namun ini dapat dianggap wajar, karena setiap *host* memang hampir tidak mungkin melakukan koneksi TCP sebanyak 50 koneksi kepada sebuah server dalam waktu singkat pada penggunaan biasa.
7. Terkait dengan skalabilitas pada Gambar 5.16 dan Gambar 5.17, sistem yang dibangun mengalami kenaikan pada CPU *usage* walaupun terjadi

penurunan saat terdapat 7 *host* penyerang dengan serangan *DoS Syn Flood Direct Attack*. Namun secara keseluruhan yaitu dari skala 1 *host* penyerang sampai 10 *host* peyerang, CPU *usage* oleh sistem cenderung mengalami kenaikan yaitu 11,2% sampai 34,7% untuk serangan *DoS Syn Flood Direct Attack* dan 14,2% sampai 31,6% untuk serangan *DoS Syn Flood Spoofed IP*. Dengan ini dapat disimpulkan bahwa semakin banyak penyerang, maka CPU *usage* oleh sistem IPS pada *controller* akan semakin tinggi. Sedangkan untuk *memory usage* oleh sistem IPS pada *controller* cenderung tetap, kenaikan hanya terjadi ketika terdapat 3 *host* penyerang dengan serangan *Dos Syn Flood Direct Attack* maupun *Spoofed IP* dan terjadi penurunan *memory usage* ketika terdapat 7 *host* penyerang dengan serangan *DoS Syn Flood spoofed IP*. Namun, kenaikan dan penurunan *memory usage* tersebut tidak terlalu signifikan yakni tidak lebih dari 1%.

8. Untuk membangun IPS pada SDN, dapat dilakukan dengan menerapkannya pada *controller*. Dalam kasus ini, instalasi sistem IPS dilakukan pada *controller* sehingga *controller* memiliki *behavior* untuk mendeteksi ancaman dan meminimalisir ancaman tersebut. Cara kerja IPS dalam kasus DoS ini adalah dengan menganalisa trafik pada sebuah *switch* lalu menerapkan *rule* yang telah dibuat untuk memutuskan langkah selanjutnya.

6.2 Saran

Berdasarkan kesimpulan yang ada pada penelitian ini, maka penulis mencoba membuat beberapa saran mengenai sistem IPS pada SDN seperti berikut :

1. Untuk penerapan IPS pada SDN, sebaiknya mencoba *controller* lain terlebih dahulu untuk mencari *controller* mana yang lebih *robust*.
2. Penelitian ini masih belum detail karena hanya melihat keberhasilan sistem dari sisi bekerja atau tidaknya sistem dalam mengatasi DoS. Sehingga kedepannya dapat diteliti lebih lanjut seperti availabilitas, skalabilitas dan lainnya.
3. *Controller* yang diteliti hanyalah *pyretic*, sehingga kedepannya penelitian serupa dapat menggunakan *controller* lain. Yang artinya implementasinya pun berbeda.
4. Peningkatan efisiensi pada penelitian selanjutnya sangat diharapkan, karena pada penelitian ini masih memiliki kekurangan yaitu keefektifan akan turun drastis ketika *host* penyerang meningkat.
5. Untuk penelitian lebih lanjut bisa dilakukan penelitian mengenai IPS untuk mengatasi ancaman selain DoS pada arsitektur jaringan SDN seperti *DNS Spoofing*, *ARP spoofing* atau yang lainnya.

DAFTAR PUSTAKA

[1] citrix, 2014. SDN 101: An Introduction to Software Defined Networking.

[2] Open Networking Foundation, 2012. Software-Defined Networking: The New Norm for Networks.

[3] Wang, H., n.d. *Detecting SYN Flooding Attacks.* s.l., EECS Department, The University of Michigan.

[4] Alexander, J., 2009. Intrusion Detection and Prevention Systems (IDS/IPS).

[5] Joshua Reich, C. M. N. F. J. R. a. D. W., n.d. Modular SDN Programming with Pyretic.

[6] Kreutz, D., n.d. Software-Defined Networking: A Comprehensive Survey.

[7] Shaw, B., 2014. *Security-Centric SDN (Security-Centric SDN),* Calabasas: ixia.

[8] Dover, J. M., n.d. *A denial of service attack against the Open Floodlight SDN controller,* s.l.: Dover Networks LLC.

[9] Liu, P., n.d. Denial of Service Attacks.

[10] Lim, S., 2014. *A SDN-Oriented DDoS Blocking Scheme for Botnet-Based Attacks,* Seoul, Korea: IEEE.

[11] Sahay, R., n.d. *Towards Autonomic DDoS Mitigation using Software Defined Networking,* France: Institut Mines-Tel ´ ecom, T ´ el ´ ecom SudParis.

[12] Muhammad Aamir, M. A. (2013). *DDoS Attack and Defense: Review of Some Traditional and Current Techniques.* Karachi, Pakistan: Interdisciplinary Information Sciences [86]. doi:10.4036/iis.2013.173.

[13] Syed Taha Ali, V. S. (n.d.). *A Survey of Securing Networks using Sowftware Defined Networking.* IEEE TRANSACTIONS ON RELIABILITY.

[14] Zachariah, Bobbin. “What Is SYN Flood Attack? Detection & Prevention In Linux”. 19 Agustus 2016. http://linoxide.com/firewall/snapshot-syn-flood-attack/

[15] Rana, D. S., 2012. *A Study and Detection of TCP SYN Flood Attacks with IP spoofing and its Mitigations,* Dehradun, India: IJCTA.