

**DETEKSI DAN MITIGASI SERANGAN *DISTRIBUTED DENIAL OF SERVICE (DDOS)* PADA *SOFTWARE DEFINED NETWORK (SDN)* MENGGUNAKAN *MULTILAYER PERCEPTRON (MLP)***

**SKRIPSI**

Disusun sebagai salah satu syarat untuk memperoleh Sarjana Teknik (S.T.)



**Disusun oleh:**

**IVAN MUNANDAR PURNAMA**

**NPM. 3332190035**

**JURUSAN TEKNIK ELEKTRO  
FAKULTAS TEKNIK  
UNIVERSITAS SULTAN AGENG TIRTAYASA  
2023**

## LEMBAR PERNYATAAN KEASLIAN SKRIPSI

Dengan ini saya menyatakan bahwa:

Judul : Deteksi dan Mitigasi Serangan *Distributed Denial of Service* (DDoS) pada *Software Defined Network* (SDN) Menggunakan *Multilayer Perceptron* (MLP)

Nama Mahasiswa : Ivan Munandar Purnama

NPM : 3332190035

Fakultas/Jurusan : Teknik/Teknik Elektro

Saya dengan tegas dan jujur menyatakan bahwa skripsi ini adalah hasil karya saya sendiri. Tidak ada bagian di dalamnya yang merupakan plagiat dari karya orang lain dan saya tidak melakukan penjiplakan atau pengutipan dengan cara yang tidak sesuai dengan etika keilmuan yang berlaku. Saya memiliki kesadaran penuh tentang pentingnya menghormati hak cipta dan keaslian karya orang lain. Oleh karena itu, saya berkomitmen untuk tidak melakukan tindakan yang dapat merugikan integritas ilmiah dan keaslian karya saya sendiri. Saya bertanggung jawab atas keseluruhan isi skripsi ini serta keabsahan dan keaslian materi yang ada di dalamnya. Saya juga menyadari bahwa jika kemudian ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya ini, atau jika ada klaim dari pihak lain terhadap keaslian karya saya ini, saya siap sepenuhnya menanggung resiko dan sanksi yang dijatuhkan kepada saya.

Cilegon, 22 Juli 2024



Ivan Munandar Purnama

NIM.3332190035

## LEMBAR PENGESAHAN

Dengan ini ditetapkan bahwa Skripsi berikut:

Judul : Deteksi dan Mitigasi Serangan *Distributed Denial of Service* (DDoS) pada *Software Defined Network* (SDN) Menggunakan *Multilayer Perceptron* (MLP)  
Nama Mahasiswa : Ivan Munandar Purnama  
NPM : 3332190035  
Fakultas/Jurusan : Teknik/Teknik Elektro

Telah di uji dan dipertahankan pada tanggal 15 Juli 2024 melewati Sidang Skripsi di Fakultas Teknik Universitas Sultan Ageng Tirtayasa Cilegon dan dinyatakan LULUS / TIDAK LULUS

### Dewan Penguji

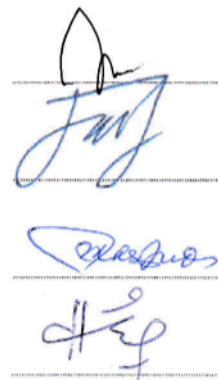
### Tanda Tangan

Pembimbing I : Rian Fahrizal, ST., MEng.

Pembimbing II : Fadil Muhammad, S.T., M.T.

Penguji I : Masjudin, S.T., M.Eng.

Penguji II : Dina Estining Tyas Lufianawati, S.T., M.T.



Mengetahui,

Ketua Jurusan Teknik Elektro



Dr. Eng. Rocky Alfanz, S.T., M.Sc.

NIP.198103282010121001

## PRAKATA

Puji syukur penulis panjatkan ke hadirat Allah SWT, karena dengan rahmat dan hidayah-Nya penulis dapat menyelesaikan penulisan skripsi ini dengan judul "Deteksi dan Mitigasi Serangan DDoS pada Lalu Lintas Jaringan *Software Defined Network* (SDN) Menggunakan *Multilayer Perceptron* (MLP)". Skripsi ini disusun sebagai salah satu syarat untuk memperoleh gelar Sarjana Teknik Elektro dari Universitas Sultan Ageng Tirtayasa.

Penulis mengucapkan terima kasih yang sebesar-besarnya kepada:

1. Orang tua tercinta, yang selalu memberikan dukungan, doa, dan kasih sayang tanpa henti. Terima kasih atas segala pengorbanan dan motivasi yang diberikan. Tanpa dukungan mereka, penulis tidak akan mampu menyelesaikan skripsi ini dengan baik.
2. Dr. Eng. Rocky Alfanz, S.T., M.Sc., selaku Ketua Jurusan Teknik Elektro, yang telah memberikan kesempatan dalam menjalankan penelitian ini. Bimbingan dari beliau sangat membantu penulis dalam memahami dan menyelesaikan masalah-masalah yang muncul selama penelitian.
3. Fadil Muhammad, S.T., M.T., selaku Pembimbing Akademik serta Pembimbing 2 skripsi, yang telah memberikan arahan, saran, dan motivasi yang sangat berarti dalam penyelesaian skripsi ini. Terima kasih atas kesabaran dan perhatian yang diberikan. Bimbingan dari beliau menginspirasi penulis untuk terus berkembang dan menghasilkan penelitian yang berkualitas.
4. Rian Fahrizal, S.T., M.Eng., selaku Pembimbing 1 skripsi, yang telah memberikan bimbingan, ilmu, dan pengalaman berharga dalam penulisan skripsi ini. Terima kasih atas bimbingan yang mendalam dan dukungan yang diberikan. Beliau telah membantu penulis dalam memahami konsep-konsep yang kompleks dan memberikan panduan yang jelas untuk mencapai tujuan penelitian.
5. Kepada teman-teman yang memberikan dukungan moral, semangat, dan persahabatan sepanjang perjalanan penulisan skripsi ini, penulis mengucapkan terima kasih yang sebesar-besarnya. Dukungan dan semangat dari teman-

teman sangat berarti bagi penulis dalam menghadapi tantangan dan melewati masa-masa sulit selama penulisan skripsi ini. Persahabatan yang terjalin juga memberikan motivasi dan kebahagiaan tersendiri dalam perjalanan ini.

Penulis juga ingin menyampaikan terima kasih kepada semua pihak yang telah memberikan kontribusi, bantuan, dan dukungan dalam penyelesaian skripsi ini. Semoga skripsi ini dapat bermanfaat bagi pengembangan ilmu pengetahuan di bidang Teknik Elektro dan menjadi sumbangan positif bagi perkembangan dunia akademik. Akhir kata, penulis menyadari bahwa skripsi ini masih jauh dari kesempurnaan. Oleh karena itu, penulis menerima dengan tangan terbuka setiap saran, masukan, dan kritik membangun untuk peningkatan kualitas penelitian di masa yang akan datang.

Cilegon, 22 Juli 2024



Ivan Munandar Purnama

NIM.3332190035

# ABSTRAK

Ivan Munandar Purnama

Teknik Elektro

Deteksi dan Mitigasi Serangan *Distributed Denial of Service* (DDoS) pada *Software Defined Network* (SDN) Menggunakan *Multilayer Perceptron* (MLP)

Serangan *Distributed Denial of Service* (DDoS) pada *Software Defined Network* (SDN) telah menjadi ancaman yang semakin meningkat, mengakibatkan gangguan serius pada ketersediaan layanan dan keandalan infrastruktur. Saat ini, teknik deteksi serangan DDoS yang paling umum digunakan pada jaringan tradisional maupun SDN adalah metode deteksi berbasis statistik, namun metode ini memiliki kelemahan dalam mendeteksi serangan yang baru atau belum pernah terjadi sebelumnya. Hal tersebut dapat diatasi dengan menggunakan metode *deep learning* yang mampu untuk menangani data bersifat non-linier, heterogen, dan berdimensi tinggi, yang sering ditemukan pada data lalu lintas jaringan. Penelitian ini mengembangkan sistem deteksi dan mitigasi serangan DDoS pada lalu lintas jaringan SDN menggunakan algoritma *Multilayer Perceptron* (MLP). Pengujian yang dilakukan menunjukkan bahwa sistem deteksi dan mitigasi serangan DDoS yang dikembangkan dalam penelitian ini memiliki tingkat akurasi sebesar 71,17%. Sistem ini terbukti efektif dalam mendeteksi serangan DDoS. Analisis performa jaringan pasca-mitigasi menunjukkan peningkatan performa yang signifikan dibandingkan dengan kondisi sebelum mitigasi. Sebelumnya, *bottleneck* teridentifikasi pada jaringan. Namun, setelah dilakukan mitigasi, jaringan dapat kembali berfungsi secara optimal.

Kata kunci: DDoS, *Deep Learning*, SDN, MLP, deteksi, mitigasi.

## **ABSTRACT**

Ivan Munandar Purnama

Electrical Engineering

### Detection and Mitigation of Distributed Denial of Service (DDoS) Attacks on Software Defined Network (SDN) Using Multilayer Perceptron (MLP)

Distributed Denial of Service (DDoS) attacks on Software Defined Networks (SDNs) have become a growing threat, resulting in serious disruptions to service availability and infrastructure reliability. Currently, the most common DDoS attack detection technique used on both traditional and SDN networks is the statistical-based detection method. However, this method has the limitation of detecting new or unprecedented attacks. This can be overcome by using deep learning methods that are able to handle non-linear, heterogeneous, and high-dimensional data, characteristic of network traffic data. This research proposed a DDoS attack detection and mitigation system on SDN network traffic using the Multilayer Perceptron (MLP) algorithm. The tests show that the DDoS attack detection and mitigation system developed in this study has an accuracy of 71.17%. This system has demonstrated effectiveness in detecting DDoS attacks. Post-mitigation network performance analysis shows a significant improvement in performance compared to pre-mitigation conditions. Previously, bottlenecks were identified in the network. However, after mitigation, the network can regain optimal functionality.

Keywords: DDoS, Deep Learning, SDN, MLP, detection, mitigation.

## DAFTAR ISI

<b>HALAMAN JUDUL</b> .....	i
<b>LEMBAR PERNYATAAN KEASLIAN SKRIPSI</b> .....	ii
<b>LEMBAR PENGESAHAN</b> .....	iii
<b>PRAKATA</b> .....	iv
<b>ABSTRAK</b> .....	vi
<b>ABSTRACT</b> .....	vii
<b>DAFTAR ISI</b> .....	viii
<b>DAFTAR TABEL</b> .....	xiv
<b>BAB I PENDAHULUAN</b> .....	1
1.1. Latar Belakang .....	1
1.2. Rumusan Masalah.....	3
1.3. Tujuan penelitian .....	3
1.4. Manfaat Penelitian .....	4
1.5. Batasan Masalah .....	4
1.6. Sistematika Penulisan .....	5
<b>BAB II TINJAUAN PUSTAKA</b> .....	6
2.1. Artificial Intelligence .....	6
2.2. <i>Machine Learning</i> .....	6
2.3. <i>Deep Learning</i> .....	7
2.4. Matriks Evaluasi pada <i>artificial intelligence</i> Model.....	7
2.4.1 Matriks Konfusi.....	8
2.4.2 Akurasi .....	8
2.4.3 Presisi .....	8
2.4.4 <i>Recall</i> .....	9
2.4.5 <i>F1 Score</i> .....	9
2.4.6 <i>Loss Function</i> .....	9



2.4.7	<i>Receiver Operating Characteristic Curve (ROC Curve)</i> .....	9
2.5.	<i>Multilayer Perceptron (MLP)</i> .....	9
2.6.	Jaringan Komputer.....	10
2.7.	<i>Software-Defined Networks</i> .....	12
2.7.2	Mininet .....	13
2.7.3	<i>Ryu controller</i> .....	14
2.8.	<i>Quality of Service</i> .....	14
2.8.2	<i>Throughput</i> .....	15
2.8.3	<i>Delay</i> .....	15
2.8.4	<i>Jitter</i> .....	15
2.8.5	<i>Packet loss</i> .....	16
2.9.	<i>Cyber Security</i> .....	16
2.10.	Serangan DDoS.....	17
2.11.	Python .....	20
2.12.	Kajian Pustaka .....	20
<b>BAB III</b>	<b>METODOLOGI PENELITIAN</b> .....	<b>23</b>
3.1.	Alur Penelitian .....	23
3.2.	Komponen Penelitian.....	24
3.2.1	Python.....	24
3.2.2	Sistem Operasi Ubuntu 22.04.....	24
3.2.3	<i>Traffict Generator</i> .....	24
3.2.4	Wireshark .....	24
3.2.5	<i>S-flow RT</i> .....	24
3.2.6	Jupyter Notebook.....	25
3.2.7	Mininet .....	25
3.2.8	<i>Ryu controller</i> .....	25

3.2.9	Hardware .....	25
3.3.	Metode Penelitian .....	26
3.3.1	Studi Literatur.....	26
3.3.2	Perancangan Topologi <i>Software Defined Network</i> (SDN).....	27
3.3.3	Akuisisi Data .....	30
3.3.4	Pra-pemrosesan data.....	32
3.3.5	Perancangan Algoritma .....	33
3.3.6	Training <i>Dataset</i> .....	34
3.3.7	Uji Keandalan Model .....	34
3.3.8	Analisis <i>Quality of Service</i> (QoS) .....	35
3.4.	Perancangan Sistem .....	36
<b>BAB IV</b>	<b>HASIL DAN PEMBAHASAN</b> .....	<b>37</b>
4.1.	Analisis Jaringan.....	37
4.2.	Analisis Hasil Deteksi dan Mitigasi.....	41
4.2.1	Lalu lintas Normal.....	41
4.2.2	ICMP <i>flood</i> .....	43
4.2.3	UDP <i>flood</i> .....	45
4.2.4	TCP SYN <i>flood</i> .....	46
4.2.5	Rata -Rata Efektivitas Deteksi.....	49
4.3.	Analisis Performa Jaringan .....	50
4.3.1	<i>Jitter</i> .....	50
4.3.2	<i>Delay</i> .....	52
4.3.3	<i>Throughput</i> .....	54
4.3.4	<i>Packet Loss</i> .....	55
4.3.5	<i>Resource Utilization</i> .....	57
4.4.	Hasil dan Analisis Pembuatan <i>Dataset</i> .....	60

4.5. Evaluasi Model MLP .....	62
4.6. Efektivitas Pelatihan Model .....	67
<b>BAB V PENUTUP</b> .....	<b>70</b>
5.1. Kesimpulan .....	70
5.2. Saran. ....	71
<b>LAMPIRAN A HASIL SIMULASI PERCOBAAN</b> .....	<b>A-1</b>
<b>LAMPIRAN B KODE PROGRAM</b> .....	<b>B-1</b>

## DAFTAR GAMBAR

Gambar 2.1	Arsitektur jaringan berdasarkan model OSI .....	11
Gambar 2.2	Jenis serangan siber .....	12
Gambar 2.3	Jenis serangan siber .....	16
Gambar 3.1	Diagram alir penelitian.....	23
Gambar 3.2	Topologi jaringan SDN .....	28
Gambar 3.3	Blok diagram pembuatan <i>dataset</i> .....	30
Gambar 3.4	Alur pra-pemrosesan data.....	32
Gambar 3.5	Alur proses training data .....	34
Gambar 3.6	Perancangan sistem .....	36
Gambar 4.1	Topologi jaringan linear .....	38
Gambar 4.2	Koneksi antara <i>host</i> dan <i>switch</i> .....	39
Gambar 4.3	Aktivitas <i>port</i> saat pertama kali terhubung .....	40
Gambar 4.4	Hasil pengujian <i>pingall</i> pada terminal mininet .....	41
Gambar 4.5	Hasil monitoring lalu lintas normal.....	42
Gambar 4.6	Hasil deteksi lalu lintas normal .....	42
Gambar 4.7	Hasil monitoring lalu lintas ICMP <i>flood</i> .....	43
Gambar 4.8	Hasil deteksi dan mitigasi lalu lintas ICMP <i>flood</i> .....	44
Gambar 4.9	Hasil monitoring lalu lintas UDP <i>flood</i> .....	45
Gambar 4.10	Hasil deteksi dan mitigasi lalu lintas UDP <i>flood</i> .....	46
Gambar 4.11	Hasil monitoring lalu lintas TCP SYN.....	47
Gambar 4.12	Lalu lintas TCP SYN <i>flood</i> pada Wireshark .....	47
Gambar 4.13	Hasil deteksi dan mitigasi lalu lintas TCP SYN <i>flood</i> .....	48
Gambar 4.14	<i>Jitter</i> pada jaringan lalu lintas normal.....	50
Gambar 4.15	Hasil <i>dataset</i> berdasarkan ekstraksi jaringan SDN .....	60
Gambar 4.16	Distribusi jumlah paket lalu lintas dalam <i>dataset</i> .....	61
Gambar 4.17	Hasil <i>pre-processing Dataset</i> .....	62
Gambar 4.18	Akurasi model MLP .....	63
Gambar 4.19	Grafik <i>loss</i> model MLP .....	64
Gambar 4.20	Grafik <i>Receiver Operating Characteristic (ROC)</i> .....	65
Gambar 4.21	Grafik presisi- <i>recall</i> .....	65

Gambar 4.22 Kurva pembelajaran model .....	66
Gambar 4.23 Laporan hasil klasifikasi .....	67
Gambar 4.24 <i>Confusion matrix</i> model.....	68
Gambar 4.25 Dataframe hasil prediksi.....	69

## DAFTAR TABEL

Tabel 2.1 Standar TIPHON .....	14
Tabel 3.1 Tabel spesifikasi <i>hardware</i> .....	26
Tabel 3.2 Tabel konfigurasi SDN .....	29
Tabel 3.3 Tabel parameter lingkungan simulasi .....	31
Tabel 3.4 Tabel arsitektur MLP .....	33
Tabel 4.1 Hasil deteksi sistem.....	49
Tabel 4.2 <i>Jitter</i> selama serangan dan setelah mitigasi DDoS .....	51
Tabel 4.3 <i>Delay</i> jaringan pada lalu lintas normal .....	52
Tabel 4.4 <i>Delay</i> selama serangan dan setelah mitigasi DDoS.....	53
Tabel 4.5 <i>Throughput</i> jaringan pada lalu lintas normal .....	54
Tabel 4.6 <i>Throughput</i> selama serangan dan setelah mitigasi DDoS.....	54
Tabel 4.7 <i>Packet loss</i> pada lalu lintas normal .....	55
Tabel 4.8 <i>Packet loss</i> selama serangan dan setelah mitigasi DDoS .....	56
Tabel 4.9 <i>Resource utilization</i> pada lalu lintas normal .....	58
Tabel 4.10 <i>Resource utilization</i> pada lalu lintas DDoS .....	58

# BAB I PENDAHULUAN

## 1.1. Latar Belakang

Dalam era digital saat ini, jaringan komputer telah menjadi salah satu infrastruktur penting bagi organisasi maupun individu. Kemajuan pesat dalam jaringan dan teknologi informasi telah memungkinkan koneksi tanpa batas untuk menyimpan dan mengkomunikasikan informasi dalam skala besar dalam bentuk teks dan suara yang sensitif [1]. Namun, kemajuan teknologi ini juga membawa ancaman terhadap keamanan jaringan. Salah satu serangan utama yang paling dominan menyebabkan penolakan layanan kepada pengguna adalah *Distributed Denial of Service* (DDoS) [2], [3].

Serangan DDoS adalah salah satu ancaman utama terhadap keamanan sistem informasi dan infrastruktur jaringan yang meningkat pesat baik frekuensi maupun intensitasnya setiap tahun. Dalam laporan internet tahunan Cisco, 2018–2023 menunjukkan bahwa tren lonjakan serangan DDoS dari tahun 2018 sebanyak 7,09 juta hingga tahun 2023 diperkirakan serangan DDoS akan meningkat sebanyak 15,4 juta [4]. Serangan DDoS bekerja dengan cara membanjiri paket data secara besar-besaran. Tujuan serangan DDoS adalah untuk menghabiskan *bandwidth* jaringan dan menolak layanan untuk pengguna yang sah dari sistem target [5], [6]. Penelitian dalam deteksi dan pencegahan serangan DDoS dalam transaksi jaringan streaming telah menjadi fokus utama selama lebih dari satu dekade [7]. Penyerang biasanya mengeksploitasi kerentanan dalam *transportasi* data, jaringan, dan protokol lapisan aplikasi seperti *Transmission Control Protocol* (TCP), *User Datagram Protocol* (UDP), dan *Internet Control Message Protocol* (ICMP) [8]. Oleh karena itu, keamanan jaringan menjadi aspek yang sangat penting untuk diperhatikan guna mencegah terjadinya serangan DDoS yang tidak hanya merugikan dari segi keuangan dan operasional, tetapi juga dapat merugikan reputasi suatu entitas atau bisnis. Sayangnya deteksi serangan terbilang kompleks dan sulit dilakukan karena meniru permintaan asli pengguna [9]. Evolusi serangan DDoS dan kompleksitasnya menekankan perlunya terus-menerus mengembangkan

metode deteksi yang lebih canggih dan solusi pencegahan yang adaptif guna mengatasi tantangan yang terus berkembang di dunia siber yang dinamis.

Tidak seperti jaringan tradisional dimana kontrol dan manajemen jaringan terdistribusi di dalam perangkat keras jaringan seperti router dan *switch*, *Software-Defined Networks* (SDN) adalah jenis jaringan berbeda dengan pengaturannya terpusat pada perangkat lunak sehingga memungkinkan SDN menjadi solusi yang dapat diandalkan dalam melindungi dari serangan DDoS dengan kontrol lebih fleksibel dan adaptif [10]. Selama beberapa tahun terakhir, SDN telah terbukti mampu menghasilkan pertahanan yang efektif terhadap berbagai jenis serangan DDoS berbasis jaringan [11]. Saat ini, teknik deteksi serangan DDoS yang paling umum digunakan pada jaringan SDN adalah metode deteksi berbasis statistik, namun metode ini memiliki kelemahan dalam mendeteksi serangan yang baru atau belum pernah terjadi sebelumnya [12]. Pada penelitian yang dilakukan oleh [13] menggunakan metode deteksi DDoS berbasis statistik memiliki beberapa kekurangan. Salah satu kekurangan utamanya adalah kinerjanya yang buruk pada serangan DDoS yang sangat besar seperti serangan berjuta-juta paket yang terjadi dalam waktu yang singkat. Selain itu, metode ini juga dapat menghasilkan banyak *false positive* dan *false negative*. Untuk mengatasi kekurangan ini, beberapa penelitian menggunakan metode berbasis *machine learning*. Metode ini mampu meningkatkan akurasi dan kinerja deteksi DDoS pada serangan yang sangat besar dan kompleks [14]. Dengan kemajuan teknik *deep learning* dan peningkatan kinerja *Graphics Processing Unit* (GPU), terdapat potensi besar untuk mempercepat pengembangan detektor serangan DDoS yang semakin kompleks [12], [15].

*Dataset* yang digunakan pada penelitian ini dibuat pada jaringan SDN dengan menggunakan simulasi *hping3* melibatkan serangkaian langkah untuk merekam data lalu lintas yang dihasilkan oleh perangkat lunak simulasi ini. Dengan memanfaatkan *hping3*, skenario serangan dapat disimulasikan, dan data yang dihasilkan selama simulasi tersebut kemudian direkam untuk membentuk *dataset*. *Dataset* yang dihasilkan dapat mencakup berbagai jenis serangan serangan *Distributed Denial of Service* (DDoS) yang direplikasi dalam simulasi. *Dataset* ini dapat digunakan untuk melatih model deteksi intrusi dan mengembangkan strategi pencegahan yang efektif dalam jaringan SDN. Selain itu, *dataset* ini juga dapat



digunakan untuk menguji keandalan dan kinerja algoritma deteksi intrusi yang ada, serta melakukan penelitian lebih lanjut dalam bidang keamanan jaringan SDN.

Dalam penelitian ini, diajukan sebuah metode deteksi serangan DDoS berbasis *deep learning* untuk jaringan berbasis SDN. Model *multilayer perceptron* (MLP) digunakan untuk mempelajari pola lalu lintas jaringan yang normal dan mengidentifikasi anomali yang mengindikasikan serangan DDoS. Dengan menggunakan *deep learning*, metode ini diharapkan dapat meningkatkan akurasi dan kecepatan deteksi serangan DDoS pada jaringan berbasis SDN. Selain itu, penelitian ini juga diharapkan dapat memberikan kontribusi pada pengembangan teknik *machine learning* dalam deteksi serangan DDoS pada jaringan komputer. Selain itu, hasil penelitian ini juga dapat memberikan panduan bagi organisasi atau perusahaan dalam mengamankan infrastruktur jaringan mereka dari serangan DDoS.

## **1.2. Rumusan Masalah**

Pada penelitian ini terdapat empat permasalahan utama yang diharapkan dapat diselesaikan diantaranya:

1. Bagaimana cara melakukan deteksi dan mitigasi yang cepat dan efektif terhadap serangan DDoS?
2. Bagaimana cara melatih dan menyesuaikan parameter model MLP agar dapat mengklasifikasikan serangan DDoS?
3. Bagaimana mempersiapkan *environment* yang tepat untuk pelatihan dan pengujian model?
4. Bagaimana performa penggunaan algoritma MLP untuk deteksi dan mitigasi serangan DDoS?

## **1.3. Tujuan penelitian**

Pada penelitian ini memiliki beberapa tujuan yang diharapkan tercapai, diantaranya:

1. Mengetahui cara membuat sistem deteksi dan mitigasi DDoS yang cepat dan efektif dengan menggunakan model *Multilayer Perceptron* (MLP).
2. Melatih dan menyesuaikan model MLP agar mampu mengklasifikasikan serangan DDoS.

3. Memilih dan mempersiapkan *environment* dan topologi yang tepat serta relevan untuk melatih dan menguji serangan DDoS.
4. Menganalisis performa *deep learning* khususnya model MLP dalam mendeteksi serangan DDoS untuk mengetahui apakah model *deep learning* mampu untuk mendeteksi dan memitigasi serangan DDoS.

#### 1.4. Manfaat Penelitian

Berikut adalah beberapa manfaat yang dapat diperoleh dari penelitian yang dilakukan, di antaranya adalah sebagai berikut:

1. Bagi pengembangan teknologi keamanan jaringan, penelitian ini diharapkan dapat menciptakan sistem deteksi dan klasifikasi DDoS yang cepat dan efektif menggunakan model MLP, yang dapat meningkatkan respons terhadap ancaman keamanan di jaringan.
2. Bagi akademisi dan peneliti keamanan siber, penelitian ini diharapkan dapat memberikan wawasan baru mengenai penerapan model *Multilayer Perceptron* (MLP) dalam deteksi dan mitigasi serangan DDoS, serta mendukung penelitian selanjutnya dalam pengembangan metode deteksi serta mitigasi yang lebih baik di masa depan.
3. Bagi pengelola jaringan, penelitian ini diharapkan dapat membantu mengurangi waktu dan biaya yang dibutuhkan untuk mengembangkan sistem deteksi dan mitigasi serangan DDoS sehingga lebih efisien dalam implementasinya.

#### 1.5. Batasan Masalah

Mengingat luasnya pembahasan sistem deteksi dan mitigasi serangan DDoS menggunakan metode *deep learning* dengan MLP, maka dalam penelitian ini permasalahan perlu di batasi pada:

1. Penelitian ini hanya berfokus pada kinerja model MLP dalam mendeteksi dan mitigasi serangan DDoS tanpa mempertimbangkan faktor lain yang dapat mempengaruhi kinerja model, seperti kecepatan lalu lintas jaringan dan arsitektur jaringan.
2. Penelitian ini hanya membahas sistem deteksi serangan DDoS menggunakan model *deep learning* pada jaringan SDN tanpa mempertimbangkan mengimplementasikannya pada sistem keamanan lainnya.

3. Data yang digunakan dalam penelitian ini terbatas pada data serangan DDoS yang dihasilkan melalui simulasi pada jaringan SDN.
4. Membangun jaringan SDN lokal dengan menggunakan mininet dan *Ryu controller* yang terdiri dari 15 PC, 5 *switch*, dan 1 *controller*.
5. Parameter *delay*, *throughput*, *packet loss*, dan *resource utilization* digunakan sebagai indikator untuk mengukur kualitas jaringan SDN ketika dilintasi *flow* data normal dan DDoS.

## 1.6. Sistematika Penulisan

Sistematika penulisan skripsi ini terdiri dari lima bab, yaitu di susun sebagai berikut.

### BAB I

Bab ini akan membahas mengenai latar belakang, identifikasi masalah, tujuan penelitian, manfaat penelitian, ruang lingkup penelitian, dan sistematika penulisan.

### BAB II

Bab ini akan membahas mengenai teori-teori dan penelitian terkait yang relevan dengan topik penelitian ini. Bab ini berisi tentang teori jaringan komputer, serangan DDoS, jaringan *Software Defined Network* (SDN), teknik deteksi serangan DDoS, Python, *deep learning*, dan *MLP*.

### BAB III

Bab ini akan menjelaskan mengenai rancangan dan metode penelitian yang digunakan pada penelitian ini, seperti teknik *pre-processing* data, pembuatan jaringan virtual, teknik klasifikasi serangan DDoS menggunakan algoritma *MLP*, alur *training*, dan pengujian hasil penelitian.

### BAB IV

Bab ini akan membahas mengenai hasil penelitian yang telah dilakukan, serta analisis terhadap hasil tersebut.

### BAB V

Bab ini akan berisi kesimpulan dari hasil penelitian yang telah dilakukan, serta saran untuk penelitian selanjutnya yang dapat dilakukan untuk pengembangan lebih lanjut.

## BAB II TINJAUAN PUSTAKA

### 2.1. Artificial Intelligence

*Artificial Intelligence* (AI) adalah sebuah cabang ilmu komputer yang berfokus pada pengembangan algoritma dan teknik-teknik yang memungkinkan mesin untuk melakukan tugas-tugas yang memerlukan kecerdasan manusia [16]. Dalam beberapa tahun terakhir, teknologi AI telah mengalami perkembangan yang sangat pesat dan menjadi topik yang semakin menarik dalam dunia akademik dan industri.

Perkembangan teknologi AI yang pesat memberikan banyak manfaat, hal ini juga membuka potensi risiko keamanan siber yang semakin meningkat. Semakin banyak bot jahat yang menggunakan teknologi AI untuk tujuan merugikan, seperti serangan DDoS yang bisa mencegah layanan internet. Oleh karena itu, keamanan siber menjadi semakin penting dalam pengembangan teknologi AI.

### 2.2. Machine Learning

*Machine learning* adalah konsep penting dalam pengembangan teknologi kecerdasan buatan. Dalam *machine learning*, suatu sistem komputer dapat belajar sendiri dari data yang diberikan sehingga tidak perlu diprogram secara eksplisit oleh manusia [17]. Dengan menggunakan algoritma yang telah diprogram, sistem komputer dapat memproses data secara mandiri dan membuat prediksi atau tindakan berdasarkan data tersebut, yang nantinya dapat digunakan untuk berbagai macam aplikasi, seperti di bidang kesehatan, keuangan, dan otomotif.

Terdapat beberapa jenis *machine learning* yang digunakan dalam pengembangan teknologi AI, seperti *supervised learning*, *unsupervised learning*, dan *reinforcement learning*, yang masing-masing memiliki kegunaan dan aplikasi yang berbeda [18]. *Supervised learning* digunakan untuk memprediksi nilai atau kelas dari suatu data yang sudah diketahui, *semi-supervised learning* digunakan untuk memanfaatkan data yang tidak dilabeli untuk meningkatkan performa model pada data yang dilabeli, sementara *unsupervised learning* digunakan untuk menemukan pola atau struktur dalam data yang belum diketahui, dan *reinforcement*

learning digunakan untuk membuat keputusan berdasarkan tindakan dan pengalaman sebelumnya [17], [19].

### 2.3. *Deep Learning*

*Deep learning* adalah salah satu bidang dalam kecerdasan buatan. *Deep learning* menggunakan arsitektur jaringan syaraf tiruan (neural network) yang lebih besar dan lebih dalam dari jaringan syaraf tiruan biasa. Dalam *deep learning*, jaringan syaraf tiruan yang terdiri dari banyak lapisan dapat belajar secara otomatis dan membuat prediksi atau klasifikasi berdasarkan data yang diberikan [20]. Salah satu keunggulan dari *deep learning* adalah kemampuannya untuk mengolah data yang kompleks, seperti gambar atau suara sehingga dapat digunakan dalam berbagai bidang, seperti pengenalan wajah, deteksi objek, atau bahkan anomali jaringan.

*Neural network* pada *deep learning* terdiri dari beberapa lapisan, seperti lapisan *input*, lapisan *hidden layer*, dan lapisan *output*. Setiap lapisan terdiri dari beberapa neuron yang saling terhubung dengan bobot dan bias tertentu [21]. Proses pelatihan pada *neural network* dilakukan dengan menggunakan *training* data yang telah dilabeli. Pada setiap iterasi pelatihan, *neural network* akan memperbaiki bobot dan bias pada setiap lapisan berdasarkan nilai error pada *output* yang dihasilkan.

Cara kerja *deep Learning* adalah dengan menggunakan beberapa lapisan unit pemrosesan nonlinier untuk mengekstraksi dan mentransformasi fitur. Lapisan yang dekat dengan *input* data mempelajari fitur-fitur sederhana, sedangkan lapisan yang lebih tinggi mempelajari fitur-fitur yang lebih kompleks yang berasal dari fitur-fitur lapisan bawah. Arsitektur membentuk representasi fitur yang hierarkis dan kuat. Berdasarkan cara kerja tersebut, *deep learning* cocok untuk menganalisis dan mengekstraksi pengetahuan yang berguna dari data dalam jumlah besar dan data berdimensi tinggi [22], [23].

### 2.4. **Matriks Evaluasi pada *artificial intelligence* Model**

Evaluasi model *artificial intelligence* memiliki berbagai macam matriks untuk mengukur performa dan efektivitas model. Berikut adalah beberapa matriks evaluasi yang umum digunakan dalam bidang kecerdasan buatan, terutama untuk model pembelajaran mesin dan *deep learning* [24].

### 2.4.1 Matriks Konfusi

Metrik konfusi digunakan untuk mengevaluasi kinerja model prediksi yang membantu dalam melihat performa dari algoritma klasifikasi. Matriks evaluasi terdiri dari empat jenis, yaitu *True Positive* (TP) yang merupakan jumlah data yang diklasifikasikan sebagai positif dan benar, *True Negative* (TN) yang merupakan jumlah data yang diklasifikasikan sebagai negatif dan benar, *False Positive* (FP) yang merupakan jumlah data yang diklasifikasikan sebagai positif namun sebenarnya negatif, dan *False Negative* (FN) yang merupakan jumlah data yang diklasifikasikan sebagai negatif namun sebenarnya positif.

### 2.4.2 Akurasi

Akurasi menunjukkan seberapa benar pengklasifikasi memprediksi titik data, seperti yang ditunjukkan pada Persamaan (3.1).

$$Akurasi = \frac{TP}{TP+TN+FP+FN} \quad (2.1)$$

Akurasi yang baik adalah akurasi yang cukup untuk mencapai tujuan. Mengejar akurasi yang tinggi mungkin tidak akan mendapatkan hasil yang diinginkan. Sebaliknya, fokus harus diarahkan pada memastikan bahwa akurasi model cukup untuk tujuan yang dimaksudkan [25]. Model yang lebih kompleks cenderung memiliki kemampuan yang lebih tinggi untuk mencapai akurasi yang lebih tinggi. Namun, peningkatan kompleksitas sering kali diiringi dengan risiko overfitting, di mana model bekerja sangat baik pada data pelatihan tetapi tidak mampu menggeneralisasi dengan baik pada data baru. Dalam banyak kasus, model yang lebih sederhana dengan akurasi yang cukup mungkin lebih diinginkan karena dapat memberikan keseimbangan antara performa dan generalisasi [26].

### 2.4.3 Presisi

Presisi memberikan kemungkinan seberapa benar pengklasifikasi memprediksi kelas positif. Presisi dihitung dengan Persamaan (3.2)

$$Presisi = \frac{TP}{TP+FN} \quad (2.2)$$

#### 2.4.4 Recall

*Recall* menunjukkan kemungkinan seberapa benar pengklasifikasi dapat mendeteksi kelas positif. *Recall* dihitung dengan Persamaan (3.3)

$$Recall = \frac{TP}{TP+FN} \quad (2.3)$$

#### 2.4.5 F1 Score

Skor F1 adalah keseimbangan rata-rata antara presisi dan *recall*, seperti yang ditunjukkan pada Persamaan (3.4)

$$F_1 \text{ Score} = 2x \frac{Presisi \times Recall}{Presisi + Recall} \quad (2.4)$$

#### 2.4.6 Loss Function

Fungsi kerugian adalah ukuran kesalahan model. Misalnya, dalam kasus klasifikasi biner, *Binary Cross-Entropy Loss* sering digunakan. Untuk regresi, *Mean Squared Error* (MSE) adalah fungsi kerugian yang umum digunakan.

#### 2.4.7 Receiver Operating Characteristic Curve (ROC Curve)

*ROC curve* adalah grafik yang menunjukkan performa model klasifikasi pada semua level *threshold* klasifikasi. Kurva ini memplot *True Positive Rate* (TPR) versus *False Positive Rate* (FPR).

### 2.5. Multilayer Perceptron (MLP)

*Multilayer Perceptron* (MLP) adalah contoh fundamental dari deep neural network. Arsitektur MLP terdiri dari beberapa hidden layer untuk menangkap dan memodelkan hubungan yang lebih kompleks antara *input* dan *output* dalam *dataset* latihan. Setiap neuron dalam hidden layer terhubung dengan neuron-neuron di lapisan sebelumnya dan lapisan berikutnya. Fungsi utamanya melibatkan pemodelan dan prediksi untuk memahami hubungan kompleks antara *input* dan *output* [27]. Dengan menerapkan metode pembelajaran yang tepat, *multilayer perceptron* mampu mengenali pola dan melakukan klasifikasi data dengan tingkat akurasi yang tinggi.

Selama proses pembelajaran, *multilayer perceptron* menggunakan algoritma backpropagation untuk mengoptimalkan bobot pada setiap koneksi antar

neuron. Proses di setiap neuron melibatkan penjumlahan *input* yang diterima, kemudian diproses melalui fungsi aktivasi. Fungsi aktivasi ini memungkinkan neuron menghasilkan *output* non-linear, memungkinkan *multilayer perceptron* untuk memodelkan hubungan yang kompleks antara *input* dan *output* [27]. Oleh karena itu *multilayer perceptron* mampu untuk memahami dan mengenali pola-pola kompleks sehingga memungkinkannya untuk memodelkan dan memprediksi masalah-masalah yang sulit.

Performa *multilayer perceptron* sangat dipengaruhi oleh jumlah dan ukuran lapisan neuron, serta jumlah data latih yang tersedia. Pemilihan fungsi aktivasi juga memainkan peran penting dalam performa jaringan sehingga dapat mengurangi kesalahan prediksi akibat *overfitting* dan meningkatkan akurasi klasifikasi [28]. Pemilihan jumlah neuron dalam lapisan tersembunyi dan jumlah lapisan tersembunyi biasanya melibatkan pendekatan uji coba dan kesalahan heuristik. Ini juga merupakan kasus penerapan penyetelan hyperparameter untuk meningkatkan kinerja jaringan. Oleh karena itu, penentuan parameter-parameter yang tepat menjadi kunci dalam penerapan *multilayer perceptron* pada bidang teknik.

## 2.6. Jaringan Komputer

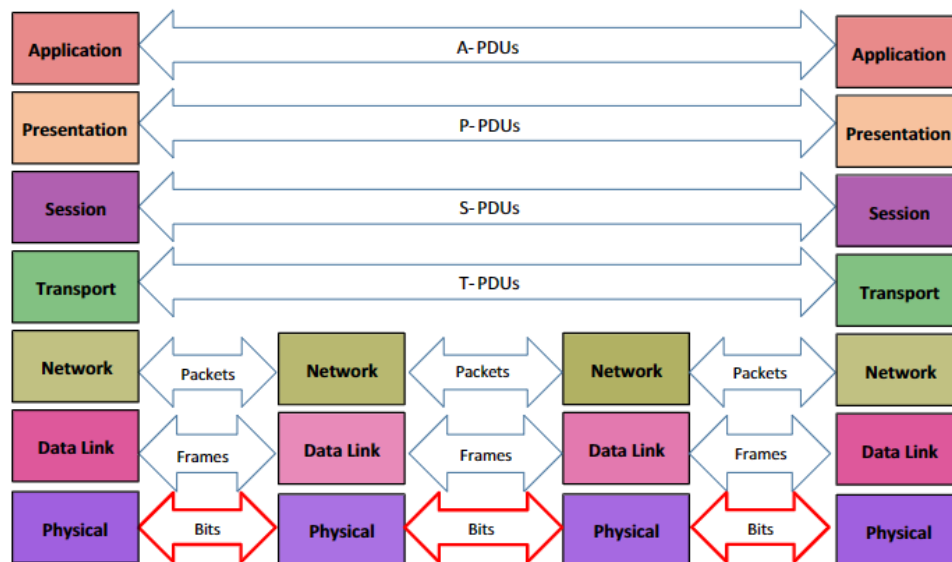
Jaringan komputer merupakan kumpulan beberapa komputer atau perangkat elektronik yang saling terhubung dan dapat berkomunikasi satu sama lain dengan tujuan untuk berbagi sumber daya dan informasi. Jaringan komputer dapat mencakup perangkat keras, perangkat lunak, dan protokol komunikasi yang digunakan untuk memfasilitasi pertukaran data antar perangkat. Jaringan komputer dimana pengguna dapat mengakses sumber daya yang tersedia dari jarak jauh, dan memungkinkan terjadinya kolaborasi dan pertukaran informasi antara pengguna di seluruh dunia disebut dengan internet [29].

Protokol jaringan komputer sangat penting dalam menjalankan jaringan komputer. Protokol jaringan komputer adalah seperangkat aturan yang mengatur bagaimana data ditransmisikan melalui jaringan. Beberapa protokol yang umum digunakan dalam jaringan komputer adalah *Transmission Control Protocol/Internet Protocol* (TCP/IP), *User Datagram Protocol* (UDP), dan *Hypertext Transfer Protocol* (HTTP) [30]. TCP/IP adalah protokol standar yang



digunakan dalam internet dan jaringan komputer modern. Protokol ini bertanggung jawab untuk mengatur pengiriman dan penerimaan data antar perangkat yang terhubung [8]. UDP merupakan protokol yang lebih sederhana dan lebih cepat dari TCP/IP, namun kurang dapat diandalkan karena tidak memiliki mekanisme pengontrolan kesalahan [8]. Sedangkan HTTP adalah protokol yang digunakan untuk mengakses halaman web di internet [31].

Proses komunikasi data melalui jaringan memiliki kerangka logika terstruktur yang di sebut model OSI (*Open Sistem Interconnection*). Model Osi menerapkan teknik penataan yang di sebut *layering*. Osi *layer* terdiri dari 7 lapisan dengan masing-masing mempunyai peran dan fungsi yang berbeda [8].



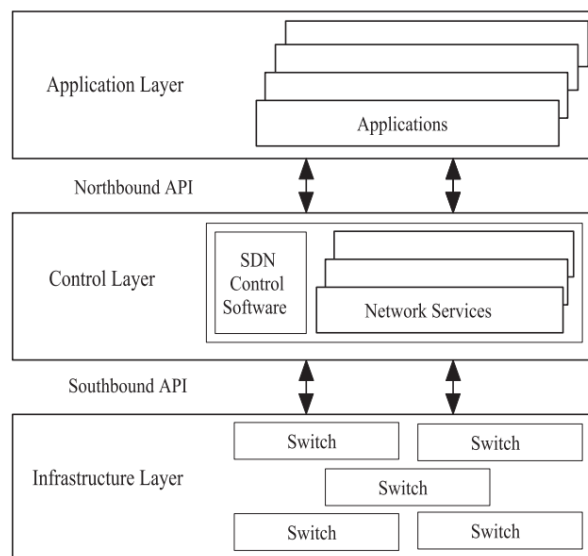
Gambar 2.1 Arsitektur jaringan berdasarkan model OSI [30].

Berikut ini merupakan penjelasan untuk gambar 2.3. Model OSI terdiri dari tujuh lapisan yang mengatur komunikasi jaringan. Lapisan Aplikasi (7) menyediakan layanan langsung ke aplikasi pengguna. Lapisan Presentasi (6) mengonversi data ke format yang dipahami aplikasi. Lapisan Sesi (5) mengelola sesi komunikasi. Lapisan *Transport* (4) memastikan pengiriman data yang andal. Lapisan Jaringan (3) merutekan paket data di jaringan. Lapisan Tautan Data (2) memproses transfer data antar perangkat. Lapisan Fisik (1) menangani transmisi sinyal fisik data. Setiap lapisan mempunyai fungsi khusus untuk mendukung operasi jaringan yang efektif. [30], [31].

## 2.7. *Software-Defined Networks*

*Software Defined Network* (SDN) merupakan suatu konsep dalam bidang jaringan komputer yang memisahkan lapisan kontrol (*control plane*) dari lapisan penerusan (*data plane*) dalam infrastruktur jaringan. Dalam konteks ini, kontrol dan manajemen jaringan diatur secara terpusat dan terpisah dari perangkat keras jaringan fisik. Penerapan SDN memungkinkan administrator jaringan untuk mengelola dan mengontrol jaringan dengan lebih fleksibel dan efisien [32]. Konsep ini memungkinkan penggunaan perangkat lunak untuk mengatur dan mengendalikan alur lalu lintas jaringan, memfasilitasi penyesuaian dan pengaturan yang lebih cepat dan mudah.

Keuntungan dari SDN melibatkan beberapa aspek krusial, seperti fleksibilitas, efisiensi, dan skalabilitas [33]. Pertama, dari segi fleksibilitas, SDN memberikan kebebasan kepada administrator untuk mengubah dan mengkonfigurasi jaringan dengan cepat dan mudah melalui perangkat lunak. Kedua, dalam hal efisiensi, SDN menyajikan sentralisasi kontrol yang mengoptimalkan penggunaan sumber daya jaringan. Oleh karena itu, SDN dapat menghindari overprovisioning dan meningkatkan efisiensi penggunaan jaringan secara menyeluruh. Terakhir, skalabilitas menjadi keunggulan utama. SDN memungkinkan jaringan untuk diperluas dengan lebih mudah dan efisien karena kontrol dan manajemen jaringan terpusat.



Gambar 2.2 Jenis serangan siber [33]

Gambar 2.2 menunjukkan arsitektur Software-Defined Network (SDN), yang terdiri dari 5 komponen utama yaitu, *application layer*, *control layer*, *infrastructure layer*, *southbound interface*, dan *northbound interface*. Lapisan Aplikasi berfungsi sebagai ruang untuk aplikasi dan layanan jaringan, yang mencakup segala sesuatu mulai dari manajemen *bandwidth* hingga optimalisasi lalu lintas. Lapisan Kontrol memegang peran sentral dalam mengatur pengaturan dan keputusan jaringan, berkomunikasi dengan perangkat keras jaringan melalui *southbound interface* seperti *OpenFlow*. Lapisan Infrastruktur, merupakan perangkat keras fisik seperti *switch* dan router, yang mengirimkan lalu lintas data sesuai dengan instruksi dari lapisan kontrol. Dengan *southbound interface*, dan *northbound interface*, SDN memungkinkan komunikasi yang efisien antara lapisan kontrol, aplikasi, dan infrastruktur. Sebagai hasilnya, SDN meningkatkan efisiensi manajemen, fleksibilitas yang ditingkatkan, dan adaptabilitas dinamis jaringan sesuai kebutuhan aplikasi.

Jaringan berbasis Software-Defined Network (SDN) memiliki dua komponen utama dalam pengembangan dan pengujian yaitu Mininet dan Ryu *controller*. Dengan menggunakan Mininet sebagai emulator dan Ryu sebagai *controller*, peneliti dapat menguji aplikasi SDN dalam lingkungan terkendali sebelum mengimplementasikannya dalam jaringan fisik yang sebenarnya. Hal ini memudahkan pengembangan, pengujian, dan validasi solusi SDN tanpa memerlukan infrastruktur jaringan yang rumit.

### 2.7.2 Mininet

Mininet adalah emulator jaringan SDN yang memungkinkan pengguna membuat jaringan virtual yang terdiri dari *host*, *switch*, dan *controller* SDN. Dengan Mininet, pengguna dapat membuat topologi jaringan yang terisolasi dalam lingkungan virtual, memungkinkan pengembang dan peneliti untuk menguji dan mengembangkan aplikasi SDN tanpa memerlukan infrastruktur fisik yang sebenarnya. Mininet menyediakan berbagai opsi konfigurasi untuk *host*, *switch*, dan *controller* sehingga pengguna dapat mensimulasikan berbagai skenario jaringan.

### 2.7.3 Ryu controller

Ryu adalah sebuah platform pengembangan *controller* SDN yang bersifat open-source. Ryu menyediakan kerangka kerja yang kuat dan fleksibel untuk membangun aplikasi pengontrol jaringan SDN. *Controller* ini mendukung protokol *OpenFlow*, yang memungkinkan pengontrol untuk mengambil keputusan terkait arus lalu lintas dan mengonfigurasi perangkat keras jaringan sesuai kebutuhan. Ryu juga dapat digunakan untuk mengimplementasikan berbagai kebijakan jaringan, seperti deteksi intrusi, manajemen *bandwidth*, dan lainnya.

### 2.8. Quality of Service

*Quality of Service* (QoS) adalah metode untuk mengukur dan memastikan kualitas jaringan komputer berdasarkan atribut kinerja yang terhubung dengan layanan tertentu [34]. QoS memiliki kemampuan untuk mendefinisikan atribut layanan jaringan yang tersedia, memastikan kehandalalan dalam penyampaian data, serta mendefinisikan tingkat kecepatan dalam sistem komunikasi. Pengukuran QoS memiliki standar yang digunakan untuk mengevaluasi kinerja jaringan telekomunikasi dan internet yaitu standar TIPHON (*Telecommunications and Internet Protocol Harmonization Over Networks*) [35]. Standar ini menetapkan kategori kualitas untuk 4 parameter QoS utama, yaitu *throughput*, *delay*, *jitter*, dan *packet loss*. Berikut adalah tabel yang menunjukkan kategori standar TIPHON untuk parameter-parameter tersebut.

Tabel 2.1 Standar TIPHON [35]

Parameter QoS	Sangat Bagus	Bagus	Sedang	Buruk
<i>Throughput</i>	> 2,1 Mbps	1200 kbps – 2,1 Mbps	700 - 1200 kbps	338 - 700 kbps
<i>Delay</i>	< 150 ms	150-300 ms	300 - 450 ms	> 450 ms
<i>Jitter</i>	0 ms	0 - 75 ms	75 - 125 ms	125 - 225 ms
<i>Packet Loss</i>	0 - 2%	3 - 14%	12 - 24%	> 25%

Tabel di atas menunjukkan standar TIPHON untuk berbagai kategori kualitas parameter QoS. Dengan menggunakan standar ini, kinerja jaringan dapat di bandingkan dan dinilai secara lebih terstruktur dan sistematis. Kategori-kategori

ini membantu dalam memahami seberapa baik jaringan berfungsi dan area mana yang perlu ditingkatkan. Implementasi QoS sesuai dengan standar TIPHON memastikan bahwa jaringan dapat memberikan layanan yang optimal dan dapat diandalkan kepada penggunanya. Berikut merupakan penjelasan dari masing-masing parameter.

### 2.8.2 *Throughput*

*Throughput* merupakan jumlah total data yang melewati *bandwidth* pada waktu tertentu. *Throughput* didapatkan dari jumlah paket yang diterima dibagi dengan waktu pengiriman [35]. *Throughput* pada umumnya bersifat dinamis, tergantung pada kondisi lalu lintas jaringan. Oleh karena itu *throughput* dapat mencerminkan kapabilitas sebenarnya dari suatu jaringan dalam mengirimkan suatu data. Semakin tinggi *throughput*, semakin baik jaringan dalam menangani volume data. Nilai *throughput* dapat dikatakan baik atau buruk berdasarkan standar TIPHON yang dapat dilihat pada Tabel 2.1.

### 2.8.3 *Delay*

*Delay* adalah waktu yang diperlukan paket data untuk bergerak dari pengirim ke penerima [34], biasanya diukur dalam milidetik (ms). *Delay* mencerminkan seberapa cepat data dapat berpindah dalam jaringan. *Delay* rendah menunjukkan jaringan yang responsif dan efisien. Namun, nilai *delay* dapat meningkat karena kemacetan jaringan, jarak fisik, atau kecepatan pengolahan. Menurut standar TIPHON, nilai *delay* dapat dikategorikan sebagai baik atau buruk, yang bisa dilihat pada Tabel 2.1.

### 2.8.4 *Jitter*

*Jitter* dapat dikatakan sebagai variasi waktu kedatangan paket data yang diterima [34]. Variasi ini dapat mengganggu aplikasi *real-time* seperti panggilan suara atau video streaming. *Jitter* diukur dalam milidetik (ms) dan nilai yang lebih rendah menunjukkan jaringan yang stabil. *Jitter* yang tinggi dapat menyebabkan kualitas komunikasi yang buruk. Standar TIPHON juga menyediakan kategori untuk menilai kualitas *jitter* pada jaringan, yang dapat dilihat pada Tabel 2.1.

### 2.8.5 Packet loss

*Packet Loss* terjadi ketika satu atau lebih paket data gagal mencapai tujuannya dalam transmisi jaringan. *Packet loss* dapat disebabkan oleh kemacetan jaringan, atau kesalahan perangkat keras [34]. *Packet loss* yang rendah menunjukkan jaringan yang andal dan efisien, sementara *packet loss* yang tinggi dapat menyebabkan masalah seperti keterlambatan pengiriman data, atau hilangnya informasi penting. Standar TIPHON menetapkan ambang batas untuk menilai kualitas *packet loss*, yang juga dapat dilihat pada Tabel 2.1.

### 2.9. Cyber Security

*Cyber Security* atau keamanan siber adalah teknik yang di rancang untuk melindungi sistem dari serangan, pencurian, kerusakan, modifikasi, atau akses yang tidak sah [24]. Tujuan utama *Cyber Security* adalah untuk memastikan perlindungan data. Oleh karena itu, Sistem komputer yang aman dan stabil harus memastikan kerahasiaan, ketersediaan, dan integritas informasi. Prinsip ini disebut triad CIA (*confidentiality, integrity, dan availability*) [8], [36].



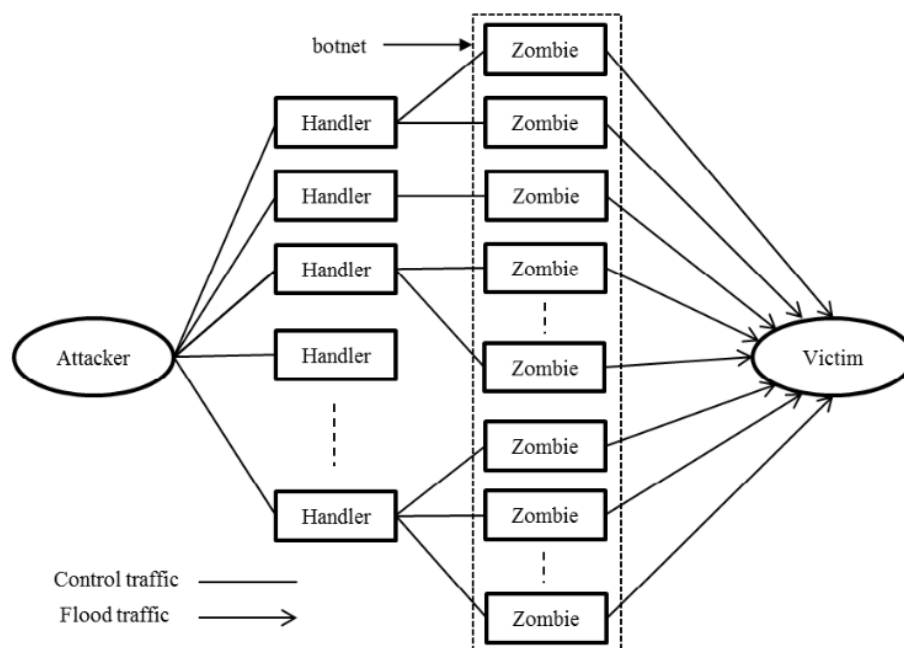
Gambar 2.3 Jenis serangan siber [37]

*Confidentiality, Integrity, dan Availability (CIA)* adalah tiga prinsip utama yang harus dipertimbangkan pada jaringan komputer. *Confidentiality* mengacu pada perlindungan data dari akses yang tidak sah atau tidak diizinkan [38]. *Integrity* berarti memastikan bahwa data yang tersimpan tidak dapat dimodifikasi oleh pihak yang tidak berwenang dan sesuai dengan kebijakan dan prosedur organisasi [39]. *Availability* berhubungan dengan ketersediaan sumber daya atau informasi yang diinginkan [36].

## 2.10. Serangan DDoS

Serangan DDoS (*Distributed Denial of Service*) merupakan jenis serangan siber yang bertujuan untuk menyerang infrastruktur komputer dengan cara membanjiri server dengan lalu lintas internet yang berlebihan [40]. Serangan DDoS dapat dilakukan dengan menggunakan banyak komputer yang terinfeksi oleh malware atau dengan menggunakan jaringan botnet yang terdiri dari ribuan komputer sehingga server menjadi overload dan tidak dapat memproses permintaan dari pengguna yang sah [41]. Dalam serangan DDoS, lalu lintas yang berlebihan dan palsu dapat berasal dari ribuan atau bahkan jutaan sumber yang terdistribusi di seluruh dunia sehingga sulit untuk mengidentifikasi dan memblokir serangan.

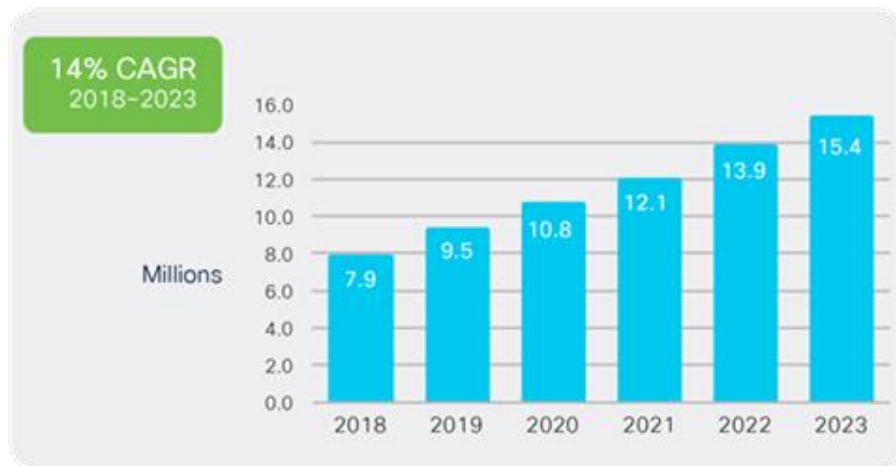
Serangan DDoS menjadi semakin canggih dalam beberapa tahun terakhir. Saat ini, alat serangan DDoS telah menjadi otomatis dan canggih sehingga memungkinkan penyerang untuk mengeksekusi serangan secara otomatis dengan sedikit campur tangan manusia. Pada dasarnya setiap penyerang menggunakan teknik perentasan yang berbeda untuk mencari kelemahan dalam sistem komputer yang terhubung ke internet. Skema serangan DDoS yaitu penyerang akan mencoba mengontrol mesin komputer sebanyak mungkin hingga membentuk zombie botnet seperti yang di tunjukan pada gambar 2.5. Botnet adalah jaringan komputer atau perangkat yang terhubung ke Internet yang berkomunikasi satu sama lain [42].



Gambar 2.1 Arsitektur serangan DDoS [41]

Satu zombie hanya menyediakan sedikit data, tetapi lalu lintas kumulatif dari banyak zombie yang muncul di sistem target akan menjadi sangat besar dan menghabiskan sumber daya. Ukuran botnet menentukan tingkat dan besarnya intensitas serangan. Botnet dalam jumlah besar bisa melakukan serangan yang menghancurkan dan parah. Sulit bagi sistem pertahanan untuk membedakan antara lalu lintas abnormal *flash crowds* dari serangan DDoS karena mereka hanya berbeda dalam beberapa parameter. *Flash Crowds* (FC) merupakan jenis lalu lintas jaringan yang mirip dengan lalu lintas DDoS, tetapi berasal dari pengguna yang sah. FC hampir serupa dengan serangan DDoS karena pengguna menggunakan akses ke sistem secara bersamaan dan tiba-tiba [41].

Dalam laporan Internet tahunan Cisco pada gambar 2.6, menunjukkan tren lonjakan serangan DDoS dari 2018 hingga 2023.



Gambar 2.2 Tren global serangan DDoS [4]

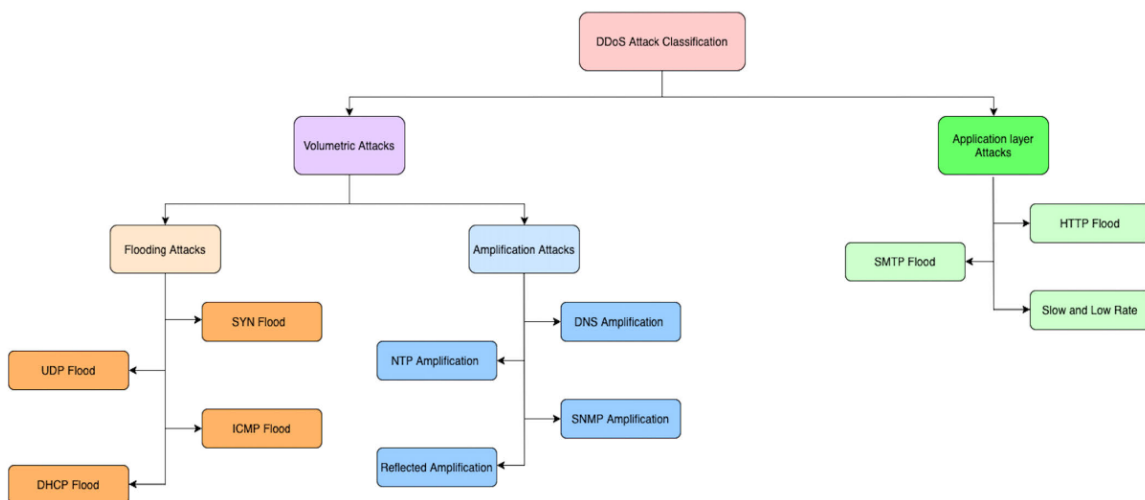
Dapat diamati bahwa pada tiap tahun terjadi peningkatan serangan DDoS. Mulai dari tahun 2018 terdapat serangan DDoS sebanyak 7,9 juta, hingga pada tahun 2023 terdapat 15,4 juta serangan [4].

Laporan Kaspersky menyebutkan bahwa terdapat peningkatan serangan DDoS pada kuartal 1 tahun 2022 yaitu sebanyak 1.5 kali lipat atau sebesar 46% dibanding tahun sebelumnya [43]. Serangan DDoS terbesar terjadi pada tahun 2016 dengan menggunakan botnet Mirai baru, dan serangan tersebut ditargetkan pada server Dyn yang bergerak di sebagian besar infrastruktur *Domain Name Sistem* (DNS) [41]. Mirai adalah sumber utama serangan pada perangkat IoT. Perkiraan



skala serangan pada server Dyn yaitu sebesar 1,2 terabit (1.200 Gbps) dengan sekitar 100.000 agen penyerang [42].

Serangan DDoS dapat di klasifikasikan berdasarkan kerentanan terhadap jaringan dan bot yang tersedia. Pengklasifikasian serangan DDoS dijelaskan pada gambar 2.7.



Gambar 2.3 Taksonomi serangan DDoS [17].

- *HTTP flood attacks*: Serangan banjir HTTP dimaksudkan untuk membanjiri sumber daya server web menggunakan sejumlah besar permintaan HTTP yang dihasilkan oleh botnet [17].
- *Slow- and low-rate attacks*: Dalam kasus ini penyerang menghasilkan lalu lintas ke server web korban dan kemudian menjaga koneksi tetap terbuka (atau aktif) tanpa balasan sampai sumber daya server dikonsumsi [44].
- *Session Initiation Protocol (SIP)*: adalah protokol yang paling umum untuk mengelola pensinyalan di antara pihak-pihak komunikasi untuk menyediakan fungsionalitas yang diperlukan untuk mendaftarkan pengguna, memeriksa status, dan mengelola sesi [45].
- *Reflector attack*: Tujuan utama serangan reflektor adalah untuk menutupi identitas penyerang sebenarnya melalui penggunaan reflektor pihak ketiga dan kemudian memanfaatkan sumber daya mereka [46].
- *DNS amplification attack*: Dalam kasus serangan DDoS jenis ini, penyerang bertujuan untuk mengeksploitasi kerentanan dalam domain name sistem (DNS) untuk mengubah (yaitu, memperkuat) pesan kecil yang awalnya dikirim oleh penyerang menjadi pesan yang jauh lebih besar [42].

- *SYN flooding attack*: Dalam kasus serangan banjir SYN, server korban menerima sejumlah besar paket SYN tetapi tidak pernah menerima ACK terakhir yang diperlukan untuk menyelesaikan metode jabat tangan tiga arah (*three-way handshake*) melalui protokol TCP [17]. Akibatnya, antrian server kewalahan, yang menyebabkan semua permintaan yang masuk dari klien yang sah dibatalkan
- *User Datagram Protocol (UDP) flooding attack*: adalah jenis serangan DDoS di mana penyerang menargetkan dan membanjiri *port* acak pada server yang ditargetkan dengan paket IP termasuk paket UDP [47].
- *ICMP flooding attack*: juga dikenal sebagai serangan *ping*, bertujuan untuk menargetkan server korban dengan sejumlah besar permintaan gema. Server korban yang ditargetkan harus mengirimkan paket respons untuk setiap permintaan yang diterima dari pengirim [17].

### 2.11. Python

Python adalah bahasa pemrograman tingkat tinggi yang sangat populer di kalangan ilmuwan data dan pengembang *machine learning*. Hubungan Python dengan *deep learning* sangat erat karena banyak library dan framework yang tersedia untuk membangun *deep learning* seperti NumPy, Pandas, SciPy, dan TensorFlow [48], [49]. Dalam pengolahan gambar, Python memiliki library OpenCV dan Pillow yang dapat digunakan sebagai pengkonversi data ke dalam bentuk gambar dan mengolah gambar secara lebih kompleks seperti deteksi objek.

### 2.12. Kajian Pustaka

Bagian ini dirancang untuk mengumpulkan dan menganalisis hasil penelitian terdahulu yang telah dilakukan sebelumnya terkait topik klasifikasi serangan DDoS dan penggunaan CNN untuk metode pengklasifikasikan. Dalam kajian pustaka ini, akan dipaparkan metode, teknik yang telah digunakan dan kesimpulan dari penelitian sebelumnya. Tujuannya adalah untuk memberikan landasan teori yang kuat dan komprehensif sebagai dasar dalam melakukan penelitian lanjutan mengenai topik ini.

Penelitian [40] menunjukkan bahwa DDoS adalah serangan yang paling sering terjadi pada jaringan dan perangkat IoT. Penelitian ini menggunakan transfer learning ResNet18 sebagai metode pengklasifikasian serangan DDoS. Hasil penelitian tersebut dibagi menjadi 2, yaitu klasifikasi binary yang digunakan untuk mendeteksi serangan DDoS dan klasifikasi *multi-class* yang menentukan jenis serangan DDoS. Dalam penelitian tersebut, ResNet18 berhasil mengenali serangan DoS dan DDoS dengan persentase 99,9% dan mampu mengklasifikasikan serangan seperti SYN *flood*, UDP *flood*, dan NTP Amplification dengan akurasi sebesar 87% [40]. Hasil dari penelitian ini menunjukkan bahwa penggunaan teknik transfer learning dapat meningkatkan kemampuan sistem untuk mengenali serangan DDoS serta lebih andal dan efektif dibandingkan dengan teknik aturan statis yang telah ditentukan sebelumnya.

Penelitian selanjutnya membahas perbandingan kinerja dari mekanisme pengklasifikasian serangan DDoS pada *Software Defined Network* (SDN). Hasil penelitian terdahulu menunjukkan bahwa metode *machine learning* merupakan pendekatan yang paling efektif dalam mendeteksi serangan DDoS [50]. Namun, metode *machine learning* memerlukan jumlah data yang besar untuk dilatih dan seringkali memerlukan waktu komputasi yang lebih lama untuk diproses. Metode statistik dan metode *threshold*, di sisi lain, tidak memerlukan banyak data dan dapat bekerja dengan cepat [50]. Namun, metode ini seringkali kurang akurat dibandingkan dengan metode *machine learning*. Oleh karena itu, penggunaan metode *machine learning*, dalam mengklasifikasikan serangan DDoS dianggap lebih unggul dibandingkan dengan metode statistik dan *threshold*.

Penelitian yang dilakukan oleh [46] membahas tentang efisiensi penggunaan algoritma *deep learning* untuk mendeteksi aktivitas abnormal seperti DDoS. Penelitian ini menggunakan metode *convolutional neural network* (CNN). Selain itu, algoritma lain juga digunakan seperti *decision tree* (D-Tree), *support vector machine* (SVM), *K-nearest neighbors* (K-NN), dan *neural network* (NN) sebagai pembanding kinerja. Dengan menggunakan iterasi yang sama, semua algoritma di atas dibandingkan kinerjanya sebagai pengklasifikasi serangan DDoS menggunakan kumpulan data serangan DDoS standar seperti NSL-KDD dan hasil simulasi serangan. Hasil penelitian menunjukkan bahwa penggunaan algoritma

CNN bekerja dengan lebih baik daripada pengklasifikasi lainnya dengan akurasi sebesar 99% [46].

Dalam penelitian yang berjudul "*Deep Learning enabled Intrusion Detection and Prevention Sistem over SDN Networks*," yang dilakukan oleh [51], kajian ini mengeksplorasi penerapan teknik *deep learning* untuk meningkatkan keamanan jaringan *Software Defined Network* (SDN). Fokus utama dari penelitian ini adalah merancang dan mengimplementasikan sistem deteksi intrusi dan pencegahan berbasis *deep learning* guna melindungi jaringan SDN dari berbagai serangan siber, khususnya *secure shell* (SSH) *brute-force attacks*. Hasil penelitian menunjukkan bahwa *Multi-Layer Perceptron* (MLP) memiliki kemampuan yang lebih baik dalam menangani masalah non-linear dan kompleks dibandingkan dengan algoritma lain seperti *Convolutional Neural Network* (CNN), *Long Short-Term Memory* (LSTM), dan *Stacked Auto-encoder* (SAE). Dengan demikian, berdasarkan temuan ini, dapat disimpulkan bahwa MLP menjadi salah satu algoritma *deep learning* yang unggul dan memiliki aplikasi luas di berbagai bidang.

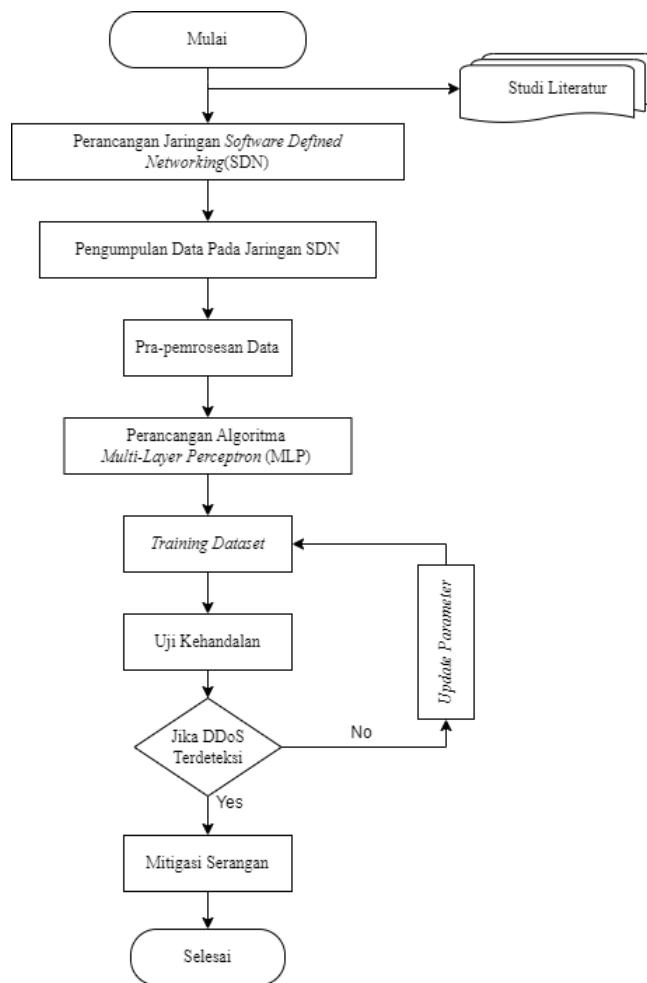
Hasil penelitian yang dilakukan oleh [52] mengenai mekanisme deteksi serangan DDoS berbasis *deep learning* dengan menggunakan sFlow dan *adaptive polling* sampling dalam *Software-Defined Networking* (SDN) menunjukkan bahwa penggunaan metode sampling berbasis sFlow dan adaptive polling di lapisan data dapat mengurangi beban jaringan dan memungkinkan *Intrusion Detection System* (IDS) seperti Snort untuk merekam aktivitas jahat secara efektif. Penelitian ini menggunakan Snort IDS dan model *deep learning Stacked Autoencoders* (SAE) untuk meningkatkan akurasi deteksi dengan data lalu lintas jaringan yang dikumpulkan secara *real-time*. Evaluasi menunjukkan bahwa pendekatan berbasis sFlow menghasilkan akurasi deteksi yang lebih tinggi dibandingkan dengan metode *adaptive polling*, dengan tingkat *True Positive* sebesar 95% dan tingkat *False Positive* kurang dari 4%. Hasil ini menunjukkan bahwa penggunaan teknik sampling yang tepat dalam lingkungan SDN dapat secara signifikan meningkatkan kemampuan deteksi serangan DDoS dengan efisiensi sumber daya yang lebih baik.

## BAB III METODOLOGI PENELITIAN

Metodologi penelitian ini bertujuan untuk merancang algoritma *Multi-Layer Perceptron* (MLP) sebagai sistem pendeteksi serangan DDoS menggunakan metode *deep learning*. Selain itu, perancangan sistem *Software-Defined Network* (SDN) juga akan menjadi komponen penting dalam pengembangan sistem pendeteksi serangan DDoS.

### 3.1. Alur Penelitian

Penelitian ini melibatkan beberapa tahapan proses untuk mencapai hasil yang diharapkan, di antaranya adalah sebagai berikut.



Gambar 3.1 Diagram alir penelitian

### 3.2. Komponen Penelitian

Berikut adalah komponen yang digunakan dalam penelitian ini:

#### 3.2.1 Python

Python adalah bahasa pemrograman yang sering digunakan untuk melakukan pengolahan data, termasuk dalam pembuatan topologi jaringan serta mengatur alur dari data *flow*. Python juga dapat digunakan untuk melakukan berbagai tugas lainnya seperti analisis data, atau pembuatan model *machine learning*.

#### 3.2.2 Sistem Operasi Ubuntu 22.04

Ubuntu 22.04 adalah distribusi Linux berbasis Debian yang digunakan sebagai lingkungan pengembangan dan eksekusi untuk komponen-komponen penelitian ini. Ubuntu dipilih karena stabilitasnya, serta menyediakan lingkungan yang ideal untuk pengembangan jaringan dan pengujian perangkat lunak.

#### 3.2.3 Traffic Generator

*Traffic generator* digunakan untuk menghasilkan lalu lintas jaringan yang digunakan dalam pengujian dan simulasi seperti *ping*, *iPerf*, dan *hping3*. Alat ini memungkinkan pembuatan skenario lalu lintas yang berbeda, termasuk lalu lintas normal dan serangan DDoS, untuk menguji efektivitas sistem deteksi dan mitigasi.

#### 3.2.4 Wireshark

Wireshark adalah alat analisis protokol jaringan yang digunakan untuk menangkap dan memeriksa paket data yang dikirim melalui jaringan. Dengan Wireshark, peneliti dapat menganalisis lalu lintas jaringan secara mendalam, mengidentifikasi pola serangan, dan memverifikasi bahwa mitigasi DDoS bekerja dengan baik.

#### 3.2.5 S-flow RT

sFlow RT adalah alat monitoring jaringan *real-time* yang digunakan untuk mengumpulkan dan menganalisis statistik lalu lintas jaringan. sFlow RT memungkinkan deteksi anomali dalam lalu lintas jaringan dan memberikan data yang diperlukan untuk mendeteksi serangan DDoS.

### 3.2.6 Jupyter Notebook

Jupyter Notebook adalah aplikasi web *open-source* yang digunakan untuk membuat dan berbagi dokumen yang berisi kode, persamaan, visualisasi, dan narasi teks. Jupyter Notebook pada penelitian ini digunakan dalam melakukan eksplorasi data, analisis data, pembuatan model *machine learning*, dan masih banyak lagi.

### 3.2.7 Mininet

Mininet adalah emulator jaringan *open-source* yang memungkinkan pengguna untuk membuat jaringan virtual untuk pengujian dan eksperimen. Salah satu fitur utama Mininet adalah kemampuannya membuat topologi jaringan dengan *switch* virtual, *host*, dan koneksi. Selain itu, Mininet terintegrasi dengan *OpenFlow*, protokol untuk jaringan SDN. Integrasi Mininet dengan *OpenFlow* memungkinkan pengguna membuat dan menguji arsitektur dan aplikasi jaringan berbasis SDN.

### 3.2.8 Ryu controller

Ryu *controller* adalah sebuah platform pengembangan perangkat lunak berbasis *OpenFlow* yang digunakan untuk mengimplementasikan jaringan SDN. Ryu menyediakan berbagai fitur dan fungsionalitas yang memungkinkan pengguna untuk mengendalikan dan mengelola jaringan dengan cara yang lebih fleksibel dan dinamis. Dengan menggunakan protokol *OpenFlow*, Ryu *controller* memperbolehkan pengguna untuk secara sentral mengontrol aliran lalu lintas jaringan dan mengatur kebijakan jaringan secara dinamis. Dengan kombinasi antara Mininet sebagai emulator jaringan dan Ryu *controller* sebagai SDN, pengguna dapat melakukan simulasi dan pengujian jaringan SDN secara efektif. Pengguna dapat membuat topologi jaringan yang kompleks di Mininet dan mengontrolnya melalui Ryu *controller* sehingga memungkinkan pengguna untuk mengembangkan dan menguji berbagai aplikasi dan kebijakan jaringan SDN.

### 3.2.9 Hardware

Penelitian ini menggunakan sebuah komputer yang dilengkapi dengan spesifikasi yang memadai untuk melakukan training model *Multi-Layer Perceptron* (MLP). Selain itu, juga digunakan sebuah mesin virtual yang berfungsi untuk menjalankan simulasi jaringan dengan tujuan memperoleh hasil yang akurat dan representatif. Dengan menggunakan komputer dan mesin virtual ini, penelitian

dapat dilakukan dengan lebih efektif dan efisien, serta menghasilkan hasil yang dapat diandalkan. Berikut merupakan spesifikasi komputer yang di gunakan:

Tabel 3.1 Tabel spesifikasi *hardware*

Komponen	Spesifikasi Asli	Spesifikasi Mesin Virtual
Processor	AMD Ryzen 9 4900HS (8 Core)	AMD Ryzen 9 4900HS (2 Core)
RAM	16 GB DDR4	4 GB DDR4
Storage	1 TB SSD	40 GB SSD
GPU	NVIDIA GeForce RTX 2060 Max-Q	AMD Radeon Graphics
Sistem operasi	Windows 11	Ubuntu 22.4

Tabel 3.1 menjelaskan spesifikasi komputer yang digunakan dalam penelitian ini. Terdapat dua komponen yang dijelaskan, yaitu spesifikasi asli untuk komputer utama yang digunakan untuk melatih model MLP, dan spesifikasi mesin virtual untuk mesin virtual yang digunakan untuk menjalankan simulasi jaringan. *Hardware* tersebut digunakan untuk menjalankan proses-proses komputasi yang membutuhkan daya komputasi yang tinggi. Dengan menggunakan komputer utama yang kuat dan mesin virtual yang memadai, penelitian ini dapat dilakukan dengan efektif dan efisien. Kombinasi spesifikasi yang tepat memungkinkan peneliti untuk melatih model MLP dengan cepat dan menjalankan simulasi jaringan dengan akurasi yang tinggi.

### 3.3. Metode Penelitian

Penelitian ini memiliki beberapa tahapan proses untuk mencapai hasil yang diharapkan, diantaranya adalah sebagai berikut.

#### 3.3.1 Studi Literatur

Studi literatur merupakan tahap awal dari metodologi penelitian yang dilakukan. Pada tahap ini, peneliti melakukan pencarian informasi mengenai serangan DDoS pada jaringan SDN, serta metode *deep learning* dengan algoritma MLP. Proses ini mencakup pembacaan dan pengumpulan informasi dari berbagai sumber seperti jurnal ilmiah, buku, dan artikel online. Setelah informasi terkumpul, peneliti melakukan analisis dan evaluasi terhadap informasi tersebut untuk memastikan bahwa sumbernya terpercaya dan relevan dengan topik penelitian. Selain itu, peneliti juga dapat mengidentifikasi kekurangan atau kesenjangan dalam penelitian terdahulu dan menemukan potensi untuk penelitian lebih lanjut pada



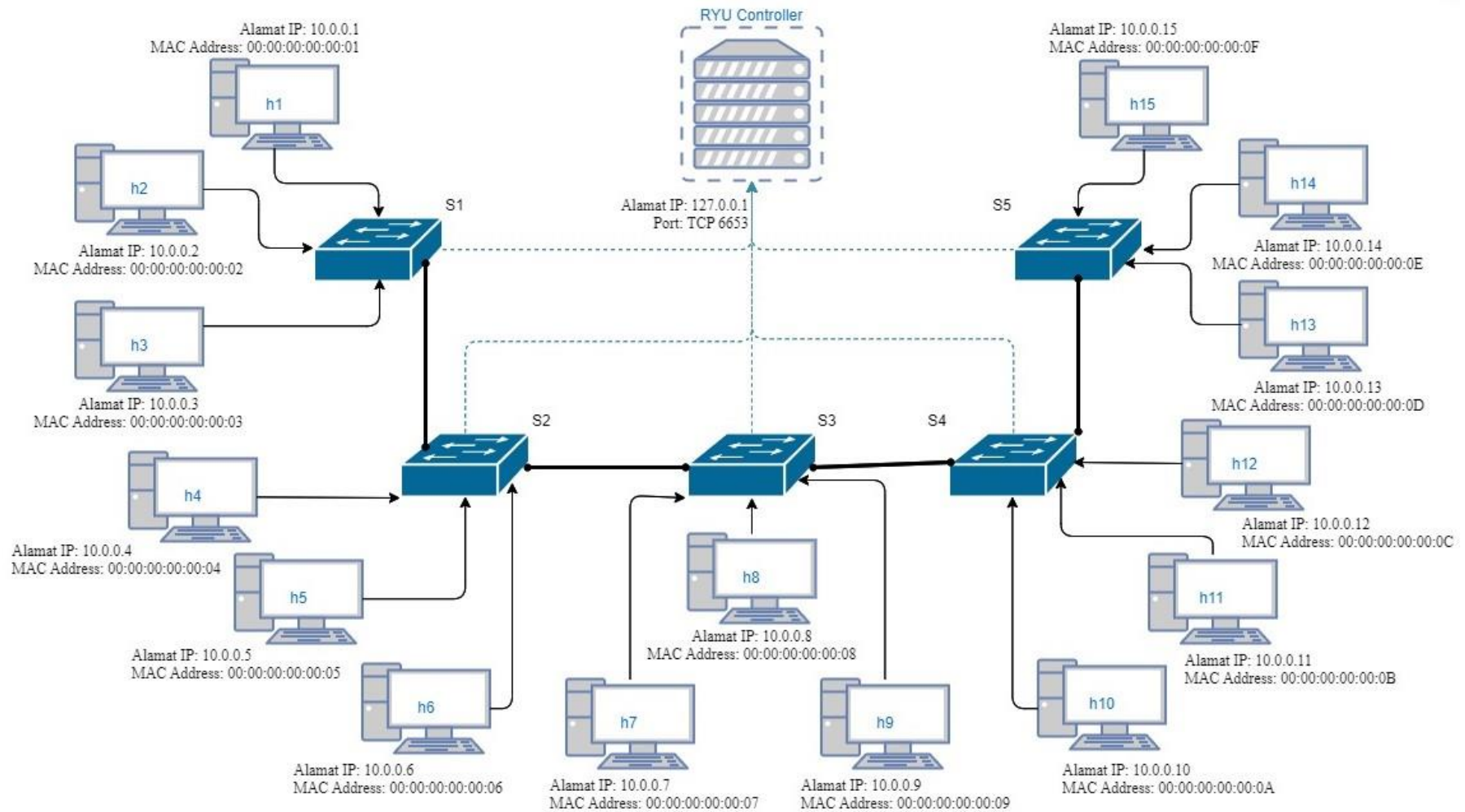
topik yang sama. Dalam hal ini, peneliti dapat mengeksplorasi faktor-faktor lain yang mempengaruhi serangan DDoS, mengevaluasi metode *deep learning* lainnya, atau mempelajari aplikasi lain dari metode SDN.

### 3.3.2 Perancangan Topologi *Software Defined Network* (SDN)

Perancangan topologi SDN dengan Mininet dan Ryu merujuk pada proses merancang atau membuat struktur jaringan menggunakan konsep *Software-Defined Network* (SDN). Dengan melakukan perancangan topologi SDN, peneliti dapat memastikan bahwa jaringan SDN yang diimplementasikan dapat beroperasi dengan efisien, fleksibel, dan mudah untuk di analisa. Perancangan topologi SDN melibatkan merancang dan mengatur struktur jaringan yang memisahkan lapisan kontrol dari lapisan penerusan.

Mininet berfungsi sebagai emulator jaringan untuk membuat topologi jaringan virtual, sementara Ryu digunakan sebagai kerangka kerja perangkat lunak untuk mengembangkan aplikasi kontroler SDN. Topologi terdiri dari 5 *switch* yang dihubungkan secara linear, di mana setiap *switch* terhubung ke 3 *host*, menghasilkan total 15 *host* dalam jaringan. Masing-masing *switch* dihubungkan satu sama lain, membentuk rantai linear yang memungkinkan pengujian efektivitas routing dan pengelolaan aliran data di jaringan. Setiap *host* memiliki alamat IP dan *MAC address* yang berurutan, mulai dari h1 hingga h15.

Pengaturan ini memudahkan manajemen dan pemetaan *host* di dalam jaringan serta membantu dalam proses *troubleshooting*. Setiap *switch* terhubung dengan controller SDN melalui koneksi virtual, dan koneksi ini memanfaatkan protokol *OpenFlow* 1.3, yang memberikan fleksibilitas dalam pengaturan aliran data serta mendukung berbagai fitur canggih. Pemantauan lalu lintas jaringan dilakukan melalui controller yang beroperasi pada *port* 6653, memungkinkan peneliti untuk melihat dan menganalisis aliran data secara *real-time*. Topologi ini dirancang untuk mendukung eksperimen dan pengujian berbagai skenario jaringan, dengan fokus pada pengujian performa jaringan SDN, analisis serangan DDoS, dan pengembangan metode mitigasi yang efektif. Berikut adalah topologi yang digunakan dalam penelitian ini.



Gambar 3.2 Topologi jaringan SDN

Gambar 3.2 menunjukkan topologi jaringan SDN yang digunakan untuk pembuatan *dataset* dan simulasi deteksi serangan DDoS. Lingkungan simulasi jaringan ini dibuat di dalam, sistem operasi Ubuntu 22.4 yang berperan sebagai platform penyedia infrastruktur dan sumber daya untuk menjalankan simulasi. Berikut merupakan tabel konfigurasi jaringan yang di gunakan dalam penelitian.

Tabel 3.2 Tabel konfigurasi SDN

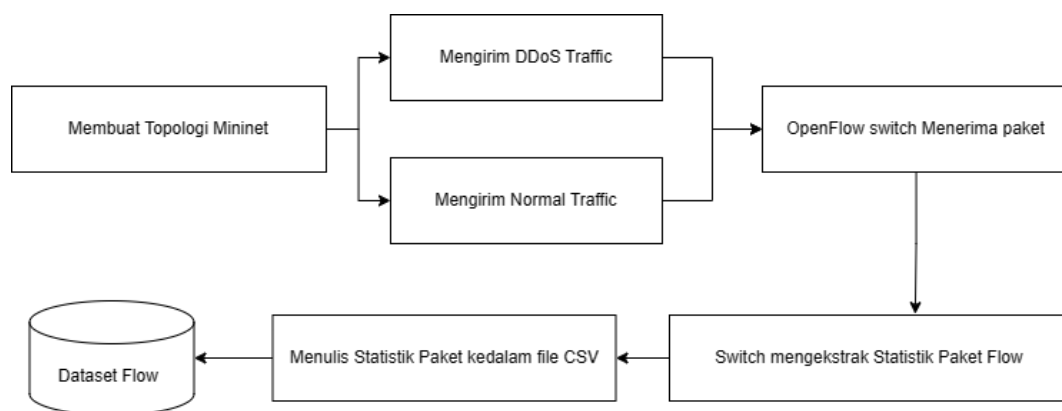
<b>Detail Konfigurasi Node</b>		
<i>Switches:</i>		
1.	s1: Terhubung ke	h1, h2, h3, s2
2.	s2: Terhubung ke	h4, h5, h6, s1, s3
3.	s3: Terhubung ke	h7, h8, h9, s2, s4
4.	s4: Terhubung ke	h10, h11, h12, s3, s5
5.	s5: Terhubung ke	h13, h14, h15, s4
<i>Hosts:</i>		
1.	h1	- IP: 10.0.0.1/24, - MAC: 00:00:00:00:00:01, - Terhubung ke: s1
2.	h2	- IP: 10.0.0.2/24, - MAC: 00:00:00:00:00:02, - Terhubung ke: s1
3.	h3	- IP: 10.0.0.3/24, - MAC: 00:00:00:00:00:03, - Terhubung ke: s1
4.	h4	- IP: 10.0.0.4/24, - MAC: 00:00:00:00:00:04, - Terhubung ke: s2
5.	h5	- IP: 10.0.0.5/24, - MAC: 00:00:00:00:00:05, - Terhubung ke: s2
6.	h6	- IP: 10.0.0.6/24, - MAC: 00:00:00:00:00:06, - Terhubung ke: s2
7.	h7	- IP: 10.0.0.7/24, - MAC: 00:00:00:00:00:07, - Terhubung ke: s3
8.	h8	- IP: 10.0.0.8/24, - MAC: 00:00:00:00:00:08, - Terhubung ke: s3
9.	h9	- IP: 10.0.0.9/24, - MAC: 00:00:00:00:00:09, - Terhubung ke: s3
10.	h10	- IP: 10.0.0.10/24, - MAC: 00:00:00:00:00:0A, - Terhubung ke: s4
11.	h11	- IP: 10.0.0.11/24, - MAC: 00:00:00:00:00:0B, - Terhubung ke: s4
12.	h12	- IP: 10.0.0.12/24,

		- MAC: 00:00:00:00:00:0C, - Terhubung ke: s4
13.	h13	- IP: 10.0.0.13/24, - MAC: 00:00:00:00:00:0D, - Terhubung ke: s5
14.	h14	- IP: 10.0.0.14/24, - MAC: 00:00:00:00:00:0E, - Terhubung ke: s5
15.	h15	- IP: 10.0.0.15/24, - MAC: 00:00:00:00:00:0F, - Terhubung ke: s5
Monitoring dan <i>controller</i> :		
1.	sFlow:	Fungsi <i>wrapper</i> dari <i>sflow.py</i> digunakan untuk monitoring jaringan.
2.	<i>Controller</i>	- IP: 127.0.0.1 - Port: 6653 - Protokol: OpenFlow13 - Menggunakan <i>remote controller</i> di Mininet

Secara keseluruhan, topologi SDN ini menciptakan lingkungan jaringan yang terpusat dan terkontrol dengan fleksibilitas, efisiensi, dan kemampuan adaptasi yang tinggi dalam pengaturan dan manajemen jaringan, serta memberikan fondasi yang kuat untuk penelitian dan eksperimen dalam bidang keamanan jaringan dan deteksi intrusi.

### 3.3.3 Akuisisi Data

Pembuatan *dataset* adalah proses pengumpulan, pengolahan, dan penyusunan data dalam sebuah format yang dapat digunakan untuk analisis. *Dataset* yang baik harus memiliki keragaman data yang mencakup berbagai variabel dan kategori. Berikut merupakan proses pembuatan *dataset*.



Gambar 3.3 Blok diagram pembuatan *dataset*

Gambar 3.3 menggambarkan blok diagram proses yang diterapkan dalam langkah-langkah pembuatan *dataset*. Untuk membuat *dataset* pada jaringan SDN, langkah-langkah berikut perlu dilakukan. Pertama, kita perlu membuat topologi jaringan yang diinginkan menggunakan Mininet, di mana jaringan virtual dibuat sesuai dengan kebutuhan. Setelah itu, pada langkah kedua, dilakukan pengiriman paket data normal dan DDoS ke jaringan SDN dengan menggunakan alat seperti, *ping*, *iperf*, dan *hping*. Langkah ketiga melibatkan *openflow switch* sebagai penerima paket data dalam jaringan tersebut. Selanjutnya, pada langkah keempat, *switch* melakukan ekstraksi statistik dari aliran paket yang diterima. Hasil ekstraksi statistik tersebut, disimpan oleh sistem ke dalam file berformat CSV, membentuk *datasetflow.csv* yang nantinya dapat digunakan untuk analisis lebih lanjut terkait aliran data dalam jaringan. Cara diatas juga di gunakan sebagai bahan pengumpulan *dataset* yang akan di gunakan sebagai bahan prediksi untuk sistem pendeteksi serangan DDoS.

Penelitian ini menggunakan *dataset* yang terdiri dari 22 fitur yang umumnya digunakan untuk menggambarkan aliran data dalam jaringan yang menggunakan protokol *OpenFlow*. *Dataset* ini memberikan informasi yang cukup lengkap dan mendalam tentang karakteristik aliran data yang berjalan melalui jaringan. Dengan menggunakan *dataset* ini, penelitian dapat memberikan pemahaman yang lebih komprehensif tentang penggunaan protokol *OpenFlow* dalam mengelola aliran data dalam jaringan. Penjelasan dari masing - masing fitur di gambarkan pada tabel 3.2 berikut ini.

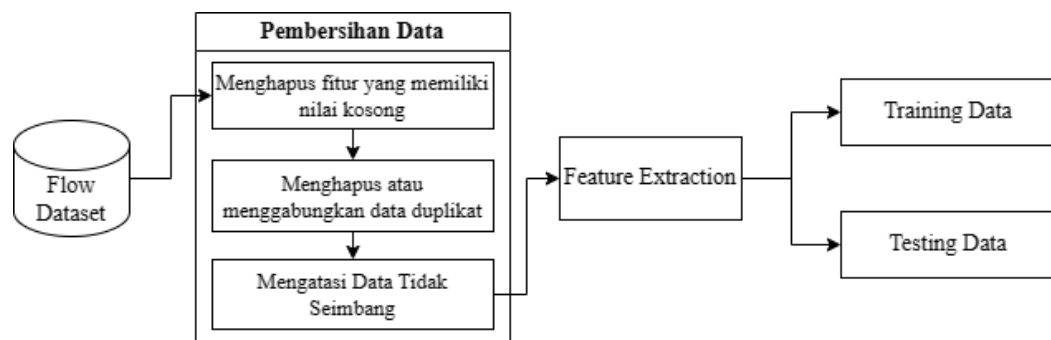
Tabel 3.3 Tabel parameter lingkungan simulasi

No	Fitur pada <i>dataset</i>	Deskripsi
1	Timestamp	Waktu saat aliran data dimulai atau diakhiri
2	<i>datapath_id</i>	Pengenal unik dari <i>switch OpenFlow</i> yang terlibat dalam aliran data
3	<i>flow_id</i>	Pengenal unik dari aliran data itu sendiri
4	<i>ip_src</i>	Alamat IP Sumber
5	<i>tp_src</i>	<i>Port</i> sumber
6	<i>ip_dst</i>	Alamat IP tujuan
7	<i>tp_dst</i>	<i>Port</i> tujuan
8	<i>ip_proto</i>	Protokol IP (TCP, UDP, ICMP)
9	<i>icmp_code</i>	Kode ICMP yang digunakan oleh paket data dalam aliran
10	<i>icmp_type</i>	Tipe ICMP yang digunakan

11	<i>flow_duration_sec</i>	Durasi aliran data dalam detik
12	<i>flow_duration_nsec</i>	Durasi aliran data dalam <i>nanodetik</i>
13	<i>idle_timeout</i>	Waktu maksimum aliran data tidak aktif sebelum dihapus
14	<i>hard_timeout</i>	Waktu maksimum aliran data tetap ada sebelum dihapus
15	<i>flags</i>	Tanda status atau sifat aliran data
16	<i>packet_count</i>	Jumlah paket data dalam aliran
17	<i>byte_count</i>	Jumlah byte data dalam aliran
18	<i>packet count per second</i>	Rata-rata jumlah paket data per detik
19	<i>packet count per nsecond</i>	Rata-rata jumlah paket data per <i>nanodetik</i>
20	<i>byte count per second</i>	Rata-rata jumlah byte data per detik
21	<i>byte count per nsecond</i>	Rata-rata jumlah byte data per <i>nanodetik</i>

### 3.3.4 Pra-pemrosesan data

Tujuan dari pra-pemrosesan data adalah untuk mengubah data mentah menjadi bentuk yang berguna dan mudah dalam analisis. Berikut adalah gambaran alur dari pra-pemrosesan data.



Gambar 3.4 Alur pra-pemrosesan data

Gambar 3.4 merupakan *flowchart* prosedur pra-pemrosesan dataset dalam konteks jaringan lalu lintas Software-Defined Network (SDN). Langkah pertama adalah menghilangkan fitur yang memiliki nilai kosong, yang penting untuk menghindari bias dalam analisis karena informasi yang tidak tercatat atau tidak tersedia. Selanjutnya, data duplikat dihapus atau digabungkan untuk mencegah distorsi dalam pola yang dianalisis, mengingat bahwa dalam lalu lintas jaringan, data duplikat mungkin terjadi akibat pengumpulan informasi yang sama berulang kali. Langkah ketiga adalah menangani data yang tidak seimbang, ini sering terjadi ketika sampel dari kelas berbeda tidak merata, seperti lebih banyak contoh lalu lintas normal dibandingkan serangan jaringan, yang dapat mengganggu model pembelajaran mesin dalam mendeteksi serangan. Untuk mengatasi ini, diterapkan

teknik seperti *oversampling* atau *undersampling*. Berikutnya, fitur diekstraksi dari data mentah untuk mengidentifikasi pola lalu lintas normal dan anomali, seperti durasi aliran rata-rata atau ukuran paket. Setelah itu, 80% dari dataset yang telah diproses digunakan sebagai data latihan untuk model pembelajaran mesin, sedangkan 20% sisanya disimpan sebagai data pengujian untuk mengevaluasi kinerja model terhadap data baru. Seluruh proses ini esensial untuk memastikan bahwa model yang dilatih memiliki kemampuan untuk memprediksi dan menggeneralisasi dengan baik saat diterapkan dalam skenario nyata.

### 3.3.5 Perancangan Algoritma

Perancangan *deep learning* dengan arsitektur *Multi Layer Perceptron* (MLP) untuk mengklasifikasikan serangan DDoS merupakan langkah awal dalam merancang suatu sistem yang dapat secara efektif membedakan dan mengidentifikasi pola serangan yang mungkin terjadi pada jaringan. Perancangan algoritma atau arsitektur MLP ini memainkan peran kunci dalam memberikan kecerdasan buatan yang dapat mengenali tanda-tanda dan perilaku karakteristik serangan DDoS sehingga memungkinkan sistem untuk memberikan respon yang cepat dan efisien dalam melindungi integritas dan ketersediaan layanan jaringan.

Tabel 3.4 Tabel arsitektur MLP

Models	Hyperparameters
MLP	Layer : 2 layer (256 dan 128 neuron)
	Activation : relu
	Regulasi L2 : 0.0001
	Random state : 42
	Solver : adam
	Learning rate : 0.001

Tabel 3.3 merupakan rancangan arsitektur MLP yang digunakan dalam penelitian ini. Dalam rancangan arsitektur MLP ini, terdapat dua lapisan tersembunyi dengan jumlah *neuron* yang ditentukan. Lapisan tersembunyi bertugas untuk mengekstraksi fitur dari data masukan, sedangkan lapisan keluaran bertugas untuk menghasilkan prediksi atau klasifikasi dari data. Pada setiap *neuron* dalam MLP, digunakan *rectified linear unit* untuk memperoleh hasil yang lebih baik.

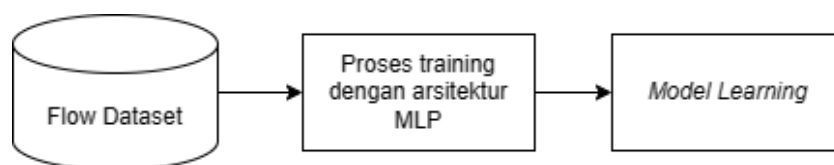
Untuk menghindari *overfitting*, digunakan regulasi L2 dengan nilai 0.0001. Regulasi L2 adalah teknik yang digunakan untuk mengurangi kompleksitas model

dengan menghukum bobot parameter yang besar dan kompleks pada model. Selain itu, rancangan ini menggunakan algoritma optimasi adam dengan laju pembelajaran sebesar 0.001. Algoritma optimasi adam adalah algoritma yang adaptif dan dapat meningkatkan kinerja MLP dengan mengatasi masalah gradien yang meledak atau stagnan.

Rancangan arsitektur MLP ini memiliki spesifikasi yang dapat diimplementasikan dalam penelitian untuk mengklasifikasikan serangan DDoS. Dengan menggunakan rancangan ini, diharapkan dapat memperoleh hasil yang akurat dan efektif dalam melindungi integritas dan ketersediaan layanan jaringan.

### 3.3.6 Training Dataset

Berikut adalah gambaran alur pelatihan yang bertujuan untuk melatih model agar mampu membedakan antara lalu lintas jaringan normal dan serangan DDoS.



Gambar 3.5 Alur proses training data

Gambar 3.5 yang terlampir di bawah ini menjelaskan secara rinci alur dari proses pelatihan model. Pada tahap awal, digunakanlah *dataset* yang berisi data jaringan yang telah diekstrak dari jaringan SDN. *Dataset* ini mencakup berbagai jenis serangan DDoS dan juga lalu lintas normal. Proses pelatihan dimulai dengan melakukan impor *dataset* yang telah disiapkan sebelumnya. Selanjutnya, algoritma *Multilayer Perceptron* (MLP) digunakan untuk melatih model yang telah disiapkan sebelumnya. Proses pelatihan dilakukan dalam beberapa iterasi, dimana setiap iterasi bertujuan untuk meningkatkan akurasi model yang sedang dilatih. Selama proses pelatihan berlangsung, model yang sedang dilatih dievaluasi secara berkala untuk memastikan bahwa kinerjanya sesuai dengan harapan.

### 3.3.7 Uji Keandalan Model

Setelah model berhasil dilatih, langkah selanjutnya adalah mengevaluasi performa dari model yang telah dibuat. Evaluasi ini dilakukan untuk mengetahui seberapa akurat dan handal model dalam membedakan antara serangan DDoS dan



lalu lintas jaringan normal. Evaluasi model dilakukan dengan data yang belum digunakan pada proses *training*.

Setelah metrik evaluasi dihitung, hasilnya dianalisis untuk mendapatkan kinerja model. Akurasi memberikan gambaran umum tentang seberapa sering model membuat prediksi yang benar. Presisi dan *recall* menunjukkan seberapa banyak prediksi serangan yang benar-benar serangan, sementara *recall* menunjukkan seberapa banyak serangan yang berhasil dideteksi dari semua serangan yang ada. Skor F1, kombinasi dari presisi dan *recall*, memberikan metrik tunggal untuk mengevaluasi model, terutama ketika ada *trade-off* antara presisi dan *recall*.

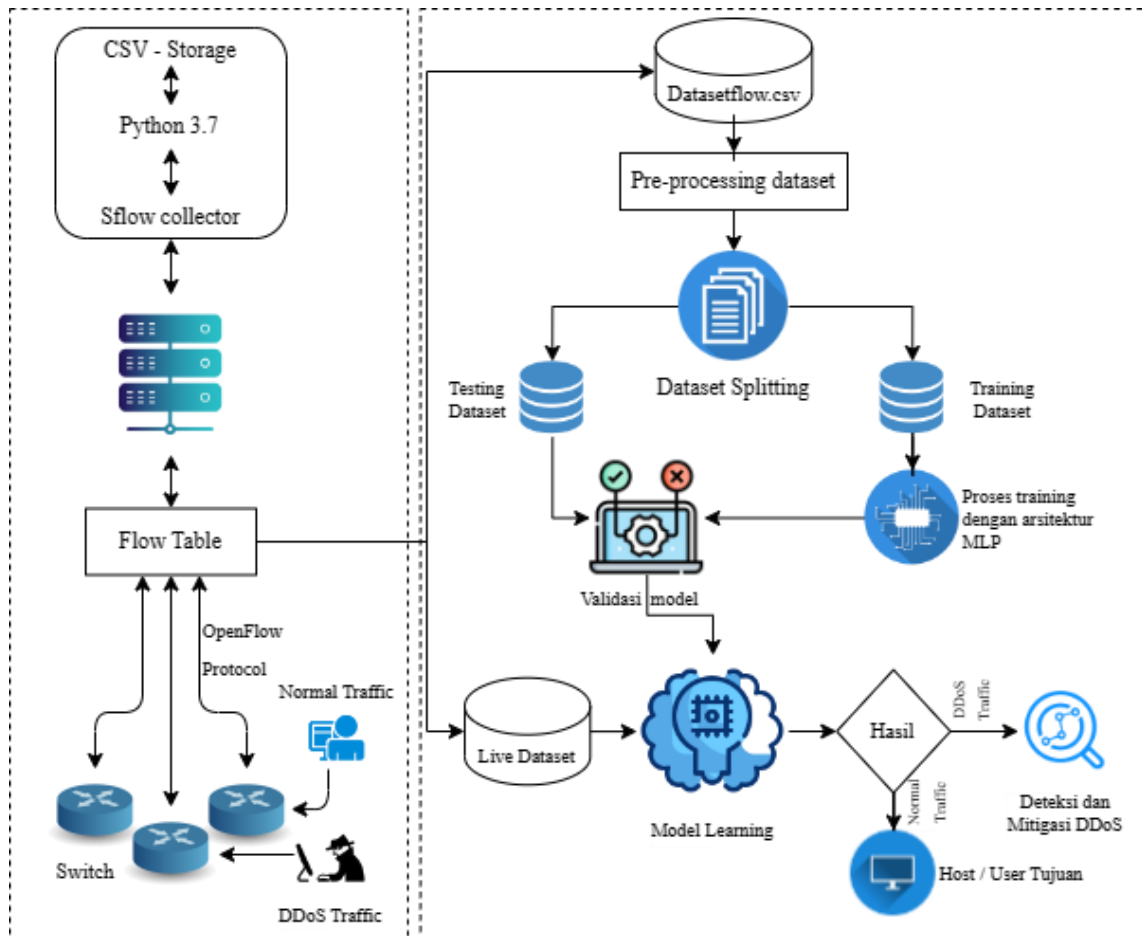
### 3.3.8 Analisis *Quality of Service* (QoS)

Analisis QoS dilakukan dengan menggunakan standar TIPHON untuk mengukur kualitas jaringan. Parameter yang di gunakan untuk melakukan analisis QoS yaitu *throughput*, *delay*, *jitter*, dan *packet loss*. Parameter ini kemudian akan di ukur dan dibandingkan dengan standar TIPHON untuk menentukan kategori kualitas jaringan, apakah masuk dalam kategori "Sangat Bagus", "Bagus", "Sedang", atau "Buruk". Analisis ini dilakukan baik saat jaringan berjalan normal, selama serangan DDoS, maupun setelah mitigasi. Oleh karena itu analisis ini dapat digunakan untuk membantu mengevaluasi dampak serangan DDoS terhadap performa jaringan dan efektivitas model mitigasi yang diterapkan.

Dalam penelitian ini, *iPerf* digunakan untuk mengukur *throughput* jaringan, *Ping* untuk mengukur *delay* dan *packet loss*, Wireshark untuk menganalisis lalu lintas jaringan secara detail, dan sFlow untuk memonitor lalu lintas serta menghitung metrik *resource utilization*.

Dengan membandingkan hasil pengukuran QoS terhadap standar TIPHON, dapat ditentukan apakah tindakan mitigasi yang dilakukan berhasil meningkatkan kualitas layanan jaringan. Analisis ini tidak hanya memberikan gambaran tentang kondisi jaringan pasca-mitigasi, tetapi juga membantu dalam pengambilan keputusan lebih lanjut untuk meningkatkan performa jaringan, mengurangi latensi, dan meminimalkan kehilangan data. Evaluasi ini sangat penting untuk memastikan bahwa jaringan dapat terus berfungsi secara optimal, bahkan di bawah ancaman serangan DDoS.

### 3.4. Perancangan Sistem



Gambar 3.6 Perancangan sistem

Sistem deteksi serangan DDoS pada jaringan SDN menggunakan algoritma MLP terdiri dari dua proses utama, yaitu proses pembelajaran dan proses pendeteksian. Proses pembelajaran dilakukan dengan menggunakan *dataset* yang berisi *flow* data jaringan normal dan DDoS yang dikumpulkan dari topologi jaringan SDN. *Dataset* tersebut kemudian digunakan sebagai data masukan untuk algoritma MLP yang digunakan untuk mengenali pola dari *flow* data jaringan.

Setelah proses pembelajaran selesai, model yang dihasilkan oleh algoritma MLP disimpan pada *controller* SDN. Proses pendeteksian dilakukan dengan menggunakan data *realtime* yang berasal dari topologi jaringan SDN yang sedang beroperasi. Data *realtime* tersebut akan diteruskan ke SDN untuk dilakukan klasifikasi oleh model yang telah dibangun sebelumnya. Jika *flow* data yang masuk kedalam *controller* SDN terdeteksi sebagai DDoS, maka *controller* SDN akan melakukan mitigasi dan memberikan peringatan kepada administrator jaringan.

## BAB IV

### HASIL DAN PEMBAHASAN

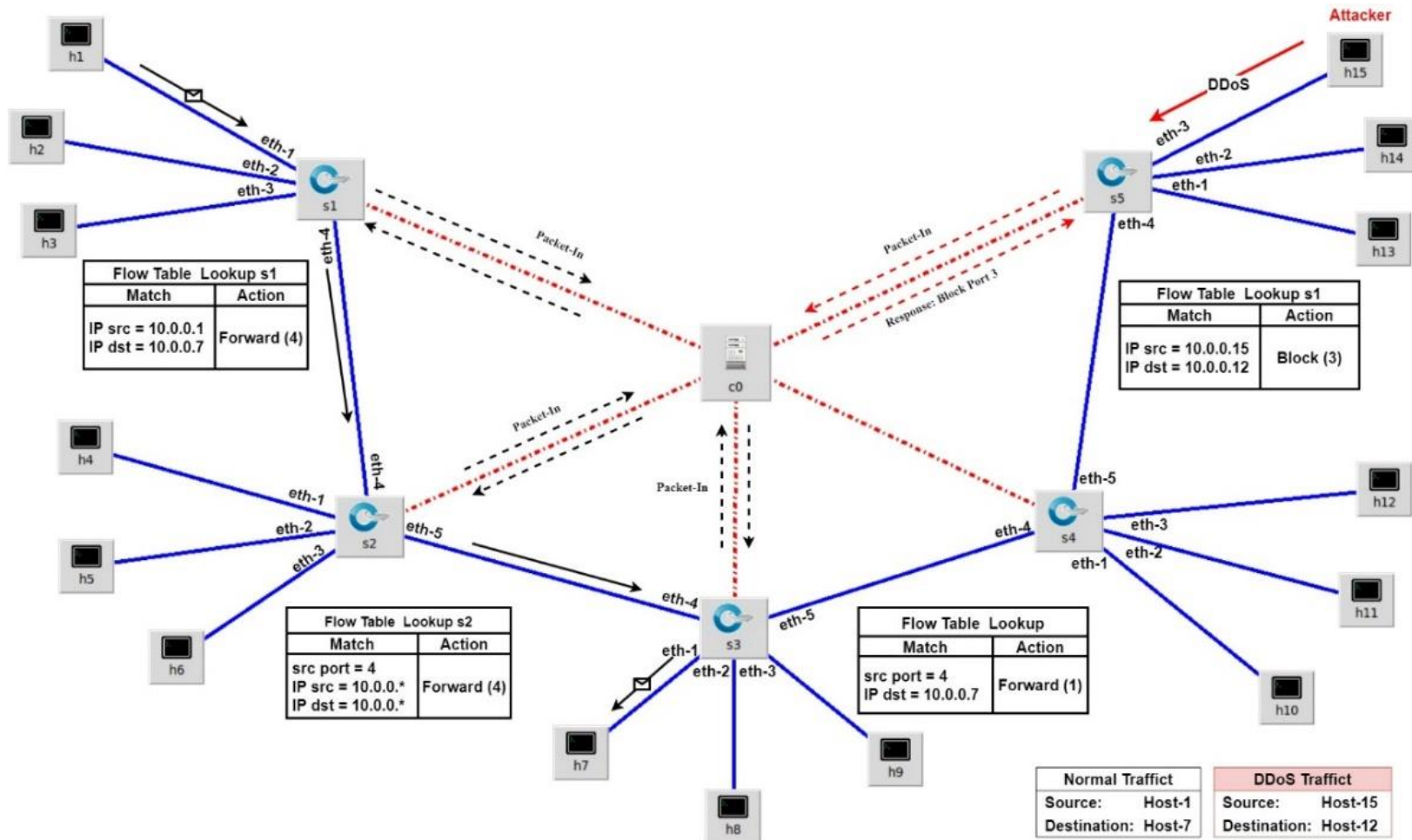
Pada bab ini, membahas mengenai analisis dan hasil dari penelitian yang dilakukan untuk mendeteksi dan memitigasi serangan *Distributed Denial of Service* (DDoS) pada jaringan *Software Defined Network* (SDN) menggunakan algoritma *Multilayer Perceptron* (MLP). Penelitian ini mencakup eksperimen yang dilakukan menggunakan Mininet sebagai emulator jaringan, Ryu *controller* sebagai pengendali jaringan, dan sFlow RT sebagai alat pemantau lalu lintas jaringan. Analisis dilakukan dengan melihat performa jaringan melalui pengukuran *delay*, *throughput*, *packet loss* dan sumber daya sistem.

#### 4.1. Analisis Jaringan

Topologi pada penelitian ini mencakup berbagai *node* dan *switch* sehingga dapat meniru skenario jaringan yang kompleks. Jaringan yang digunakan dalam penelitian ini adalah *traffic generator*, yang memungkinkan pengukuran nilai *delay*, *throughput*, dan *packet loss*. Berikut adalah gambar topologi dalam simulasi jaringan yang digunakan

*Traffic generator* digunakan untuk menghasilkan lalu lintas jaringan yang bervariasi sehingga dapat mensimulasikan berbagai jenis serangan DDoS. Berdasarkan hasil *traffic generator* tersebut dapat peneliti dapat mengukur respon jaringan terhadap serangan tersebut. Pengukuran performa jaringan ini penting untuk mengevaluasi seberapa efektif algoritma MLP dalam mendeteksi dan memitigasi serangan DDoS. Setiap metrik pengukuran dapat memberikan gambaran tentang bagaimana serangan mempengaruhi kinerja jaringan dan seberapa baik mitigasi yang diterapkan dapat memperbaiki performa jaringan.

Selain itu, topologi jaringan yang digunakan juga dirancang agar cukup fleksibel untuk diuji dengan berbagai skenario serangan, baik serangan yang bersifat volumetrik maupun serangan yang lebih canggih, seperti serangan berbasis protokol. Fleksibilitas ini memungkinkan peneliti untuk menilai efektivitas mitigasi dalam berbagai kondisi jaringan yang berbeda. Berikut adalah gambar topologi dalam simulasi jaringan yang digunakan.



Gambar 4.1 Topologi jaringan linear

Gambar 4.1 menunjukkan simulasi topologi jaringan SDN yang digunakan dalam penelitian ini. Terdapat tiga komponen utama dalam pembentukan jaringan SDN yaitu *host*, *OVS Switch*, dan *controller*. Setiap *switch* dikendalikan oleh *controller remote* yang berjalan pada alamat 127.0.0.1 di *port* 6653. *Switch* yang digunakan dalam penelitian ini berfungsi sebagai pengatur lalu lintas data dan pembuat aturan *flow* menggunakan *Ryu controller* dengan protokol *OpenFlow* 1.3.

Setiap *host* diatur dengan alamat IP dan MAC yang spesifik. Misalnya, *host* h1 memiliki IP 10.0.0.1/24 dan MAC address 00:00:00:00:00:01, sedangkan *host* h2 memiliki IP 10.0.0.2/24 dan MAC address 00:00:00:00:00:02. Pengaturan ini memungkinkan pengujian lalu lintas data yang terstruktur dan terkontrol antara *host-host* dalam jaringan. Berikut merupakan gambar *port* yang terhubung di masing-masing *node* pada topologi.

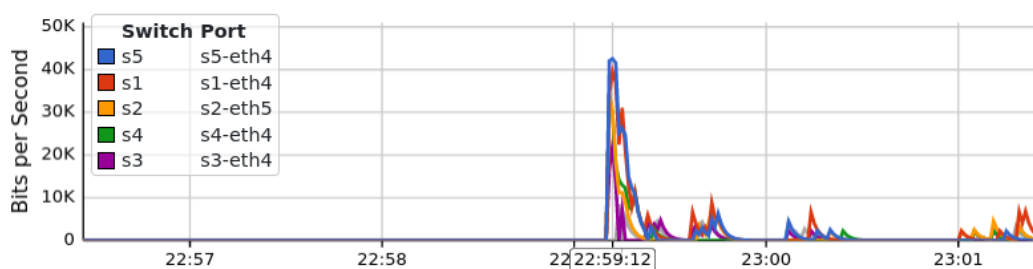
```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s2-eth1
h5 h5-eth0:s2-eth2
h6 h6-eth0:s2-eth3
h7 h7-eth0:s3-eth1
h8 h8-eth0:s3-eth2
h9 h9-eth0:s3-eth3
h10 h10-eth0:s4-eth1
h11 h11-eth0:s4-eth2
h12 h12-eth0:s4-eth3
h13 h13-eth0:s5-eth1
h14 h14-eth0:s5-eth2
h15 h15-eth0:s5-eth3
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0 s1-eth4:s2-eth4
s2 lo: s2-eth1:h4-eth0 s2-eth2:h5-eth0 s2-eth3:h6-eth0 s2-eth4:s1-eth4 s2-eth5:
s3-eth4
s3 lo: s3-eth1:h7-eth0 s3-eth2:h8-eth0 s3-eth3:h9-eth0 s3-eth4:s2-eth5 s3-eth5:
s4-eth4
s4 lo: s4-eth1:h10-eth0 s4-eth2:h11-eth0 s4-eth3:h12-eth0 s4-eth4:s3-eth5 s4-eth5:
h5:s5-eth4
```

Gambar 4.2 Koneksi antara *host* dan *switch*

Gambar 4.2 adalah hasil dari perintah `net` pada terminal Mininet yang menunjukkan topologi jaringan dan koneksi antara *host* dan *switch*. Dalam topologi ini, ada lima *switch* yaitu s1 hingga s5 dan lima belas *host* yaitu h1 hingga h15. Setiap *host* terhubung ke sebuah *switch* melalui *port* tertentu. Misalnya, *host* h1 terhubung ke *switch* s1 pada *port* s1-eth1, *host* h2 ke s1 pada *port* s1-eth2, dan seterusnya. Selain itu, *switch* juga saling terhubung satu sama lain, membentuk jaringan yang lebih kompleks. *Switch* s1 terhubung ke *switch* s2 dan s4, *switch* s2 terhubung ke s1 dan s3, *switch* s3 terhubung ke s2 dan s4, *switch* s4 terhubung ke

s3 dan s5, dan *switch* s5 terhubung ke s4. Topologi ini memungkinkan adanya beberapa jalur antara *host*, yang meningkatkan redundansi dan ketahanan jaringan.

Saat jaringan pertama kali terhubung, *switch* dikendalikan oleh aplikasi `SimpleSwitch13` yang berjalan pada *Ryu controller*. *Switch* diinisialisasi untuk mengirimkan semua paket yang tidak dikenali ke *controller* melalui aturan default yang ditetapkan pada event `EventOFPSwitchFeatures`. Ketika paket diterima oleh *switch*, handler `EventOFPPacketIn` mempelajari alamat MAC sumber dan memetakan *port*-nya, serta menentukan *port* tujuan berdasarkan alamat MAC tujuan. Jika alamat tujuan diketahui, *flow* baru diinstal untuk menangani lalu lintas serupa di masa depan tanpa perlu mengirimkan paket ke *controller* lagi sehingga meningkatkan efisiensi jaringan. *Flow* diinstal dengan mempertimbangkan protokol yang berbeda seperti ICMP, TCP, dan UDP. Berikut merupakan gambar aktivitas *port* ketika *switch* pertama kali terhubung kedalam jaringan.



Gambar 4.3 Aktivitas *port* saat pertama kali terhubung

Gambar 4.3 menunjukkan aktivitas *port* pada *switch* di jaringan saat pertama kali terhubung. Grafik tersebut memvisualisasikan bit per detik untuk berbagai *port* pada *switch* yang berbeda. Pada awal koneksi, terlihat adanya lonjakan aktivitas lalu lintas jaringan yang signifikan. Ini menandakan bahwa ada volume data yang tinggi melewati *port-port* tersebut ketika jaringan mulai beroperasi. Lonjakan ini disebabkan oleh proses inisialisasi dan pengaturan *flow* pada *switch* oleh *Ryu controller*. Setelah lonjakan awal, aktivitas lalu lintas cenderung menurun dan stabil dengan beberapa fluktuasi kecil, yang menunjukkan bahwa jaringan mulai beroperasi dalam kondisi normal setelah proses inisialisasi selesai.

Selanjutnya fungsi *pingall* digunakan untuk melakukan pengecekan konektivitas antar seluruh *host* yang ada dalam jaringan. Cara kerja *pingall* yaitu dengan mendapatkan daftar semua *host* dan kemudian mengirimkan paket *Internet Control Message Protocol* (ICMP) dari satu *host* ke *host* lainnya untuk memastikan bahwa *host* tujuan dapat menerima dan merespons paket tersebut. Berikut merupakan hasil pada terminal mininet.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10 h11 h12 h13 h14 h15
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10 h11 h12 h13 h14 h15
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12 h13 h14 h15
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12 h13 h14 h15
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12 h13 h14 h15
h11 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12 h13 h14 h15
h12 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h13 h14 h15
h13 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h14 h15
h14 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h15
h15 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14
*** Results: 0% dropped (210/210 received)
```

Gambar 4.4 Hasil pengujian *pingall* pada terminal mininet

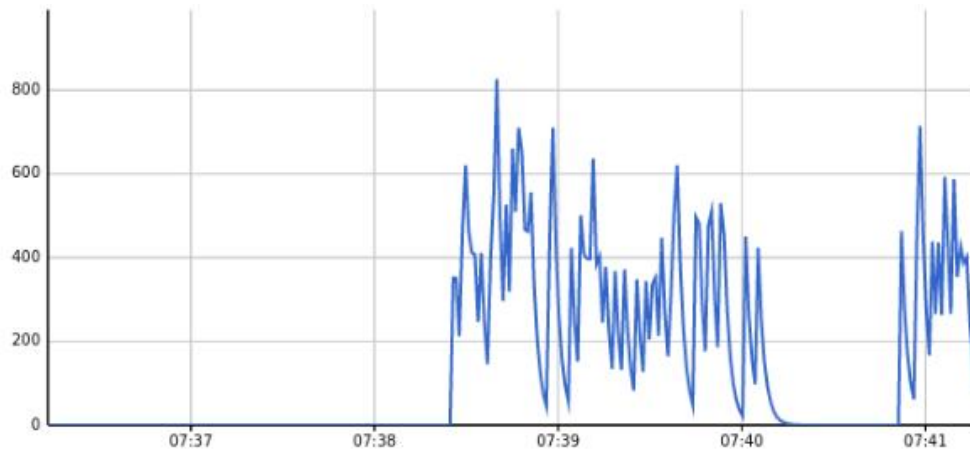
Gambar 4.4 menunjukkan hasil dari pengujian *pingall*, di mana setiap *host* mengirimkan paket ICMP ke semua *host* lainnya. Hasil ini memastikan bahwa semua *host* dan *switch* berfungsi dengan baik dalam jaringan. Nilai *drop packet* sebesar 0% menandakan tidak ada masalah konektivitas, seperti *host* yang tidak dapat dijangkau atau adanya *delay* tinggi dalam pengiriman paket. Dengan hasil ini, dapat dipastikan bahwa topologi jaringan telah terkonfigurasi dengan benar dan setiap perangkat dalam jaringan dapat berkomunikasi secara efektif tanpa ada hambatan.

## 4.2. Analisis Hasil Deteksi dan Mitigasi

### 4.2.1 Lalu lintas Normal

Pada kondisi lalu lintas normal, model *Multilayer Perceptron* (MLP) digunakan untuk memantau dan mendeteksi anomali dalam jaringan. Untuk

mengecek respon sistem terhadap lalu lintas sah digunakan perintah *ping*. Berikut merupakan hasil monitoring sistem ketika di lalui lalu lintas normal.



Gambar 4.5 Hasil monitoring lalu lintas normal

Gambar 4.5 merupakan grafik jumlah aliran data yang mengalir pada sistem dengan satuan Bits/s. Gambar di atas menunjukkan karakteristik lalu lintas normal dengan *traffic generator* yaitu *ping*, menghasilkan lalu lintas data sebanyak 800 Bits/s. Pada gambar di atas juga terlihat bahwa volume trafik stabil dan konsisten tanpa adanya lonjakan yang signifikan. Berikut merupakan hasil deteksi pada saat dilewati oleh lalu lintas normal atau lalu lintas sah.

```
Traffic is mostly Normal.
-----
Traffic Analysis:
Normal traffic count: 264
DDoS traffic count: 0
Total traffic analyzed: 264
Traffic is mostly Normal.
-----
Traffic Analysis:
Normal traffic count: 2
DDoS traffic count: 0
Total traffic analyzed: 2
Traffic is mostly Normal.
-----
Traffic Analysis:
Normal traffic count: 2
DDoS traffic count: 0
Total traffic analyzed: 2
Traffic is mostly Normal.
-----
```

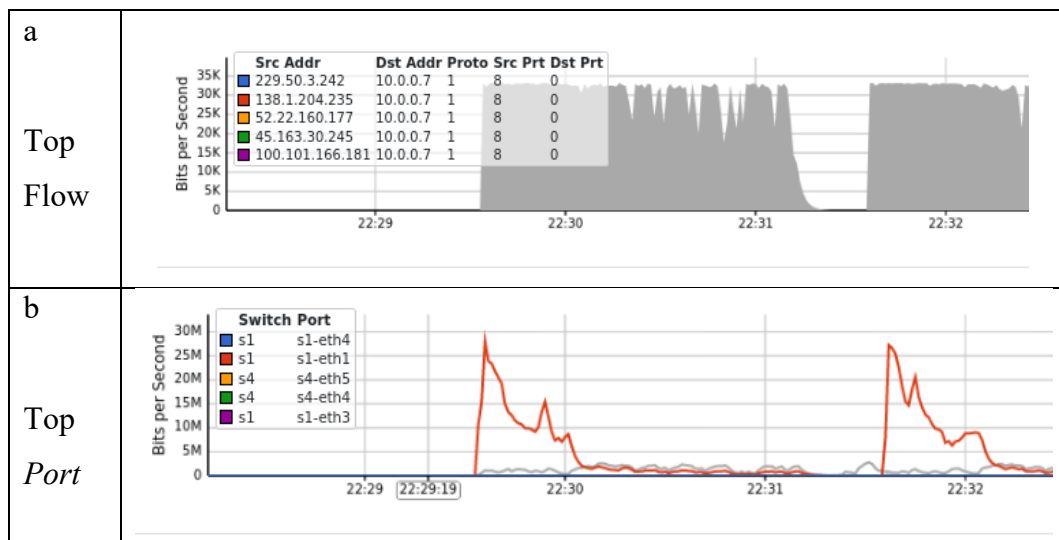
Gambar 4.6 Hasil deteksi lalu lintas normal



Berdasarkan Gambar 4.6, hasil deteksi menunjukkan bahwa MLP mampu dengan efektif mengenali pola lalu lintas yang normal dan stabil, tanpa memberikan *false positives*. Ini menunjukkan bahwa model MLP terlatih dengan baik untuk membedakan antara lalu lintas yang wajar dan pola yang mungkin mencurigakan.

#### 4.2.2 ICMP flood

Dalam eksperimen ini, serangan ICMP *flood* diinisiasi menggunakan perintah *hping3* untuk mengirimkan paket ICMP secara masif ke target. Perintah *hping3* dilakukan dari *host 1* menuju *host 7* dengan mengirimkan alamat IP sumber yang acak untuk setiap paket yang dikirim. Berikut merupakan hasil monitoring jaringan pada saat sistem di lewati oleh hasil simulasi ICMP *flood*.



Gambar 4.7 Hasil monitoring lalu lintas ICMP *flood*, (a) Hasil analisis flow saat ICMP *flood*, (b) Hasil analisis port saat ICMP *flood*

Gambar 4.7 menunjukkan karakteristik lalu lintas ICMP *flood* dalam hal jumlah bits setiap *flow* dan jumlah bits di setiap *port*. Pada grafik *Top Flows*, terlihat bahwa terdapat lonjakan volume trafik yang sangat tinggi mencapai 30 juta Bits/s, menandakan adanya serangan ICMP *flood* dengan bits per second yang meningkat tajam hingga mencapai puncaknya dengan tujuan menghabiskan *bandwidth* dan sumber daya jaringan. Adapun grafik *Top Ports* memperlihatkan aktivitas *port* selama serangan, dengan peningkatan signifikan pada jumlah bits per second di *port* S1 yang merupakan sumber ICMP *flood*, mencerminkan dampak serangan pada *throughput* jaringan. Serangan ICMP *flood* sering menggunakan sumber lalu lintas

yang acak, yaitu berbagai alamat IP sumber yang berbeda-beda, untuk menghindari deteksi dan pemblokiran mudah. Ini ditunjukkan dalam grafik *top flows* di mana terdapat beberapa alamat IP yang berkontribusi pada volume trafik yang tinggi. Respon sistem terhadap *traffic ICMP flood* ditunjukkan pada gambar 4.8.

```
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
-----
Traffic Analysis:
Legitimate Traffic = 0
DoS Traffic = 470
Total Traffic analyzed: 470
DDoS traffic detected.
Victim Host: h7
Mitigation process in progress!
-----
```

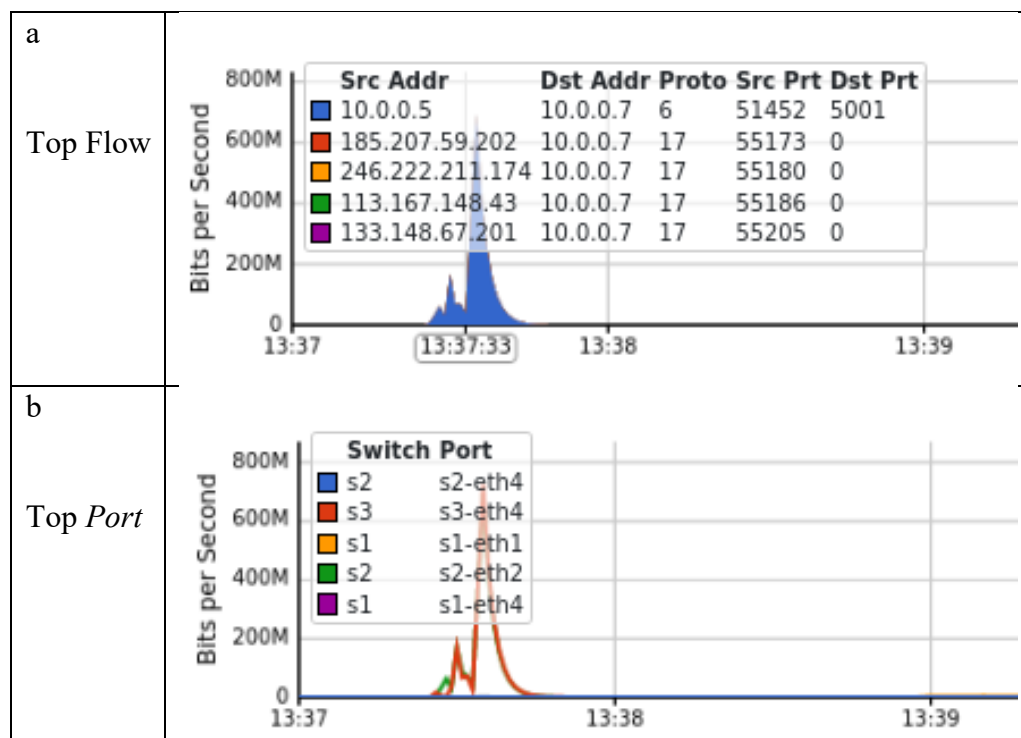
Gambar 4.8 Hasil deteksi dan mitigasi lalu lintas ICMP *flood*

Gambar 4.8 Merupakan hasil prediksi yang ditampilkan pada terminal, MLP berhasil mendeteksi serangan ICMP *flood* dengan akurasi yang tinggi. Hasil analisis menunjukkan bahwa tidak ada lalu lintas normal yang terdeteksi selama serangan. Sebagai gantinya, semua paket yang dianalisis diklasifikasikan sebagai lalu lintas DDoS. Selain itu, MLP berhasil mengidentifikasi *host* yang menjadi potensi korban serangan, yaitu h7, dalam setiap analisis. Ini menunjukkan bahwa model tidak hanya mampu mendeteksi serangan tetapi juga mengidentifikasi target yang diserang.

Berdasarkan hasil deteksi, sistem kemudian melakukan mitigasi terhadap paket ICMP *flood* dengan cara memblokir *port* sumber serangan. Pada gambar 4.8, menunjukkan *port* yang di blokir yaitu *port* 1 pada *switch* 1. Hal ini sesuai dengan perintah *traffic generator* yang menjalankan *hping3* dari *host* 1 menuju *host* 7.

### 4.2.3 UDP flood

Dalam eksperimen ini, serangan UDP *flood* dicirikan oleh pengiriman jumlah besar paket UDP ke *port* acak, yang dilakukan menggunakan *traffic generator* yaitu *hping3* yang dijalankan dari *host* 1 menuju *host* 7. Berikut merupakan karakteristik lalu lintas UDP *flood* saat melawati sistem deteksi DDoS pada jaringan SDN.



Gambar 4.9 Hasil monitoring lalu lintas UDP *flood*, (a) Hasil analisis flow saat UDP *flood*, (b) Hasil analisis *port* saat UDP *flood*

Gambar 4.9, menampilkan aktivitas *flow* UDP *flood* yang mengalir pada sistem. Pada grafik *Top Flows*, terlihat bahwa terdapat lonjakan volume trafik pada topologi jaringan yang mencapai 700 juta Bits/s, menandakan adanya serangan UDP *flood* dengan volume yang sangat tinggi. Tidak seperti TCP, UDP adalah protokol tanpa koneksi yang tidak memerlukan *handshake* tiga arah sehingga penyerang dapat mengirim paket dengan kecepatan tinggi tanpa menunggu respons dari target. Pada grafik *Top Port* juga mencatat aliran data yang mengalir pada setiap *port* di masing-masing *switch*. Aliran data tertinggi tercatat pada *switch* 3 dengan *port* ethernet 4. Trafik jaringan dengan jumlah yang besar ini mampu di deteksi sistem yang di tunjukan pada gambar 4.10

```

Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
-----
Traffic Analysis:
Legitimate Traffic = 0
DoS Traffic = 526
Total Traffic analyzed: 526
DDoS traffic detected.
Victim Host: h7
Mitigation process in progress!
-----
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1

```

Gambar 4.10 Hasil deteksi dan mitigasi lalu lintas UDP *flood*

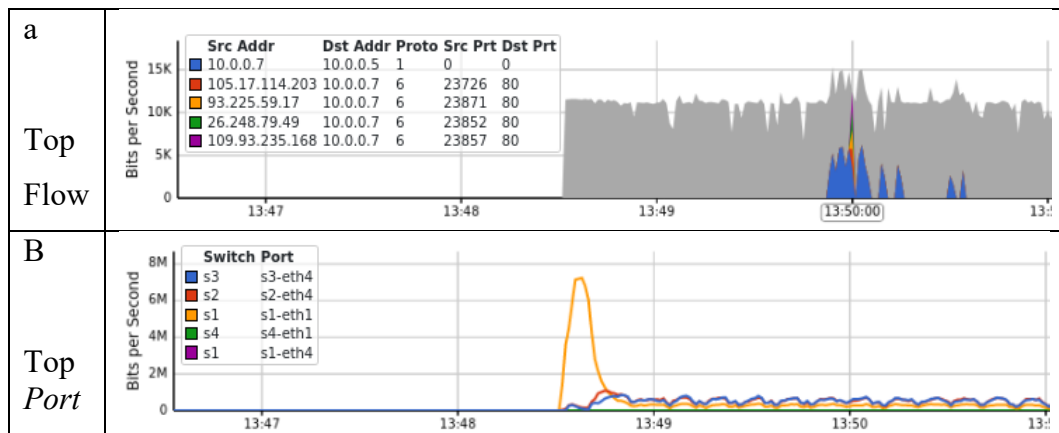
Gambar 4.10 Merupakan hasil deteksi dan mitigasi lalu lintas UDP *flood* yang ditampilkan pada terminal, MLP berhasil mendeteksi serangan UDP *flood* dengan akurasi tinggi. Hasil deteksi menunjukkan bahwa MLP dapat secara efektif membedakan antara lalu lintas normal dan lalu lintas serangan, dengan minimal kesalahan deteksi atau *false negative*. Dari 526 paket UDP *flood* yang di kirimkan, sistem menunjukkan bahwa tidak ada lalu lintas normal yang terdeteksi selama serangan. Semua paket yang dianalisis diklasifikasikan sebagai lalu lintas DDoS.

Pada sistem, berhasil mengidentifikasi *host 7* sebagai potensi korban serangan dalam setiap analisis. Selain itu terdapat respon sistem yang menunjukan *port* dan *switch* yang di blokir yaitu pada *port 1* di *switch 1*. Hasil ini telah sesuai dengan perintah yang di kirimkan sebelumnya untuk membuat paket UDP *flood* yaitu perintah *hping3* dari *host 1* menuju *host 7*.

#### 4.2.4 TCP SYN *flood*

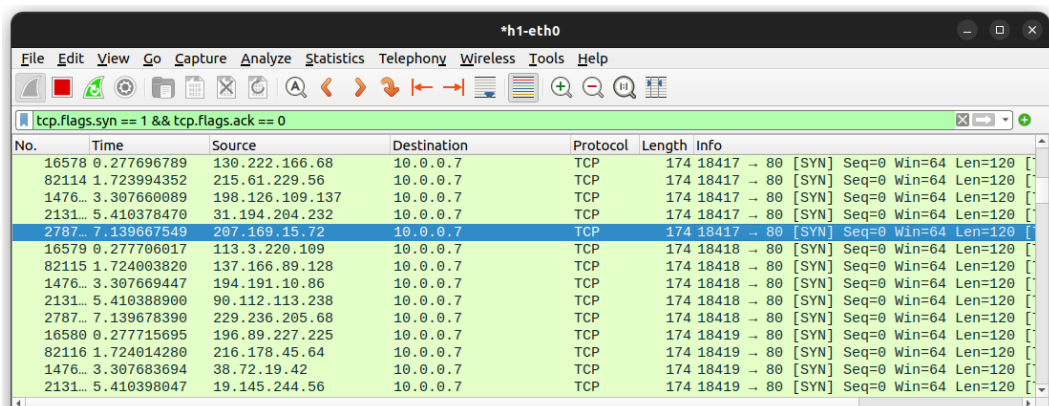
Serangan TCP SYN *flood* ditujukan untuk menghabiskan sumber daya server dengan membanjiri SYN *packets*. Deteksi dalam kasus ini dilakukan dengan mengamati peningkatan tajam dalam jumlah SYN *packets* yang tidak diikuti oleh ACK *packets*. Berikut merupakan gambar karakteristik TCP SYN *flood* yang di

ciptakan oleh *traffic generator Hping3* dari *host 1* menuju *host 7* dimana Penyerang mengirimkan sejumlah besar *SYN packets* ke server target.



Gambar 4.11 Hasil monitoring lalu lintas TCP SYN, (a) Hasil analisis flow saat TCP SYN flood, (b) Hasil analisis port saat TCP SYN flood

Gambar 4.11 merupakan hasil monitoring paket TCP SYN flood pada sistem. Pada grafik *Top Flows*, terlihat lonjakan volume trafik yang sangat tinggi mencapai 15 juta *Bits/s*. Adapun pada port *s1-eth1* yang merupakan sumber DDoS menampilkan jumlah aliran data sebanyak 7 juta *Bits/s* yang diakibatkan oleh sejumlah besar *SYN packets* yang dikirimkan oleh penyerang tanpa menyelesaikan proses *handshake* sehingga meninggalkan koneksi dalam status setengah terbuka dan memaksa server untuk mengalokasikan sumber daya untuk setiap koneksi yang tidak pernah diselesaikan. Untuk melihat *SYN packets* pada sistem digunakan Wireshark, yang di tunjukan pada gambar berikut.



Gambar 4.12 Lalu lintas TCP SYN flood pada Wireshark

Gambar 4.12 menunjukkan aplikasi Wireshark dengan filter “tcp.flags.syn == 1 && tcp.flags.ack == 0”, yang digunakan untuk menampilkan hanya paket TCP SYN tanpa bit ACK yang disetel. Hasil monitoring menunjukkan bahwa ada banyak koneksi yang tidak pernah diselesaikan yaitu paket SYN tanpa ACK. Hal ini dapat diidentifikasi dengan membandingkan jumlah paket SYN terhadap paket SYN-ACK dan ACK. Jumlah paket SYN yang jauh lebih tinggi dibandingkan dengan paket ACK menunjukkan adanya serangan TCP SYN *flood*.

```
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
-----
Traffic Analysis:
Legitimate Traffic = 0
DoS Traffic = 100
Total Traffic analyzed: 100
DDoS traffic detected.
Victim Host: h7
Mitigation process in progress!
```

Gambar 4.13 Hasil deteksi dan mitigasi lalu lintas TCP SYN *flood*

Berdasarkan Gambar 4.13, MLP menunjukkan kemampuan yang sangat baik dalam mengidentifikasi dan membedakan serangan ini dari lalu lintas TCP normal. Dari 100 paket yang di dikirimkan oleh *traffic generator*, semuanya terdeteksi sebagai DDoS. Dari hasil deteksi ini selanjutnya, sistem melakukan mitigasi kepada sumber *port* yang diklasifikasikan sebagai lalu lintas DDoS. Hasil ini menunjukkan bahwa sistem mampu untuk memblokir *port* 1 pada *switch* 1 yang merupakan sumber DDoS. Berdasarkan data ini, dapat disimpulkan bahwa MLP efektif dalam mengenali pola lalu lintas abnormal dan mendeteksi serangan TCP SYN *flood* dengan tepat, memastikan perlindungan yang lebih baik terhadap ancaman jaringan.

#### 4.2.5 Rata -Rata Efektivitas Deteksi

Tabel 4.1 merupakan hasil dari deteksi sistem. Tabel ini menjelaskan performa model *Multilayer Perceptron* (MLP) dalam mendeteksi serangan DDoS dan mengenali lalu lintas normal. *Confusion matrix* pada tabel ini menggambarkan hasil deteksi untuk empat jenis lalu lintas, yaitu lalu lintas normal, *ICMP flood*, *UDP flood*, dan *TCP SYN flood*.

Tabel 4.1 Hasil deteksi sistem

		Prediksi			
		Normal	ICMP <i>flood</i>	UDP <i>flood</i>	TCP SYN <i>flood</i>
Aktual	Normal	264	0	0	0
	ICMP <i>flood</i>	0	470	0	0
	UDP <i>flood</i>	0	0	526	0
	TCP SYN <i>flood</i>	0	0	0	100

Pada kondisi lalu lintas normal, model MLP berhasil mendeteksi seluruh paket dengan benar sebagai lalu lintas normal, menunjukkan True Positives (TP) sebesar 264 dan tidak ada False Positives (FP) atau *False Negatives* (FN). Ini menunjukkan bahwa model memiliki akurasi yang sempurna dalam mengenali lalu lintas yang wajar.

Dalam eksperimen *ICMP flood*, model MLP mendeteksi seluruh paket serangan sebagai DDoS dengan TP sebesar 470 dan tanpa FP atau FN. Hasil ini menunjukkan bahwa model sangat efektif dalam mengenali pola serangan *ICMP flood* dengan akurasi yang tinggi.

Pada serangan *UDP flood*, model MLP juga menunjukkan performa yang sangat baik dengan TP sebesar 526 dan tidak ada FP atau FN, yang menunjukkan bahwa model dapat secara efektif membedakan antara lalu lintas normal dan lalu lintas serangan *UDP flood*.

Serangan *TCP SYN flood* juga dideteksi dengan sempurna oleh model MLP, dengan TP sebesar 100 dan tanpa adanya FP atau FN. Ini menunjukkan kemampuan model dalam mengidentifikasi dan mengklasifikasikan serangan *TCP SYN flood* dengan akurasi yang tinggi.

Secara keseluruhan, hasil yang ditampilkan dalam Tabel 4.1 menunjukkan bahwa model MLP memiliki akurasi pengujian yang sangat baik mencapai 100% dalam mendeteksi serangan DDoS. Tidak adanya *false positives* dan *false negatives* menunjukkan keefektifan dan keandalan model dalam menjaga keamanan jaringan dengan mengklasifikasikan lalu lintas secara akurat. Selain itu model MLP tidak hanya efektif dalam mendeteksi berbagai jenis serangan DDoS tetapi juga dalam memastikan bahwa lalu lintas normal tidak salah dikenali sebagai ancaman.

Efektifitas hasil deteksi akan berpengaruh besar pada hasil mitigasi. Hal ini menunjukkan bahwa model dapat digunakan dalam lingkungan jaringan yang sibuk dan kompleks tanpa menyebabkan peringatan palsu yang dapat mengganggu operasi jaringan.

### 4.3. Analisis Performa Jaringan

Untuk mengukur performa jaringan dan efektifitas sistem dalam mendeteksi dan mengatasi serangan DDoS pada penelitian ini, digunakan beberapa parameter utama yaitu *delay*, *throughput*, *jitter*, dan *resource utilization*. Pengukuran dilakukan pada kondisi normal dan selama serangan DDoS untuk memahami dampak serangan terhadap performa jaringan.

#### 4.3.1 Jitter

*Jitter* adalah variasi waktu pengiriman paket dalam jaringan, yang merupakan salah satu parameter penting dalam mengukur kualitas layanan jaringan. *Jitter* diukur sebagai selisih antara waktu tiba dari paket-paket berturut-turut yang diterima. Berikut merupakan data hasil pengukuran *jitter* pada berbagai skenario jaringan.

```
-----
Client connecting to 10.0.0.2, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11215,21 us (kalman adjust)
UDP buffer size: 208 KByte (default)
-----
[ 1] local 10.0.0.10 port 57098 connected with 10.0.0.2 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 1] 0.0000-10.0158 sec 1.25 MBytes  1.05 Mbits/sec
[ 1] Sent 896 datagrams
[ 1] Server Report:
[ ID] Interval      Transfer      Bandwidth      Jitter    Lost/Total Datagrams
[ 1] 0.0000-10.0105 sec 1.25 MBytes  1.05 Mbits/sec  0.026 ms  0/895 (0%)
root@ivan-virtual-machine:~/Documents/DDoS Detection V3# █
```

Gambar 4.14 *Jitter* pada jaringan lalu lintas normal



Gambar 4.14 merupakan hasil pengukuran *jitter* pada saat dilalui lalu lintas normal. Pengukuran *jitter* pada kondisi normal menunjukkan nilai yang sangat rendah, yaitu 0.026 ms. Menurut standar TIPHON, nilai ini masuk dalam kategori "sangat bagus" karena berada di bawah 75 ms. Hal ini mencerminkan performa jaringan yang optimal tanpa adanya gangguan signifikan.

Pada skenario kedua, sistem diuji selama terjadinya serangan DDoS, sementara pada skenario ketiga, sistem diuji dengan paket DDoS namun disertai dengan proses mitigasi. Paket DDoS dihasilkan menggunakan *traffic generator hping3* dan dikirimkan dari H1 ke H7. Pengujian *jitter* dilakukan pada H2 untuk mengukur dampak dari serangan tersebut. Proses mitigasi dilakukan dengan memblokir *port* sumber DDoS, yaitu *port eth1* pada *switch 1*. Tujuan dari mitigasi ini adalah untuk menilai efektivitasnya dalam mengurangi dampak serangan DDoS pada jaringan. Berikut adalah hasil pengukuran *jitter* saat serangan DDoS terjadi dan setelah dilakukan mitigasi.

Tabel 4.2 *Jitter* selama serangan dan setelah mitigasi DDoS

<b>Jenis Lalu lintas DDoS</b>	<b><i>Jitter</i> Selama DDoS Terjadi (ms)</b>	<b><i>Jitter</i> Setelah Mitigasi (ms)</b>
ICMP <i>flood</i>	RTO	0.011
UDP <i>flood</i>	RTO	0.013
TCP SYN <i>flood</i>	RTO	0.007

Tabel 4.2 merupakan pengukuran *jitter* saat serangan DDoS terjadi, pengukuran tidak dapat dilakukan karena paket tidak dapat diterima atau biasa disebut *Request time out* (RTO). Hal ini disebabkan oleh kondisi jaringan yang tidak stabil akibat banyaknya paket yang membanjiri jaringan selama serangan DDoS. Kemacetan dan kelebihan beban jaringan yang dihasilkan oleh serangan tersebut mengakibatkan ketidakmampuan jaringan untuk menjaga interval waktu pengiriman paket yang konsisten.

Namun, setelah mitigasi diterapkan, *jitter* menunjukkan penurunan yang signifikan, mendekati kondisi normal. Pada serangan ICMP *flood*, *jitter* turun menjadi 0.011 ms, pada serangan UDP *flood* menjadi 0.013 ms, dan pada serangan TCP SYN *flood* menjadi 0.007 ms. Berdasarkan standar TIPHON, nilai ini masuk dalam kategori "sangat bagus" karena berada di bawah 75 ms. Mekanisme mitigasi

yang efektif berhasil mengurangi dampak serangan DDoS, mengembalikan stabilitas jaringan, dan memastikan bahwa paket data tiba dengan interval waktu yang lebih konsisten. Hasil pengukuran *jitter* setelah mitigasi menunjukkan nilai jauh lebih rendah dibandingkan dengan saat serangan berlangsung, menandakan bahwa langkah mitigasi yang diterapkan berhasil memulihkan performa jaringan, memastikan kualitas layanan yang optimal bagi aplikasi *real-time*.

#### 4.3.2 Delay

*Delay* menunjukkan waktu round-trip dari pengiriman permintaan hingga penerimaan respons. *Delay* jaringan diukur dengan menggunakan perintah "*ping*". Perintah "*ping*" ini mengirimkan paket ICMP Echo Request dan mengukur waktu yang dibutuhkan untuk menerima ICMP Echo Reply dari *host* tujuan. Berikut adalah contoh hasil dari perintah "*ping*" yang menunjukkan *delay* jaringan.

Tabel 4.3 *Delay* jaringan pada lalu lintas normal

<b>Delay Lalu lintas Normal</b>			
min	avg	max	<i>mdev</i>
0.052 ms	0.280 ms	8.492 ms	1.1 ms

Berdasarkan Tabel 4.3, pada lalu lintas data normal, *delay* rendah dan stabil, menunjukkan bahwa paket ICMP *echo request* dan *echo reply* mengalir dengan lancar. Nilai *delay* rata-rata pada lalu lintas jaringan normal menunjukkan angka 0.28 ms. Berdasarkan standar TIPHON, nilai ini masuk dalam kategori "sangat bagus" yang menandakan bahwa jaringan sehat dan tidak ada gangguan signifikan dalam jalur komunikasi antara *host*.

Selanjutnya, sistem diuji selama berlangsungnya serangan DDoS, dan sistem diuji dengan adanya paket DDoS namun disertai dengan proses mitigasi. Paket DDoS dihasilkan menggunakan *traffic generator hping3* dan dikirimkan dari H1 ke H7. Kemudian dilakukan pengukuran *delay* pada H2 untuk mengevaluasi dampak serangan tersebut. Pada proses mitigasi sistem akan memblokir *port* sumber DDoS dari hasil prediksi model MLP, yaitu *port eth1* pada *switch 1*. Berikut adalah hasil pengukuran *delay* saat serangan DDoS terjadi dan setelah mitigasi diterapkan.

Tabel 4.4 *Delay* selama serangan dan setelah mitigasi DDoS

Jenis Lalu lintas DDoS	Selama DDoS Terjadi (ms)				Setelah Mitigasi (ms)			
	min	avg	max	<i>mdev</i>	min	avg	max	<i>mdev</i>
ICMP <i>flood</i>	RTO	RTO	RTO	RTO	0.028	0.242	8.717	1.162
UDP <i>flood</i>	RTO	RTO	RTO	RTO	0.037	0.4	20.409	2.322
TCP SYN <i>flood</i>	RTO	RTO	RTO	RTO	0.04	0.375	17.544	2.045

Tabel 4.4 adalah hasil pengukuran *delay* pada dua skenario jaringan yaitu Selama DDoS Terjadi dan setelah mitigasi. Pada kondisi jaringan selama serangan DDoS. Pengukuran *delay* tidak dapat dilakukan karena kemacetan yang sangat parah dalam jaringan, yang mengakibatkan banyak paket ICMP Echo Request tidak mendapatkan respons ICMP Echo Reply dari *host* tujuan sehingga terjadi RTO. Ketidakmampuan untuk mengukur *delay* ini menunjukkan adanya kemacetan yang ekstrem dan ketidakstabilan dalam jaringan.

Setelah mitigasi, kondisi *delay* jaringan menunjukkan perubahan yang signifikan dibandingkan dengan saat serangan DDoS berlangsung. Untuk serangan ICMP *flood*, setelah mitigasi diterapkan, *delay* menunjukkan nilai minimum sebesar 0.028 ms, rata-rata 0.242 ms, maksimum 8.717 ms, dan deviasi standar (*mdev*) sebesar 1.162 ms. Angka-angka ini menunjukkan bahwa mekanisme mitigasi mampu mengembalikan *delay* jaringan ke tingkat yang mendekati kondisi normal, meskipun masih ada sedikit fluktuasi. Berdasarkan standar TIPHON, nilai rata-rata *delay* ini masuk dalam kategori "Sangat Bagus" karena berada di bawah 150 ms.

Pada serangan UDP *flood*, *delay* setelah mitigasi mencatat nilai minimum sebesar 0.037 ms, rata-rata 0.4 ms, maksimum 20.409 ms, dan deviasi standar (*mdev*) sebesar 2.322 ms. Ini menandakan bahwa mitigasi berhasil mengurangi dampak serangan, meskipun ada beberapa lonjakan dalam waktu *round-trip* yang mungkin disebabkan oleh sisa-sisa kemacetan lalu lintas DDoS. Menurut standar TIPHON, nilai rata-rata *delay* setelah mitigasi pada serangan UDP *flood* juga masih dalam kategori "Sangat Bagus".

Pada serangan TCP SYN *flood*, meskipun ada peningkatan dalam nilai maksimum, rata-rata *delay* dan deviasi standar menunjukkan adanya perbaikan signifikan dalam stabilitas jaringan setelah mitigasi. Nilai rata-rata *delay* setelah di

lakukan mitigasi pada serangan TCP SYN *flood* juga termasuk dalam kategori "Sangat Bagus" berdasarkan standar TIPHON.

#### 4.3.3 Throughput

*Throughput* jaringan dapat diukur dengan memeriksa jumlah data yang berhasil dikirimkan dalam satuan waktu tertentu. Untuk mengukur *throughput*, digunakan *iperf*, untuk mengukur kecepatan transmisi data antara dua *host*. Berikut adalah contoh hasil dari *iperf* yang menunjukkan *throughput*.

Tabel 4.5 *Throughput* jaringan pada lalu lintas normal

<b><i>Throughput</i> Lalu lintas Normal</b>		
Interval	Transfer	<i>Bandwidth</i>
0-10 s	6.24 Gbps	5.36 Gbps

Tabel 4.5 menunjukkan bagaimana mitigasi mempengaruhi jumlah data yang di transfer dan *bandwidth* pada berbagai jenis lalu lintas data. Pada lalu lintas normal menunjukkan bahwa dalam interval waktu 10 detik, sebanyak 6.24 Gbps data berhasil ditransfer dengan *bandwidth* sebesar 5.36 Gbits/sec. Menurut standar TIPHON, nilai ini menunjukkan performa jaringan yang masuk dalam kategori "sangat bagus" karena *throughput* yang tinggi. Hal ini menandakan bahwa jaringan berada dalam kondisi optimal sehingga mampu menangani volume data yang besar.

Selanjutnya, dilakukan pengukuran *throughput* selama serangan DDoS, dan setelah mitigasi diterapkan. Paket DDoS dari H1 ke H7 dihasilkan menggunakan *hping3* dan pengukuran *throughput* dilakukan pada H2. Mitigasi dilakukan berdasarkan prediksi model MLP dengan memblokir *port* sumber serangan DDoS yaitu *port eth1* pada *switch* 1.

Tabel 4.6 *Throughput* selama serangan dan setelah mitigasi DDoS

Jenis Lalu lintas DDoS	Interval	Selama DDoS Terjadi (Gbyts)		Setelah Mitigasi (Gbyts)	
		Transfer	<i>Bandwidth</i>	Transfer	<i>Bandwidth</i>
ICMP <i>flood</i>	0-10 s	RTO	RTO	3.75	3.21
UDP <i>flood</i>		RTO	RTO	3.61	3.09
TCP SYN <i>flood</i>		RTO	RTO	4.06	3.49

Tabel 4.6 Menunjukkan nilai *throughput* jaringan selama serangan DDoS berlangsung, *throughput* tidak dapat di ukur karena topologi jaringan tidak lagi dapat saling berkomunikasi, yang ditandai dengan pesan kesalahan *tcp connect failed* saat melakukan pengecekan nilai *throughput*. Penyebab rusaknya koneksi antar jaringan adalah beban jaringan yang berlebihan, yang menyebabkan penurunan efisiensi dalam transmisi data. Hal ini menunjukkan bahwa serangan DDoS dapat secara signifikan mengurangi kapasitas jaringan untuk mentransmisikan data.

Setelah mitigasi diterapkan, kondisi jaringan menunjukkan perbaikan yang signifikan pada berbagai jenis serangan DDoS. Pada serangan ICMP *flood*, transfer tercatat sebesar 3.75 *Gbytes* dengan *bandwidth* 3.21 *Gbits/sec*. Pada serangan UDP *flood*, setelah mitigasi, transfer mencapai 3.61 *Gbytes* dengan *bandwidth* sebesar 3.09 *Gbits/sec*. Untuk serangan TCP SYN *flood*, transfer setelah mitigasi tercatat sebesar 4.06 *Gbytes* dengan *bandwidth* 3.49 *Gbits/sec*. Nilai-nilai ini menunjukkan bahwa mitigasi berhasil mengurangi dampak serangan tetapi dan memungkinkan jaringan untuk berfungsi dengan kinerja yang hampir optimal. Menurut standar TIPHON, nilai ini masuk dalam kategori "sangat bagus" karena *throughput* jaringan mencapai *gigabit* per detik.

#### 4.3.4 Packet Loss

*Packet loss* adalah parameter penting yang digunakan untuk mengukur performa jaringan, terutama dalam kondisi normal dan saat terjadi serangan DDoS. *Packet loss* mengindikasikan persentase paket data yang hilang atau tidak berhasil mencapai tujuan selama proses transmisi. Parameter ini sangat penting untuk mengevaluasi keandalan dan stabilitas jaringan. Berikut adalah hasil pengukuran *packet loss* menggunakan perintah "*ping*".

Tabel 4.7 *Packet loss* pada lalu lintas normal

<b>Packet Loss Lalu lintas Normal</b>	
Jumlah Paket	<i>packet loss</i> (%)
100	0

Tabel 4.7 menunjukkan hasil pengukuran *packet loss* pada kondisi normal. Hasil pengukuran *packet loss* pada lalu lintas jaringan normal, menunjukkan tidak ada *packet loss* yang terjadi selama 100 pengiriman paket, yang menunjukkan bahwa jaringan beroperasi dalam kondisi optimal. Semua paket *ICMP echo request* yang dikirim oleh h1 diterima dan direspon oleh h7 tanpa adanya kehilangan paket. Ini mencerminkan performa jaringan yang stabil dan efisien. Menurut standar TIPHON, nilai ini masuk dalam kategori "sangat bagus" karena *packet loss* berada di bawah 1%. Ini mencerminkan performa jaringan yang stabil dan efisien.

Selanjutnya, dilakukan pengukuran *packet loss* selama terjadinya serangan DDoS dan setelah penerapan mitigasi. Paket DDoS dihasilkan menggunakan *hping3* dan dikirim dari H1 ke H7, sementara pengukuran *packet loss* dilakukan pada H2. Proses mitigasi dilakukan berdasarkan prediksi model MLP dengan memblokir *port* sumber serangan DDoS, yaitu *port eth1* pada *switch 1*.

Tabel 4.8 *Packet loss* selama serangan dan setelah mitigasi DDoS

Jenis Lalu lintas Data	Jumlah Paket	Selama DDoS Terjadi	Setelah Mitigasi
		<i>packet loss</i> (%)	<i>packet loss</i> (%)
ICMP <i>flood</i>	100	100	39
UDP <i>flood</i>		100	35
TCP SYN <i>flood</i>		100	49

Tabel 4.8 adalah data *packet loss* Selama DDoS Terjadi dan setelah mitigasi dilakukan, yang di tandai oleh tingkat *packet loss* meningkat secara signifikan selama serangan DDoS. Pada tabel di atas dapat diamati bahwa nilai *packet loss* pada saat serangan DDoS tanpa adanya proses mitigasi serangan menyebabkan kemacetan dalam jaringan, mengakibatkan banyak paket yang hilang dan tidak mencapai tujuan mereka. Tingkat *packet loss* yang tinggi menunjukkan bahwa serangan DDoS secara drastis mengganggu aliran data dalam jaringan, menurunkan efisiensi dan kinerja jaringan secara keseluruhan.

Selama serangan ICMP *flood*, *packet loss* Selama DDoS Terjadi adalah 100%, menunjukkan bahwa tidak ada paket yang berhasil mencapai tujuan karena kemacetan jaringan yang parah. Setelah mitigasi, *packet loss* turun menjadi 39%, menunjukkan perbaikan signifikan dalam performa jaringan meskipun belum sepenuhnya pulih. Menurut standar TIPHON, nilai ini masih dalam kategori "buruk" karena lebih dari 25%.

Pada serangan UDP *flood*, *packet loss* juga mencapai 100%. Setelah mitigasi, *packet loss* berkurang menjadi 35%, yang masih dalam kategori "buruk" menurut standar TIPHON, tetapi menunjukkan bahwa tindakan mitigasi efektif dalam mengurangi dampak serangan dan memungkinkan lebih banyak paket untuk berhasil dikirimkan.

Pada serangan TCP SYN *flood*, *packet loss* adalah 100%. Setelah mitigasi, *packet loss* berkurang menjadi 49%, menunjukkan bahwa meskipun masih ada kehilangan paket yang signifikan, jaringan menunjukkan perbaikan dalam kemampuannya untuk menangani lalu lintas data, meskipun nilai ini tetap dalam kategori "buruk" menurut standar TIPHON.

Secara keseluruhan, hasil pengukuran *packet loss* setelah mitigasi menunjukkan bahwa langkah-langkah mitigasi yang diterapkan efektif dalam mengurangi dampak serangan DDoS. Meskipun *packet loss* belum sepenuhnya kembali ke kondisi normal, terdapat perbaikan yang signifikan. Hal ini membuktikan bahwa strategi mitigasi yang diterapkan berhasil melindungi jaringan dari dampak negatif serangan DDoS. Menurut standar TIPHON, meskipun masih dalam kategori "buruk" mitigasi telah membawa peningkatan yang berarti dalam stabilitas dan efisiensi jaringan.

#### 4.3.5 *Resource Utilization*

*Resource utilization* adalah parameter penting untuk memahami seberapa efisien sistem menggunakan sumber dayanya, seperti CPU dan memori, dalam berbagai kondisi lalu lintas jaringan. Analisis *resource utilization* dilakukan untuk mengevaluasi performa sistem dalam kondisi normal dan selama serangan DDoS, serta untuk memastikan bahwa sistem dapat menangani beban yang tinggi tanpa mengalami penurunan performa yang signifikan.

Namun karena simulasi dilakukan secara virtual menggunakan Mininet sehingga memiliki keterbatasan dalam pembagian dan pemantauan sumber daya CPU ke setiap *host* virtual. *Host* dalam Mininet adalah *namespace* jaringan Linux, yang berarti mereka adalah entitas virtual yang berjalan di atas kernel Linux yang sama dengan sistem *host*. Meskipun mereka dapat memiliki alamat IP dan aturan jaringan mereka sendiri, mereka berbagi kernel yang sama dan sumber daya CPU

dengan sistem *host*. Sehingga penggunaan CPU yang terbaca mencerminkan penggunaan seluruh sistem, bukan *host* virtual individu. Namun pada penelitian ini tetap dilakukan pemantauan *resource* untuk melihat respon sistem terhadap berbagai kondisi.

Tabel 4.9 *Resource utilization* pada lalu lintas normal

Resource Lalu lintas Normal (%)		
CPU Sistem	Memory Usage	CPU Process
12.5	39.1	0.3

Tabel 4.9 merupakan hasil pemantauan *resource utilization* dalam kondisi normal menunjukkan bahwa sistem beroperasi dengan efisien, dengan penggunaan CPU dan memori yang stabil. CPU Process menunjukkan angka 0.3%, yang menandakan persentase penggunaan CPU dalam kondisi normal, nilai ini cenderung rendah. CPU Sistem menunjukkan angka 12.5%, yang merupakan persentase total penggunaan CPU sistem dalam kondisi normal, nilai ini juga relatif rendah. Memory Usage berada pada angka 39.1%, nilai ini stabil dan tidak menunjukkan lonjakan yang signifikan. Ini mencerminkan bahwa jaringan mampu menangani lalu lintas normal tanpa mengalami kelebihan beban.

Selanjutnya, dilakukan pengukuran *resource utilization* selama serangan DDoS dan setelah mitigasi diterapkan. Paket DDoS dikirim dari H1 ke H7 menggunakan *traffic generator hping3*, dan pemantauan pemanfaatan sumber daya dilakukan pada H2. Pengukuran mencakup penggunaan CPU, memori, dan *bandwidth* untuk menilai dampak serangan terhadap kinerja sistem. Mitigasi dilakukan berdasarkan prediksi model MLP dengan memblokir *port* sumber serangan DDoS, yaitu *port eth1* pada *switch* 1. Tujuan dari pengukuran ini adalah untuk menilai efektivitas mitigasi dalam mengurangi beban pada sumber daya jaringan dan memastikan bahwa sistem tetap berfungsi dengan baik meskipun terjadi serangan.

Tabel 4.10 *Resource utilization* pada lalu lintas DDoS

Jenis Lalu lintas DDoS	Resource Selama DDoS Terjadi (%)			Resource Setelah Mitigasi (%)		
	CPU Sistem	Memory Usage	CPU Process	CPU Sistem	Memory Usage	CPU Process



ICMP <i>flood</i>	99.2	52	1.9	74.7	8.56	11.1
UDP <i>flood</i>	100	59.8	3.37	53.6	51.2	0.39
TCP SYN <i>flood</i>	100	55.9	0.58	48.8	39.6	0.37

Tabel 4.10 menunjukkan penggunaan sumber daya CPU dan memori pada berbagai jenis lalu lintas data sebelum dan setelah mitigasi diterapkan. Berikut adalah penjelasan detail berdasarkan data yang ditunjukkan dalam tabel.

Selama serangan ICMP *flood*, mitigasi berhasil mengurangi penggunaan CPU sistem dari 99.2% menjadi 74.7%, dan penggunaan memori dari 52% menjadi 8.56%. Namun, penggunaan CPU oleh proses tertentu meningkat dari 1.9% menjadi 11.1%, menunjukkan bahwa meskipun beban keseluruhan berkurang, proses spesifik yang terkait dengan mitigasi harus bekerja lebih keras untuk menangani serangan.

Pada serangan UDP *flood*, penggunaan CPU sistem berkurang secara signifikan dari 100% menjadi 53.6%, dan penggunaan memori menurun dari 59.8% menjadi 51.2%. Penggunaan CPU oleh proses tertentu juga mengalami penurunan drastis dari 3.37% menjadi 0.39%, menunjukkan bahwa mitigasi tidak hanya mengurangi beban keseluruhan tetapi juga mengoptimalkan penggunaan sumber daya pada *level proses*.

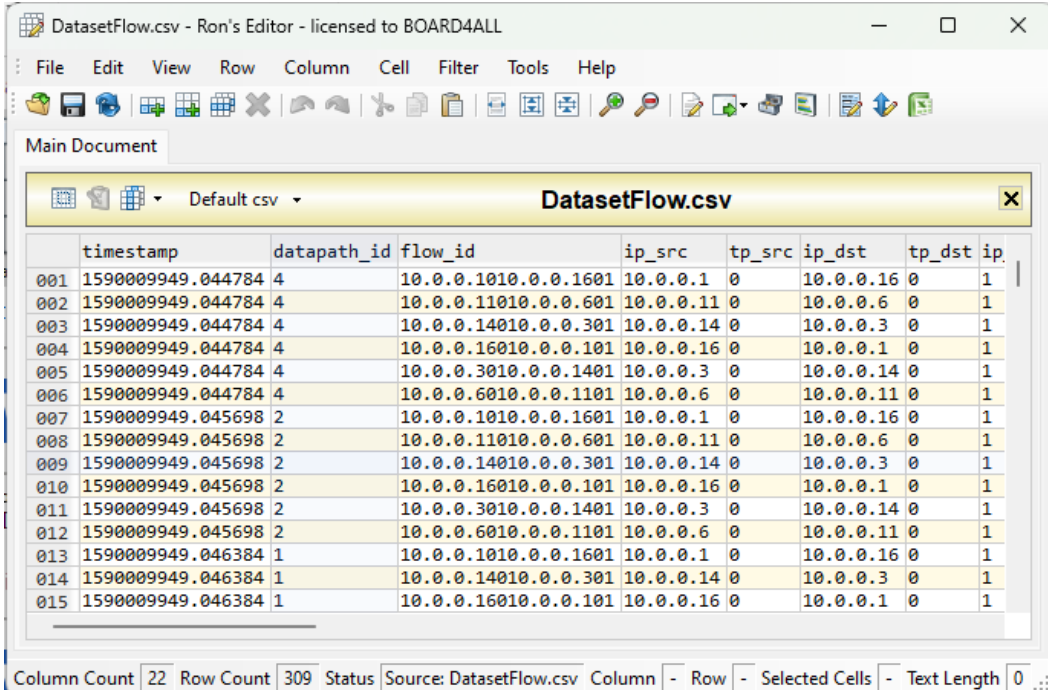
Untuk serangan TCP SYN *flood*, penggunaan CPU sistem turun dari 100% menjadi 48.8%, dan penggunaan memori berkurang dari 55.9% menjadi 39.6%. Penggunaan CPU oleh proses tertentu juga menurun dari 0.58% menjadi 0.37%, menunjukkan bahwa mitigasi berhasil mengurangi beban pada seluruh sistem dan memastikan bahwa sumber daya digunakan dengan lebih efisien.

Penurunan penggunaan sumber daya ini menunjukkan efektivitas langkah-langkah mitigasi dalam mengurangi dampak serangan DDoS, memulihkan performa jaringan, dan memastikan stabilitas serta efisiensi operasi jaringan. Penurunan penggunaan sumber daya ini dikarenakan sistem tidak lagi perlu menyimpan dan memproses informasi yang berlebihan terkait dengan lalu lintas berbahaya. Secara keseluruhan, hasil ini menunjukkan bahwa tindakan mitigasi yang diterapkan efektif dalam mengurangi beban pada sumber daya CPU dan memori selama serangan DDoS. Mitigasi yang diterapkan membantu memulihkan performa jaringan, mengurangi kemacetan, dan memastikan operasi jaringan tetap

stabil dan efisien. Meskipun ada beberapa peningkatan dalam penggunaan CPU oleh proses tertentu selama serangan ICMP *flood*, namun proses mitigasi mampu mengurangi dampak negatif serangan dan memperbaiki kondisi jaringan.

#### 4.4. Hasil dan Analisis Pembuatan *Dataset*

Pembuatan *dataset* merupakan tahap krusial dalam penelitian ini, yang bertujuan untuk mengembangkan model yang dapat dengan akurat mendeteksi dan memitigasi serangan DDoS dalam jaringan SDN. *Dataset* dikumpulkan dengan menggunakan metode simulasi, di mana perintah *ping* digunakan untuk menghasilkan lalu lintas jaringan normal dan *Hping3* digunakan untuk menghasilkan lalu lintas serangan DDoS, yang merefleksikan karakteristik lalu lintas jaringan yang beragam. Dibawah ini merupakan contoh *dataset* yang di hasilkan pada penelitian ini.

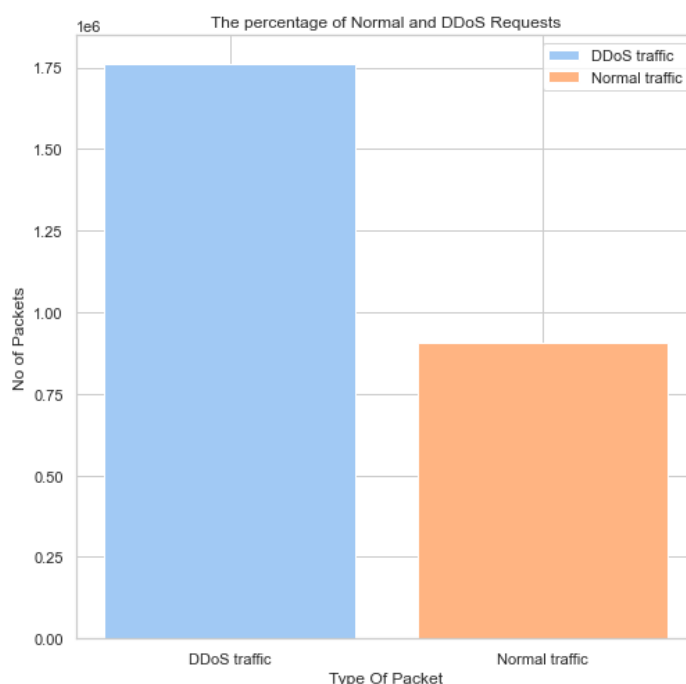


	timestamp	datapath_id	flow_id	ip_src	tp_src	ip_dst	tp_dst	ip
001	1590009949.044784	4	10.0.0.1010.0.0.1601	10.0.0.1	0	10.0.0.16	0	1
002	1590009949.044784	4	10.0.0.11010.0.0.601	10.0.0.11	0	10.0.0.6	0	1
003	1590009949.044784	4	10.0.0.14010.0.0.301	10.0.0.14	0	10.0.0.3	0	1
004	1590009949.044784	4	10.0.0.16010.0.0.101	10.0.0.16	0	10.0.0.1	0	1
005	1590009949.044784	4	10.0.0.3010.0.0.1401	10.0.0.3	0	10.0.0.14	0	1
006	1590009949.044784	4	10.0.0.6010.0.0.1101	10.0.0.6	0	10.0.0.11	0	1
007	1590009949.045698	2	10.0.0.1010.0.0.1601	10.0.0.1	0	10.0.0.16	0	1
008	1590009949.045698	2	10.0.0.11010.0.0.601	10.0.0.11	0	10.0.0.6	0	1
009	1590009949.045698	2	10.0.0.14010.0.0.301	10.0.0.14	0	10.0.0.3	0	1
010	1590009949.045698	2	10.0.0.16010.0.0.101	10.0.0.16	0	10.0.0.1	0	1
011	1590009949.045698	2	10.0.0.3010.0.0.1401	10.0.0.3	0	10.0.0.14	0	1
012	1590009949.045698	2	10.0.0.6010.0.0.1101	10.0.0.6	0	10.0.0.11	0	1
013	1590009949.046384	1	10.0.0.1010.0.0.1601	10.0.0.1	0	10.0.0.16	0	1
014	1590009949.046384	1	10.0.0.14010.0.0.301	10.0.0.14	0	10.0.0.3	0	1
015	1590009949.046384	1	10.0.0.16010.0.0.101	10.0.0.16	0	10.0.0.1	0	1

Gambar 4.15 Hasil *dataset* berdasarkan ekstraksi jaringan SDN

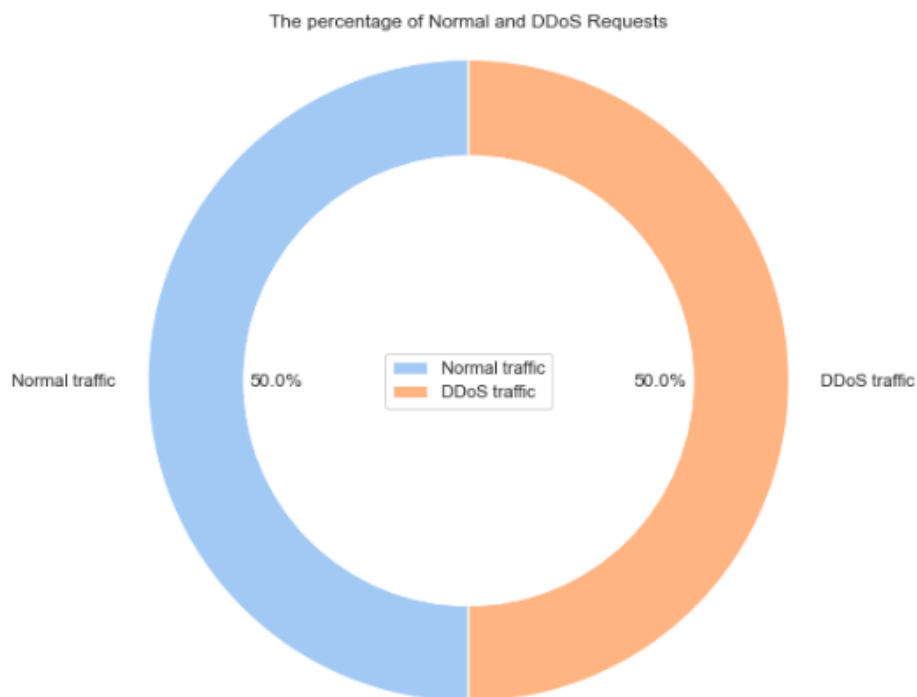
Berdasarkan gambar 4.15, *dataset* yang terkumpul terdiri dari dua kelas utama: lalu lintas normal dan lalu lintas serangan. Setiap kelas memiliki karakteristik tertentu yang tercermin dalam fitur-fitur *dataset*, seperti frekuensi paket, ukuran paket, dan interval waktu antar paket. Analisis awal *dataset*

menunjukkan bahwa ada perbedaan statistik yang signifikan antara dua kelas tersebut, yang memberikan dasar yang baik untuk pembelajaran model.



Gambar 4.16 Distribusi jumlah paket lalu lintas dalam *dataset*

Gambar 4.16 memberikan visualisasi yang jelas tentang komposisi *dataset* yang dikumpulkan sebelum proses *pre-processing*. Grafik tersebut mengilustrasikan perbandingan antara jumlah paket yang dikategorikan sebagai lalu lintas DDoS, yang mencapai 1,760,670 paket, dengan lalu lintas normal, yang berjumlah 906,853 paket. Selanjutnya merupakan tahap *pre-processing* yang bertujuan untuk menormalkan distribusi paket antara dua kelas sehingga model dapat belajar dengan representasi yang lebih seimbang dari kedua kondisi lalu lintas. Hal ini penting karena dengan menyelaraskan skala data, model MLP dapat memperoleh pemahaman yang lebih baik tentang variasi nilai yang ada dalam *dataset*. Selain itu, penghilangan nilai *NAN* juga dilakukan untuk mengurangi distorsi yang mungkin terjadi dalam data. Terakhir, teknik pemisahan data latih dan uji disusun untuk validasi model. Langkah ini memungkinkan model untuk diuji dengan data yang tidak pernah dilihat sebelumnya sehingga dapat mengevaluasi keefektifan dan keakuratan model dalam mendeteksi serangan DDoS. Berikut ini adalah visualisasi dari data hasil *pre-processing*.

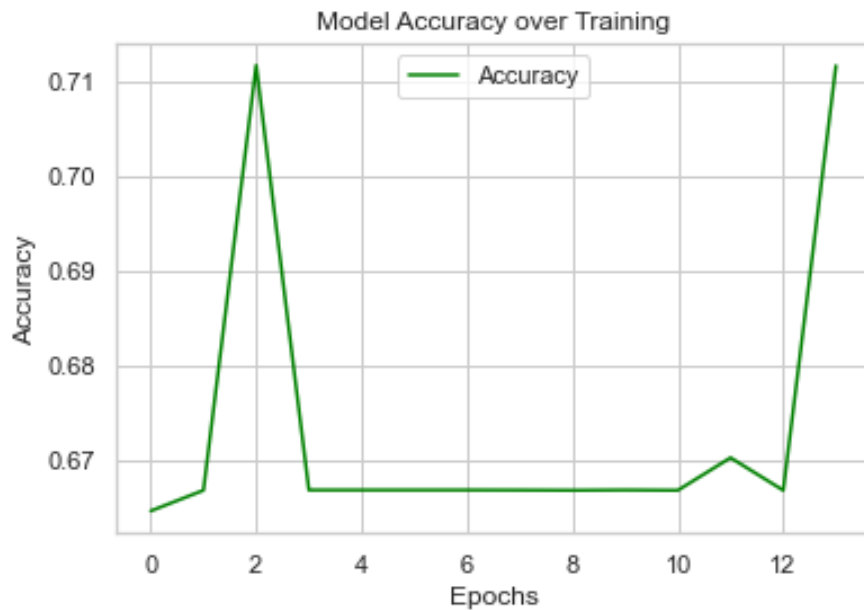


Gambar 4.17 Hasil *pre-processing Dataset*

Gambar 4.17 menunjukkan hasil dari proses *pre-processing dataset*, di mana jumlah data antara lalu lintas normal dan DDoS telah diseimbangkan menjadi 50% untuk masing-masing kategori. Kesetaraan ini penting untuk memastikan bahwa model pembelajaran mesin, seperti MLP yang akan digunakan dalam penelitian ini, tidak bias terhadap satu kelas dan dapat belajar dengan akurat dari kedua jenis lalu lintas. Kesetaraan distribusi ini mendukung pembelajaran yang lebih efektif dan penilaian yang lebih adil terhadap kinerja model dalam mendeteksi serangan DDoS.

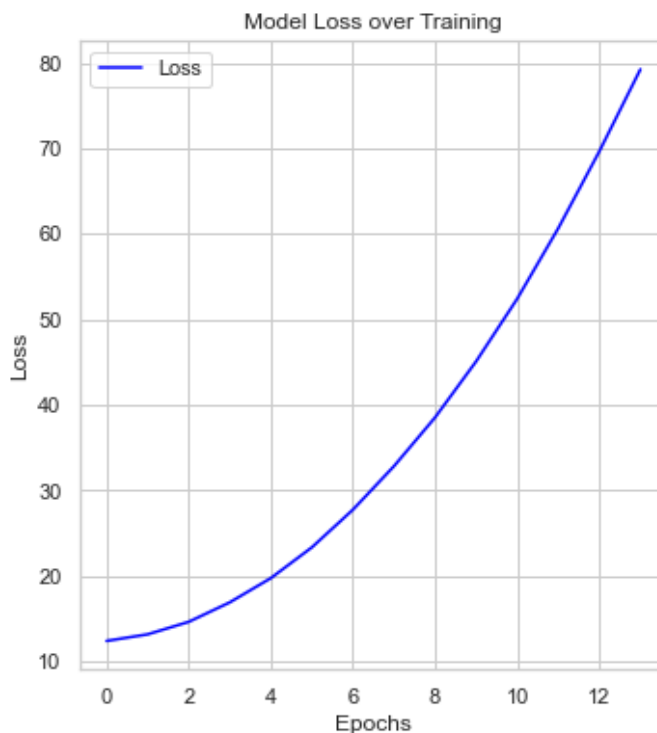
#### 4.5. Evaluasi Model MLP

Evaluasi merupakan aspek penting yang menentukan efektivitas model dalam mengklasifikasikan lalu lintas jaringan sebagai normal atau serangan DDoS. Dengan menggunakan metrik kinerja seperti akurasi, presisi, *recall*, dan *F1-score*, serta analisis kurva ROC. Analisis ini bertujuan untuk memberikan gambaran mengenai kinerja model, serta mengidentifikasi kelebihan serta kelemahan dari model. Dibawah ini merupakan grafik akurasi yang merupakan salah satu evaluasi model MLP.



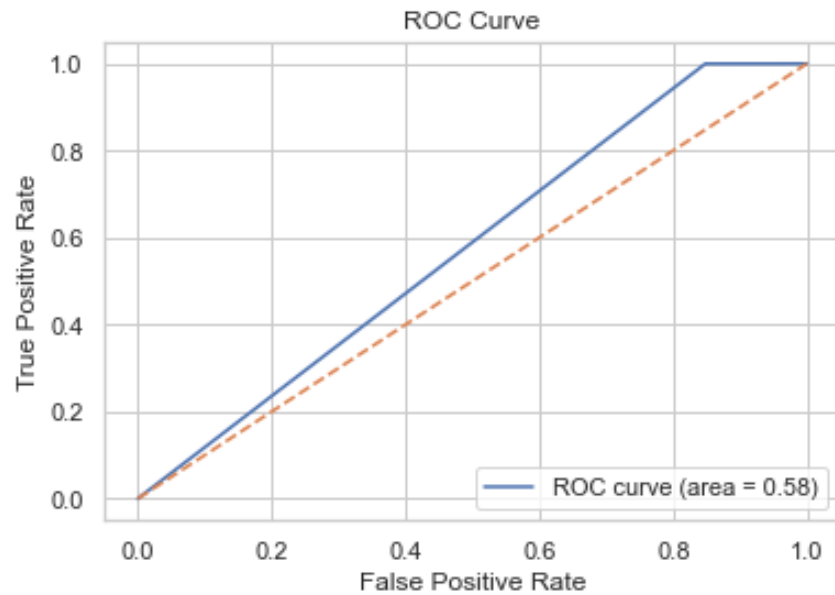
Gambar 4.18 Akurasi model MLP

Gambar 4.18 memperlihatkan tren akurasi model MLP dengan penggunaan teknik *early stopping* selama proses pelatihan. Grafik menunjukkan fluktuasi signifikan dalam akurasi, dengan beberapa puncak yang menembus di atas 70% dan kemudian menurun kembali. Selain itu, terlihat bahwa terdapat variasi yang cukup besar dalam akurasi pada setiap *epoch* yang menunjukkan adanya variasi yang signifikan dalam performa model. Berdasarkan konfigurasi model yang diberikan, di mana *early stopping* diaktifkan untuk menghindari *overfitting*, grafik menunjukkan bahwa pelatihan model berhenti secara otomatis sebelum semua *epoch* terlaksana, hal ini tercermin dari perubahan tajam pada akurasi yang terjadi pada *epoch* terakhir. Penerapan *early stopping* dengan 10% data sebagai validasi menjamin bahwa model tidak melanjutkan pelatihan yang berlebihan, yang dapat dilihat dari stabilisasi akurasi sebelum terjadinya peningkatan tajam tersebut. Hal ini menunjukkan bahwa model telah mencapai kemampuan prediksi yang baik pada data pelatihan dan data validasi. Dengan kata lain, model tidak terlalu fokus pada detail-detail kecil yang hanya ada pada data pelatihan, tetapi mampu mengidentifikasi pola-pola yang lebih umum dan dapat digeneralisasi pada data baru. Selain dari akurasi model, terdapat juga nilai dari kerugian model atau bisa disebut dengan *model loss*. Berikut ini merupakan visualisasi dari nilai kerugian model.



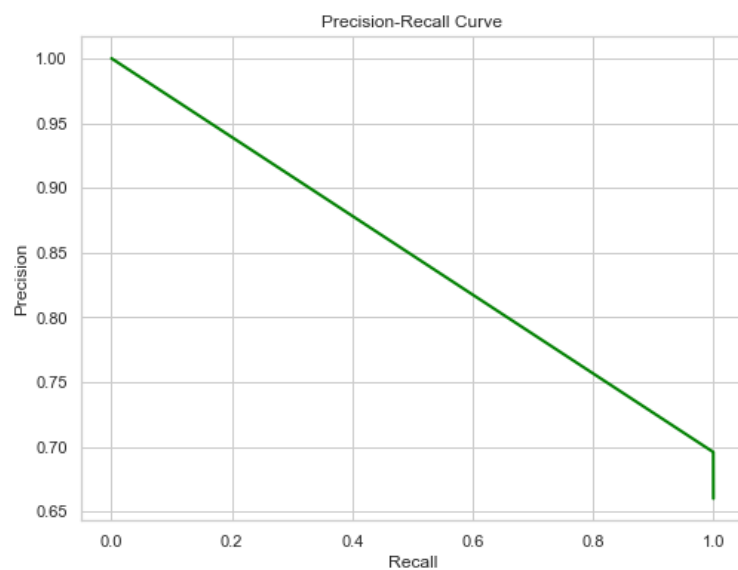
Gambar 4.19 Grafik *loss* model MLP

Gambar 4.19 menampilkan grafik kerugian model selama proses pelatihan dari model MLP. Grafik ini menggambarkan peningkatan kerugian yang terjadi seiring bertambahnya *epoch*. Peningkatan ini menunjukkan adanya isu pada proses pelatihan yang perlu ditangani. Kenaikan kerugian yang terus-menerus seperti ini menunjukkan bahwa model mungkin mengalami kesulitan dalam menyesuaikan bobot untuk mengurangi kesalahan prediksi. Dalam hal ini, penyesuaian bobot yang tidak efektif dapat mengarah pada peningkatan kerugian yang signifikan selama proses pelatihan. Dalam konteks ini, peningkatan kerugian yang terus-menerus memicu aktivasi *early stopping* untuk mencegah pembelajaran yang berlebihan dan tidak efektif. Tujuannya adalah untuk mencegah model dari *overfitting*, di mana model terlalu memfokuskan pada detail kecil yang hanya ada pada data pelatihan tetapi tidak dapat digeneralisasi dengan baik pada data baru. Dengan demikian, model MLP dapat mencapai hasil yang optimal dan dapat digunakan untuk deteksi serangan DDOS dengan akurasi yang tinggi. Dibawah ini merupakan grafik dari ROC yang akan menunjukkan kinerja model dengan cara membandingkan nilai dari *true positif* dan nilai dari *false positive*.



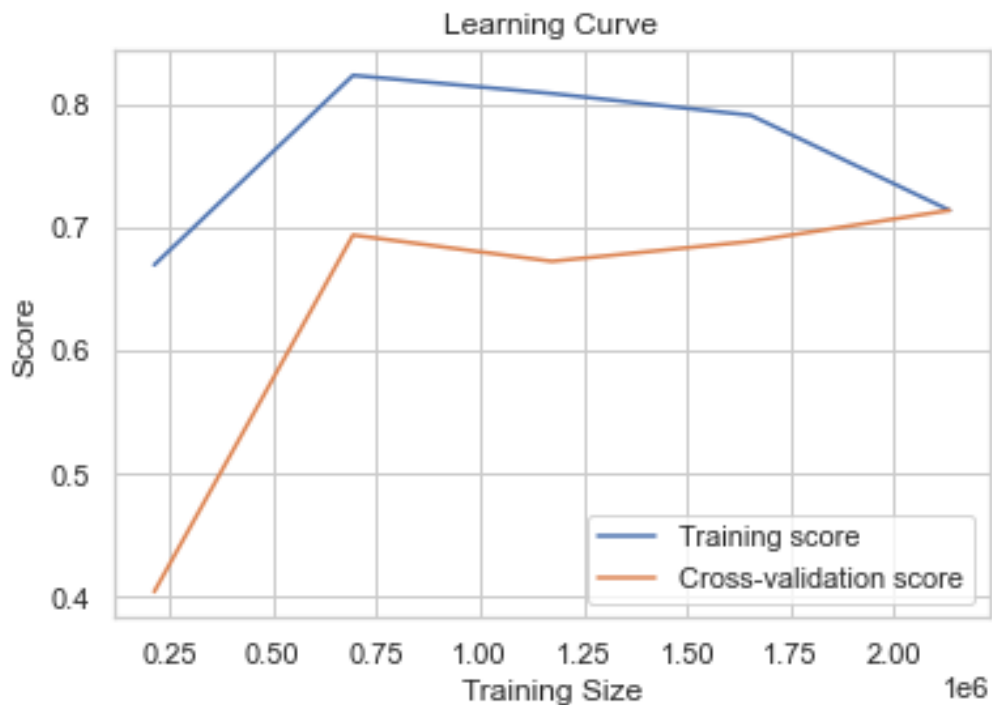
Gambar 4.20 Grafik *Receiver Operating Characteristic* (ROC)

Gambar 4.20 menunjukkan kurva *Receiver Operating Characteristic* (ROC) untuk model MLP yang telah dilatih. Dari grafik tersebut, dapat kita lihat bahwa area di bawah kurva AUC memiliki nilai sebesar 0.58. Nilai tersebut menunjukkan bahwa model ini tidak memiliki kemampuan diskriminasi yang baik dalam membedakan antara kelas positif dan negatif. Untuk menjadi model yang ideal, kurva ROC seharusnya mendekati sudut kiri atas dan memiliki AUC yang mendekati 1.



Gambar 4.21 Grafik presisi-*recall*

Gambar 4.21 yang ditampilkan di bawah ini menunjukkan grafik Precision-Recall yang digunakan untuk mengevaluasi performa model MLP dalam konteks klasifikasi. Grafik ini memberikan gambaran tentang hubungan antara presisi dan *recall*. Presisi merupakan nilai keakuratan model dalam memprediksi nilai positif, sedangkan *recall* adalah nilai yang mengukur keberhasilan model dalam mendeteksi seberapa banyak kasus positif yang berhasil diidentifikasi oleh model. Dari analisis grafik tersebut, dapat terlihat bahwa presisi cenderung menurun seiring dengan peningkatan *recall*. Hal ini mengindikasikan bahwa ketika model berusaha untuk mengidentifikasi lebih banyak kasus positif atau *recall* yang tinggi, kemampuan model untuk menjaga keakuratan prediksinya atau presisi cenderung menurun. Terdapat penurunan yang signifikan pada nilai presisi saat *recall* mendekati 1.0, menunjukkan bahwa model tersebut mungkin mengklasifikasikan banyak kasus negatif sebagai positif dalam rangka mencapai *recall* yang tinggi.



Gambar 4.22 Kurva pembelajaran model

Gambar 4.22 menyajikan grafik learning curve yang menggambarkan perbandingan antara skor pelatihan dan skor validasi silang model MLP terhadap berbagai ukuran set pelatihan. Berdasarkan grafik di atas skor pelatihan meningkat seiring bertambahnya data, menunjukkan peningkatan performa model pada data



yang dikenal. Namun, setelah melewati titik tertentu, skor pelatihan menunjukkan penurunan, yang bisa mengindikasikan *overfitting*. Di sisi lain, skor validasi silang meningkat dan setelah mencapai puncak, menurun sedikit sebelum stabil, menandakan model mulai menggeneralisasi dengan lebih baik terhadap data yang tidak dikenal. Stabilitas skor validasi silang pada akhirnya menunjukkan bahwa model telah mempelajari pola yang relevan dari *dataset*.

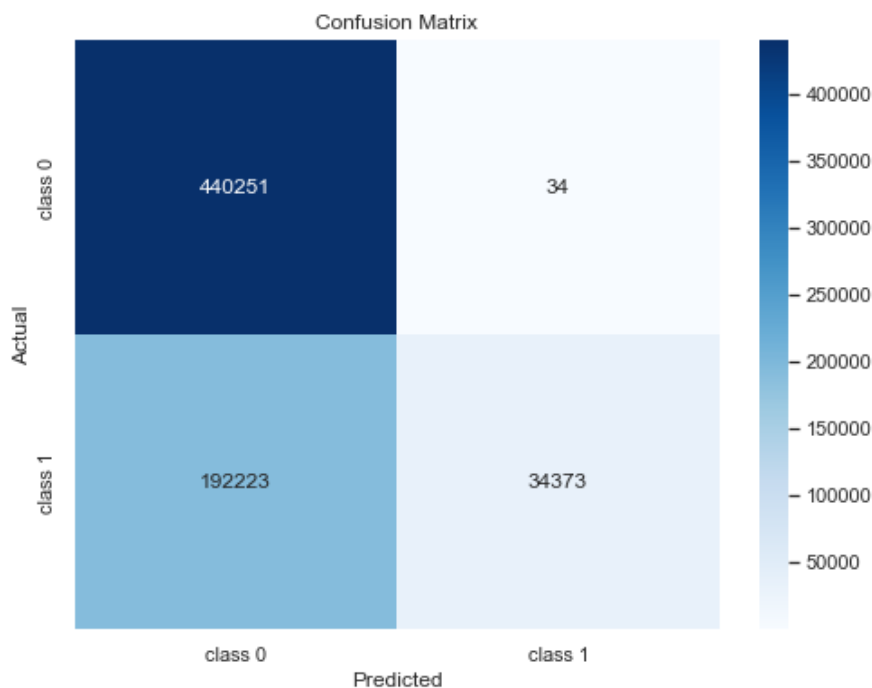
#### 4.6. Efektivitas Pelatihan Model

Pada bagian ini, akan dibahas mengenai efektivitas pelatihan model MLP dalam penelitian ini. Evaluasi dan analisis yang dilakukan bertujuan untuk memahami sejauh mana model MLP dapat mengklasifikasikan lalu lintas jaringan sebagai normal atau serangan DDoS. Berikut adalah laporan hasil klasifikasi, *confusion matrix*, dan dataframe hasil prediksi.

	precision	recall	f1-score	support
0	1.00	0.15	0.26	226596
1	0.70	1.00	0.82	440285
accuracy			0.71	666881
macro avg	0.85	0.58	0.54	666881
weighted avg	0.80	0.71	0.63	666881

Gambar 4.23 Laporan hasil klasifikasi

Gambar 4.23 menampilkan laporan klasifikasi yang mengukur efektivitas model MLP yang telah dilatih. Model ini menunjukkan presisi sempurna untuk kelas 0 (non-DDoS) tetapi dengan *recall* yang rendah, mengindikasikan bahwa sementara semua prediksi positif untuk kelas ini benar, banyak kasus positif sebenarnya yang terlewat. Sebaliknya, untuk kelas 1 (DDoS), model memiliki *recall* sempurna namun presisi yang lebih rendah, menandakan bahwa meskipun semua kasus positif sebenarnya teridentifikasi, terdapat banyak kasus negatif yang salah diklasifikasikan sebagai positif. Akurasi keseluruhan model adalah 71%. Akurasi ini dianggap baik karena mampu mencapai tujuan yaitu mendeteksi DDoS dan membedakannya dengan lalu lintas jaringan normal. Dengan akurasi sebesar 71%, sistem mampu mencapai tujuan yaitu mendeteksi dan memitigasi serangan DDoS serta membedakannya dari lalu lintas normal.



Gambar 4.24 *Confusion matrix* model

Gambar 4.24 menampilkan matriks kebingungan (*confusion matrix*) untuk model MLP yang telah diuji. Dari matriks ini, dapat dilihat beberapa hal penting. Pertama, model MLP berhasil mengklasifikasikan 440,251 kasus sebagai kelas negatif (kelas 0) dengan benar. Ini menunjukkan kemampuan model dalam mengenali kasus negatif. Namun, ada hal yang perlu diperhatikan, yaitu model juga salah mengklasifikasikan 192,223 kasus positif (kelas 1) sebagai negatif. Ini menandakan bahwa ada ruang untuk peningkatan kualitas klasifikasi model ini.

Selanjutnya, jika dilihat lebih detail, terdapat 34 kasus negatif yang salah diklasifikasikan sebagai positif. Meskipun jumlahnya tidak besar, namun hal ini tetap perlu diperbaiki agar klasifikasi lebih akurat. Namun, yang menarik adalah bahwa 34,373 kasus positif berhasil diklasifikasikan dengan benar. Ini menunjukkan keunggulan model dalam mengidentifikasi kasus positif.

Namun, perlu diakui adanya kecenderungan tinggi dari model untuk mengklasifikasikan kasus sebagai negatif. Hal ini bisa disebabkan oleh ketidakseimbangan kelas dalam data pelatihan atau mungkin ada bias dalam model itu sendiri. Oleh karena itu, untuk meningkatkan kualitas klasifikasi, perlu dilakukan penanganan terhadap ketidakseimbangan kelas dan evaluasi lebih lanjut terhadap model.

Sample Data:

	Actual	Predicted
0	1	1
1	0	0
2	1	1
3	0	1
4	1	1

Model Accuracy: 71.17%

Gambar 4.25 Dataframe hasil prediksi

Gambar 4.25 menampilkan sebuah dataframe yang mengilustrasikan hasil prediksi dari model *Multilayer Perceptron* (MLP) dengan menggunakan contoh data sampel yang telah disediakan. DataFrame ini memberikan gambaran tentang performa model dalam melakukan prediksi. Kolom *Actual* menunjukkan label sebenarnya dari data, sedangkan kolom *Predicted* menunjukkan label yang diprediksi oleh model. Kesesuaian antara kedua kolom ini menunjukkan sejauh mana prediksi model dapat menghasilkan hasil yang akurat.

Dalam sampel yang ditampilkan, terdapat beberapa kesalahan prediksi yang dapat dilihat dari perbedaan nilai antara kolom *Actual* dan *Predicted*. Meskipun demikian, secara keseluruhan, model MLP mencapai tingkat akurasi sebesar 71.17%. Ini berarti bahwa model berhasil memprediksi dengan benar sekitar 71.17% dari data yang diberikan. Namun, berdasarkan pengujian yang merujuk pada Tabel 4.1, yaitu hasil deteksi sistem, menunjukkan akurasi saat pengujian mencapai 100%. Oleh karena itu, akurasi ini dianggap mampu mendeteksi dan memitigasi serangan DDoS secara efektif. Dengan adanya gambaran ini, dapat disimpulkan bahwa model MLP telah memberikan hasil prediksi yang cukup baik, meskipun masih terdapat ruang untuk perbaikan.

## BAB V

### PENUTUP

#### 5.1. Kesimpulan

Berdasarkan hasil eksperimen yang telah dibahas sebelumnya, berikut ini adalah kesimpulan yang telah didapat sebagai berikut:

1. Dalam penelitian ini, model *Multilayer Perceptron* (MLP) telah berhasil dilatih dan dikembangkan untuk deteksi dan klasifikasi serangan DDoS pada jaringan SDN. Model MLP dengan akurasi sebesar 71.17% menunjukkan efisiensi dan kecepatan tinggi dalam mendeteksi serangan DDoS sehingga menunjukkan potensi besar dalam meningkatkan keamanan jaringan melalui metode *deep learning*.
2. Proses pengembangan model MLP melibatkan pemilihan dan persiapan dataset yang relevan dan representatif untuk serangan DDoS. Dataset diambil dari hasil pengumpulan data menggunakan tabel sFlow dengan *hping3*, menghasilkan 2 juta paket data yang terdiri dari lalu lintas normal dan lalu lintas DDoS. Data tersebut kemudian diratakan agar seimbang untuk menghindari bias. Hasilnya, model yang dibentuk menggunakan algoritma *RandomSearchCV* menunjukkan akurasi 71.17% dalam mengklasifikasikan serangan DDoS, memastikan efektivitas model dalam mendeteksi dan mengklasifikasikan serangan.
3. Persiapan *environment* yang tepat untuk pelatihan dan pengujian model sangat penting. Dalam penelitian ini, *environment* telah disiapkan dengan hati-hati menggunakan Ubuntu 22.04 sebagai sistem operasi, Mininet sebagai emulator jaringan, Ryu *controller* untuk pengendalian jaringan, dan Python sebagai bahasa pemrograman. Hasilnya, proses pelatihan dan pengujian dapat berjalan dengan lancar dan efisien, mendukung pengembangan dan evaluasi model MLP dengan optimal.
4. Hasil mitigasi serangan DDoS menunjukkan perbaikan signifikan dalam analisis jaringan. Berdasarkan pengukuran *jitter*, *delay*, *throughput*, *packet loss*, dan *resource utilization*, tindakan mitigasi terbukti efektif. Meskipun *packet loss* belum sepenuhnya pulih dan masih berada dalam kategori "buruk" menurut standar TIPHON, mitigasi berhasil mengurangi dampak serangan DDoS secara

keseluruhan dan memulihkan performa jaringan ke kategori "bagus" hingga "sangat bagus" pada metrik lainnya. Ini menandakan bahwa langkah-langkah mitigasi yang diterapkan berhasil meningkatkan kualitas layanan jaringan sesuai dengan standar yang diterima.

## 5.2. Saran

Berdasarkan temuan-temuan ini, peneliti merekomendasikan adanya upaya lanjutan dalam pengembangan model deteksi serangan DDoS.

1. Dalam penelitian mendatang, dapat dilakukan penelitian lebih lanjut terhadap teknik *deep learning* lainnya, seperti *Convolutional Neural Network* (CNN) atau *Recurrent Neural Network* (RNN), untuk membandingkan kinerja dan efektivitasnya dalam penanganan paket DDoS.
2. Mengumpulkan dataset yang lebih besar dan lebih beragam untuk melatih model. Dataset yang mencakup berbagai jenis serangan DDoS dan pola lalu lintas jaringan dapat meningkatkan kemampuan model dalam mendeteksi dan mengklasifikasikan serangan dengan lebih akurat.
3. Melakukan pengujian dan evaluasi model pada lingkungan jaringan nyata untuk menilai kinerja dan efektivitas model dalam kondisi dunia nyata. Hal ini penting untuk memastikan bahwa model dapat diandalkan dalam situasi sebenarnya.

## Daftar Pustaka

- [1] P. A and S. S, “DDOS ATTACK DETECTION IN TELECOMMUNICATION NETWORK USING MACHINE LEARNING,” *Journal of Ubiquitous Computing and Communication Technologies*, vol. 01, no. 01, pp. 33–44, Sep. 2019, doi: 10.36548/jucct.2019.1.004.
- [2] F. J. Abdullayeva, “Distributed denial of service attack detection in E-government cloud via data clustering,” *Array*, vol. 15, Sep. 2022, doi: 10.1016/j.array.2022.100229.
- [3] P. Khuphiran, P. Leelaprute, P. Uthayopas, K. Ichikawa, and W. Watanakeesuntorn, “Performance Comparison of Machine Learning Models for DDoS Attacks Detection,” in *2018 22nd International Computer Science and Engineering Conference (ICSEC)*, IEEE, Nov. 2018, pp. 1–4. doi: 10.1109/ICSEC.2018.8712757.
- [4] “Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper - Cisco.” Accessed: Mar. 30, 2023. [Online]. Available: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>
- [5] F. J. Abdullayeva, “Convolutional Neural Network-Based Automatic Diagnostic System for AL-DDoS Attacks Detection,” *International Journal of Cyber Warfare and Terrorism*, vol. 12, no. 1, pp. 1–15, Jul. 2022, doi: 10.4018/IJCWT.305242.
- [6] M. A. Al-Shareeda, S. Manickam, and M. A. Saare, “DDoS attacks detection using machine learning and deep learning techniques: analysis and comparison,” *Bulletin of Electrical Engineering and Informatics*, vol. 12, no. 2, pp. 930–939, Apr. 2023, doi: 10.11591/eei.v12i2.4466.
- [7] P. K. Kishore, S. Ramamoorthy, and V. N. Rajavarman, “AN IMPROVED BIO-INSPIRED BAT ALGORITHM FOR DETECTION AND PREVENTION OF HTTP FLOOD DDOS ATTACK USING MACHINE LEARNING METRICS,” 2020.
- [8] H. S. Obaid and E. H. Abeed, “Abeed,-DoS and DDoS Attacks at OSI Layers,” *International Journal of Multidisciplinary Research and Publications Hadeel S. Obaid and Esamaddin H*, vol. 2, no. 8, pp. 1–9, 2020.
- [9] M. S. Mahmoud and Y. Xia, “Distributed denial-of-service attacks,” in *Cloud Control Systems*, S. Ison, Ed., Elsevier, 2020, pp. 51–76. doi: 10.1016/B978-0-12-818701-2.00011-1.
- [10] B. Nugraha and R. N. Murthy, “Deep Learning-based Slow DDoS Attack Detection in SDN-based Networks,” in *2020 IEEE Conference on Network*

*Function Virtualization and Software Defined Networks, NFV-SDN 2020 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., Nov. 2020, pp. 51–56. doi: 10.1109/NFV-SDN50289.2020.9289894.

- [11] P. Karthika and K. Arockiasamy, “Simulation of SDN in mininet and detection of DDoS attack using machine learning,” *Bulletin of Electrical Engineering and Informatics*, vol. 12, no. 3, pp. 1797–1805, Jun. 2023, doi: 10.11591/eei.v12i3.5232.
- [12] F. Hussain, S. G. Abbas, M. Husnain, U. U. Fayyaz, F. Shahzad, and G. A. Shah, “IoT DoS and DDoS Attack Detection using ResNet,” in *Proceedings - 2020 23rd IEEE International Multi-Topic Conference, INMIC 2020*, Institute of Electrical and Electronics Engineers Inc., Nov. 2020. doi: 10.1109/INMIC50486.2020.9318216.
- [13] H. Majed, H. N. Noura, O. Salman, M. Malli, and A. Chehab, “Efficient and secure statistical DDoS detection scheme,” in *ICETE 2020 - Proceedings of the 17th International Joint Conference on e-Business and Telecommunications*, SciTePress, 2020, pp. 153–161. doi: 10.5220/0009873801530161.
- [14] S. Hosseini and M. Azizi, “The hybrid technique for DDoS detection with supervised learning algorithms,” *Computer Networks*, vol. 158, pp. 35–45, Jul. 2019, doi: 10.1016/j.comnet.2019.04.027.
- [15] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, “A survey on deep learning for big data,” Jul. 01, 2018, *Elsevier B.V.* doi: 10.1016/j.inffus.2017.10.006.
- [16] S. Zeadally, E. Adi, Z. Baig, and I. A. Khan, “Harnessing artificial intelligence capabilities to improve cybersecurity,” *IEEE Access*, vol. 8, pp. 23817–23837, 2020, doi: 10.1109/ACCESS.2020.2968045.
- [17] A. Aljuhani, “Machine Learning Approaches for Combating Distributed Denial of Service Attacks in Modern Networking Environments,” *IEEE Access*, vol. 9, pp. 42236–42264, 2021, doi: 10.1109/ACCESS.2021.3062909.
- [18] Naveen Bindra and Manu Sood, “Detecting DDoS Attacks Using Machine Learning Techniques and Contemporary Intrusion Detection Dataset,” *Automatic Control and Computer Sciences*, vol. 53, no. 5, pp. 419–428, Sep. 2019, doi: 10.3103/S0146411619050043.
- [19] A. R. Shaaban, E. Abd-Elwanis, and M. Hussein, “DDoS attack detection and classification via Convolutional Neural Network (CNN),” *2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS)*.

- [20] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, “Deep Learning for Computer Vision: A Brief Review,” 2018, *Hindawi Limited*. doi: 10.1155/2018/7068349.
- [21] *2018 International Conference on Computing, Power and Communication Technologies (GUCON)*. IEEE, 2018.
- [22] P. P. Shinde, *A Review of Machine Learning and Deep Learning Applications*.
- [23] C. Janiesch, P. Zschech, and K. Heinrich, “Machine learning and deep learning”, doi: 10.1007/s12525-021-00475-2/Published.
- [24] T. Thomas, A. P. Vijayaraghavan, and S. Emmanuel, *Machine Learning Approaches in Cyber Security Analytics*. Singapore: Springer Singapore, 2020. doi: 10.1007/978-981-15-1706-8.
- [25] C. Janiesch, P. Zschech, and K. Heinrich, “Machine learning and deep learning,” *The International Journal On Networked Bussines*, vol. 31, pp. 685–695, 2021, doi: 10.1007/s12525-021-00475-2/Published.
- [26] N. Buduma and N. Locascio, “Deep Learning DESIGNING NEXT-GENERATION MACHINE INTELLIGENCE ALGORITHMS Nikhil Buduma with contributions by Nicholas Locascio,” 2017.
- [27] L. L. Minku, G. Cabral, M. Martins, and M. Wagner, “Introduction to Computational Intelligence,” vol. 1, pp. 105–110, 2023, doi: 10.5281/zenodo.7537827.
- [28] E. Bisong, *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Apress, 2019. doi: 10.1007/978-1-4842-4470-8.
- [29] K. Shaukat, S. Luo, V. Varadharajan, I. A. Hameed, and M. Xu, “A Survey on Machine Learning Techniques for Cyber Security in the Last Decade,” *IEEE Access*, vol. 8, pp. 222310–222354, 2020, doi: 10.1109/ACCESS.2020.3041951.
- [30] G. Howser, *Computer Networks and the Internet*. Cham: Springer International Publishing, 2020. doi: 10.1007/978-3-030-34496-2.
- [31] C. Panek, *Networking fundamentals*. Canada: John Wiley & Sons, Inc, 2020.
- [32] L. Yang, B. Ng, W. K. G. Seah, L. Groves, and D. Singh, “A survey on network forwarding in Software-Defined Networking,” Feb. 15, 2021, *Academic Press*. doi: 10.1016/j.jnca.2020.102947.
- [33] S. Saraswat, V. Agarwal, H. P. Gupta, R. Mishra, A. Gupta, and T. Dutta, “Challenges and solutions in Software Defined Networking: A survey,” *Journal of Network and Computer Applications*, vol. 141, pp. 23–58, Sep. 2019, doi: 10.1016/j.jnca.2019.04.020.

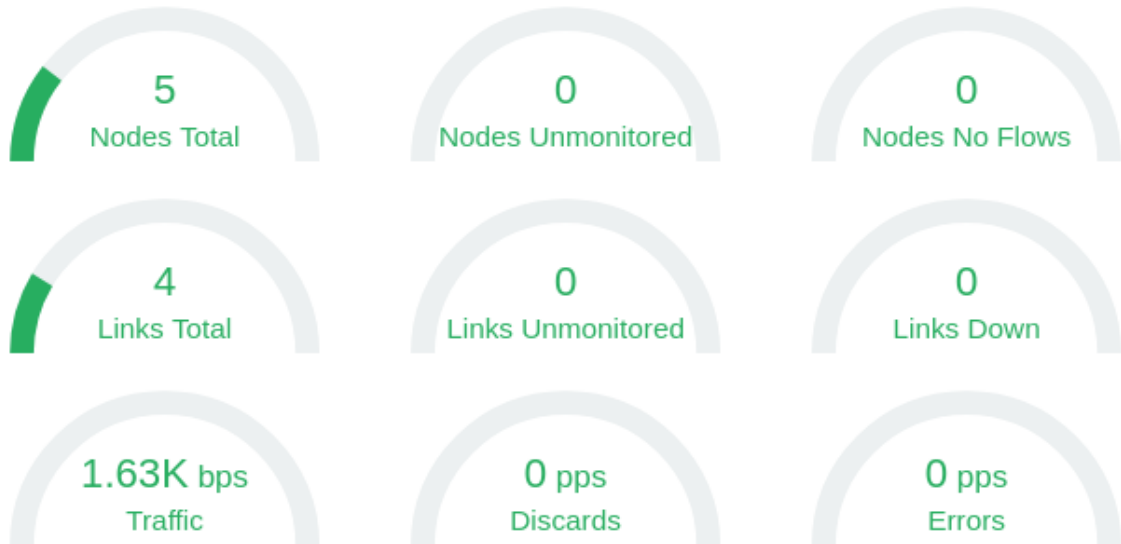


- [34] P. Ferdiansyah and U. Amikom Yogyakarta, “Analisis Perbandingan Parameter QoS Standar TIPHON Pada Jaringan Nirkabel Dalam Penerapan Metode PCQ,” 2022.
- [35] R. A. R. Q. Y. Putri, Anhar, and A. Al Nazen, “Analysis of LTE Network Quality of Service on Streaming Application,” *International Joournal of Electrical, Energy and Power System Engineering (IJEPPSE)*, vol. 6, no. 2, pp. 151–155, 2023.
- [36] M. Snehi and A. Bhandari, “Vulnerability retrospection of security solutions for software-defined Cyber-Physical System against DDoS and IoT-DDoS attacks,” May 01, 2021, *Elsevier Ireland Ltd.* doi: 10.1016/j.cosrev.2021.100371.
- [37] Y. Li and Q. Liu, “A comprehensive review study of cyber-attacks and cyber security; Emerging trends and recent developments,” *Energy Reports*, vol. 7, pp. 8176–8186, Nov. 2021, doi: 10.1016/j.egyr.2021.08.126.
- [38] A. M. Abdul and S. Umar, “Attacks of Denial-of-Service on Networks Layer of OSI Model and Maintaining of Security,” *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 5, no. 1, pp. 181–186, 2017, doi: 10.11591/ijeecs.v5.i1.pp.
- [39] S. Pande, A. Khamparia, D. Gupta, and D. N. H. Thanh, “DDOS Detection Using Machine Learning Technique,” 2021, pp. 59–68. doi: 10.1007/978-981-15-8469-5\_5.
- [40] F. Hussain, S. G. Abbas, M. Husnain, U. U. Fayyaz, F. Shahzad, and G. A. Shah, “IoT DoS and DDoS Attack Detection using ResNet,” in *Proceedings - 2020 23rd IEEE International Multi-Topic Conference, INMIC 2020*, Institute of Electrical and Electronics Engineers Inc., Nov. 2020. doi: 10.1109/INMIC50486.2020.9318216.
- [41] B. A. Khalaf, S. A. Mostafa, A. Mustapha, M. A. Mohammed, and W. M. Abdulllah, “Comprehensive review of artificial intelligence and statistical approaches in distributed denial of service attack and defense methods,” *IEEE Access*, vol. 7, pp. 51691–51713, 2019, doi: 10.1109/ACCESS.2019.2908998.
- [42] A. Bhati, A. Bouras, U. Ahmed Qidwai, and A. Belhi, “Deep learning based identification of DDoS attacks in industrial application,” in *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, IEEE, Jul. 2020, pp. 190–196. doi: 10.1109/WorldS450073.2020.9210320.
- [43] “Hacktivists step back giving way to professionals: a look at DDoS in Q3 2022 | Kaspersky.” Accessed: May 03, 2023. [Online]. Available: [https://www.kaspersky.com/about/press-releases/2022\\_hacktivists-step-back-giving-way-to-professionals-a-look-at-ddos-in-q3-2022](https://www.kaspersky.com/about/press-releases/2022_hacktivists-step-back-giving-way-to-professionals-a-look-at-ddos-in-q3-2022)

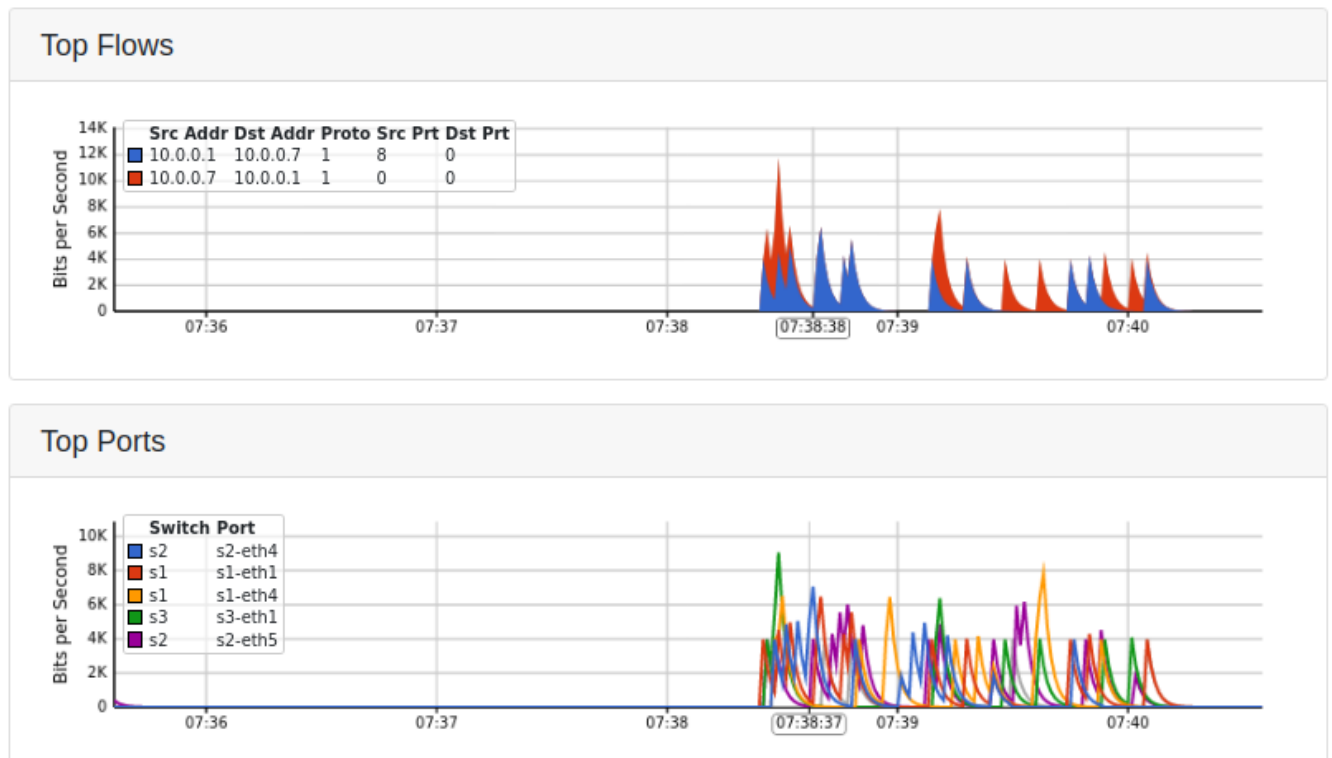
- [44] J. A. Perez-Diaz, I. A. Valdovinos, K.-K. R. Choo, and D. Zhu, "A Flexible SDN-Based Architecture for Identifying and Mitigating Low-Rate DDoS Attacks Using Machine Learning," *IEEE Access*, vol. 8, pp. 155859–155872, 2020, doi: 10.1109/ACCESS.2020.3019330.
- [45] S. H. Islam, P. Vijayakumar, M. Z. A. Bhuiyan, R. Amin, V. Rajeev M., and B. Balusamy, "A Provably Secure Three-Factor Session Initiation Protocol for Multimedia Big Data Communications," *IEEE Internet Things J*, vol. 5, no. 5, pp. 3408–3418, Oct. 2018, doi: 10.1109/JIOT.2017.2739921.
- [46] A. R. Shaaban, E. Abd-Elwanis, and M. Hussein, "DDoS attack detection and classification via Convolutional Neural Network (CNN)," in *2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS)*, IEEE, Dec. 2019, pp. 233–238. doi: 10.1109/ICICIS46948.2019.9014826.
- [47] M. S. Khaing, Y. M. Thant, T. Tun, C. S. Htwe, and M. M. S. Thwin, "IoT Botnet Detection Mechanism Based on UDP Protocol," in *2020 IEEE Conference on Computer Applications (ICCA)*, IEEE, Feb. 2020, pp. 1–7. doi: 10.1109/ICCA49400.2020.9022832.
- [48] "Top 10 Python Libraries untuk Machine Learning - Algoritma." Accessed: Apr. 27, 2023. [Online]. Available: <https://algorit.ma/blog/python-libraries-machine-learning-2022/>
- [49] N. K. Manaswi, *Deep Learning with Applications Using Python*. Berkeley, CA: Apress, 2018. doi: 10.1007/978-1-4842-3516-4.
- [50] Y. Cui *et al.*, "Towards DDoS detection mechanisms in Software-Defined Networking," Sep. 15, 2021, *Academic Press*. doi: 10.1016/j.jnca.2021.103156.
- [51] Institute of Electrical and Electronics Engineers, *2020 IEEE International Conference on Communications Workshops (ICC) : proceedings : Dublin, Ireland, 7-11 June 2020*. 2020.
- [52] R. M. A. Ujjan, Z. Pervez, K. Dahal, A. K. Bashir, R. Mumtaz, and J. González, "Towards sFlow and adaptive polling sampling for deep learning based DDoS detection in SDN," *Future Generation Computer Systems*, vol. 111, pp. 763–779, Oct. 2020, doi: 10.1016/j.future.2019.10.015.

## LAMPIRAN A HASIL SIMULASI PERCOBAAN

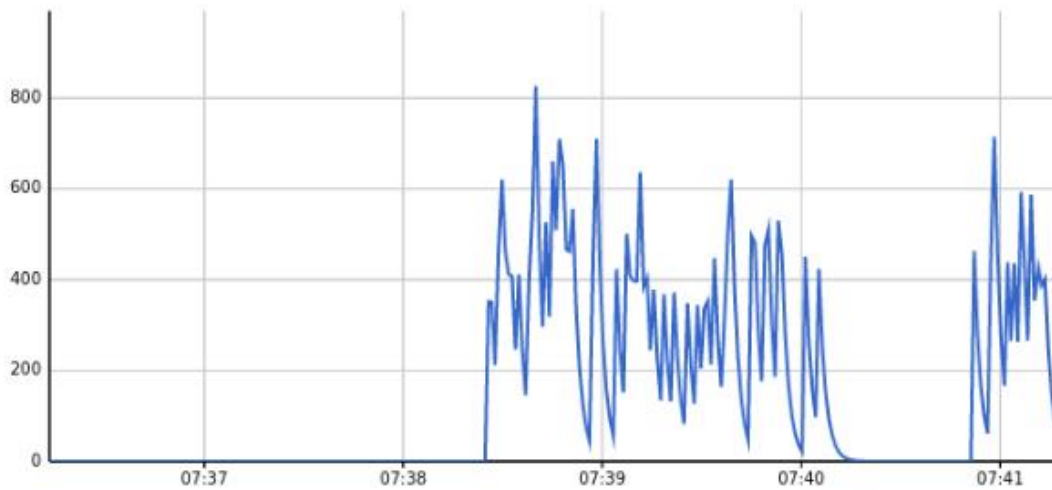
### A-1 Lalu Lintas Normal



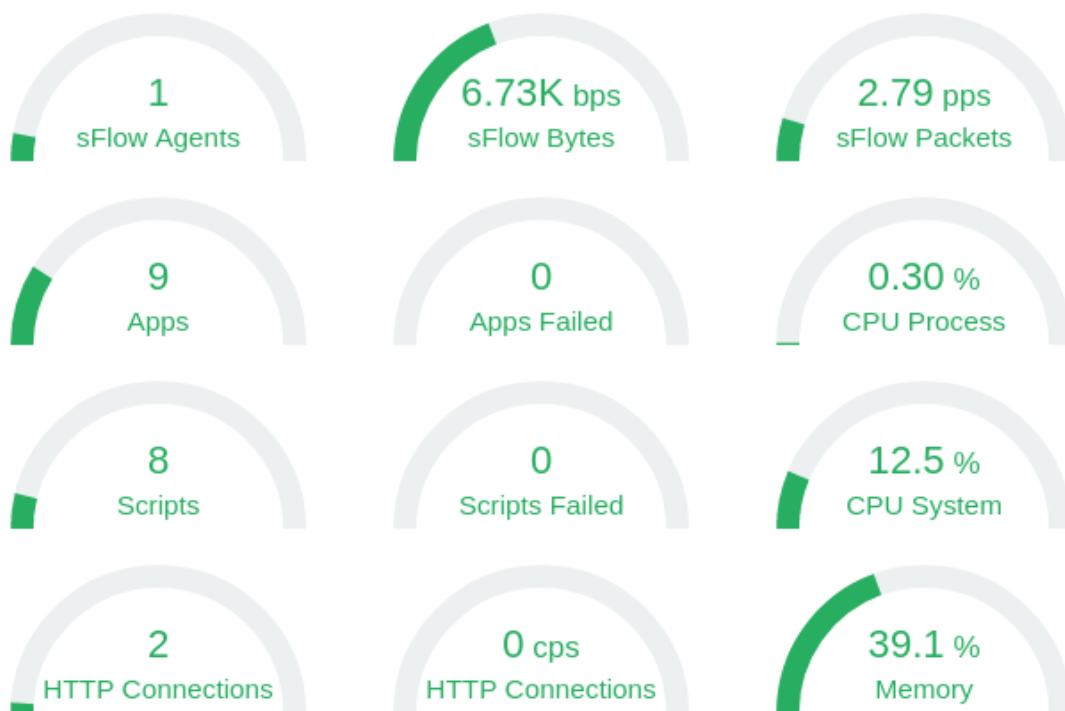
Gambar Lampiran 1.1 Dashboard Sflow RT Pada Lalu Lintas Normal



Gambar Lampiran 1.2 Flow dan Port Saat Lalu lintas Normal



Gambar Lampiran 1.3 Jumlah Flow saat Lalu Lintas Normal



Gambar Lampiran 1.4 Resource Saat Lalu Lintas Normal

```
"Node: h7"
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Mitigation V5# iperf -c 10.0.0.1
-----
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 1] local 10.0.0.7 port 41774 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-10.0076 sec 6.24 GBytes 5.36 Gbits/sec
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Mitigation V5#
```

Gambar Lampiran 1.5 Throughput Lalu Lintas Normal

```
"Node: h1"
64 bytes from 10.0.0.7: icmp_seq=82 ttl=64 time=0.052 ms
64 bytes from 10.0.0.7: icmp_seq=83 ttl=64 time=0.067 ms
64 bytes from 10.0.0.7: icmp_seq=84 ttl=64 time=0.102 ms
64 bytes from 10.0.0.7: icmp_seq=85 ttl=64 time=0.065 ms
64 bytes from 10.0.0.7: icmp_seq=86 ttl=64 time=0.065 ms
64 bytes from 10.0.0.7: icmp_seq=87 ttl=64 time=0.139 ms
64 bytes from 10.0.0.7: icmp_seq=88 ttl=64 time=0.084 ms
64 bytes from 10.0.0.7: icmp_seq=89 ttl=64 time=0.126 ms
64 bytes from 10.0.0.7: icmp_seq=90 ttl=64 time=0.174 ms
64 bytes from 10.0.0.7: icmp_seq=91 ttl=64 time=0.188 ms
64 bytes from 10.0.0.7: icmp_seq=92 ttl=64 time=0.110 ms
64 bytes from 10.0.0.7: icmp_seq=93 ttl=64 time=0.071 ms
64 bytes from 10.0.0.7: icmp_seq=94 ttl=64 time=0.128 ms
64 bytes from 10.0.0.7: icmp_seq=95 ttl=64 time=0.070 ms
64 bytes from 10.0.0.7: icmp_seq=96 ttl=64 time=0.233 ms
64 bytes from 10.0.0.7: icmp_seq=97 ttl=64 time=0.057 ms
64 bytes from 10.0.0.7: icmp_seq=98 ttl=64 time=0.174 ms
64 bytes from 10.0.0.7: icmp_seq=99 ttl=64 time=0.126 ms
64 bytes from 10.0.0.7: icmp_seq=100 ttl=64 time=8.50 ms

--- 10.0.0.7 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 101332ms
rtt min/avg/max/mdev = 0.052/0.280/8.495/1.100 ms
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Mitigation V5#
```

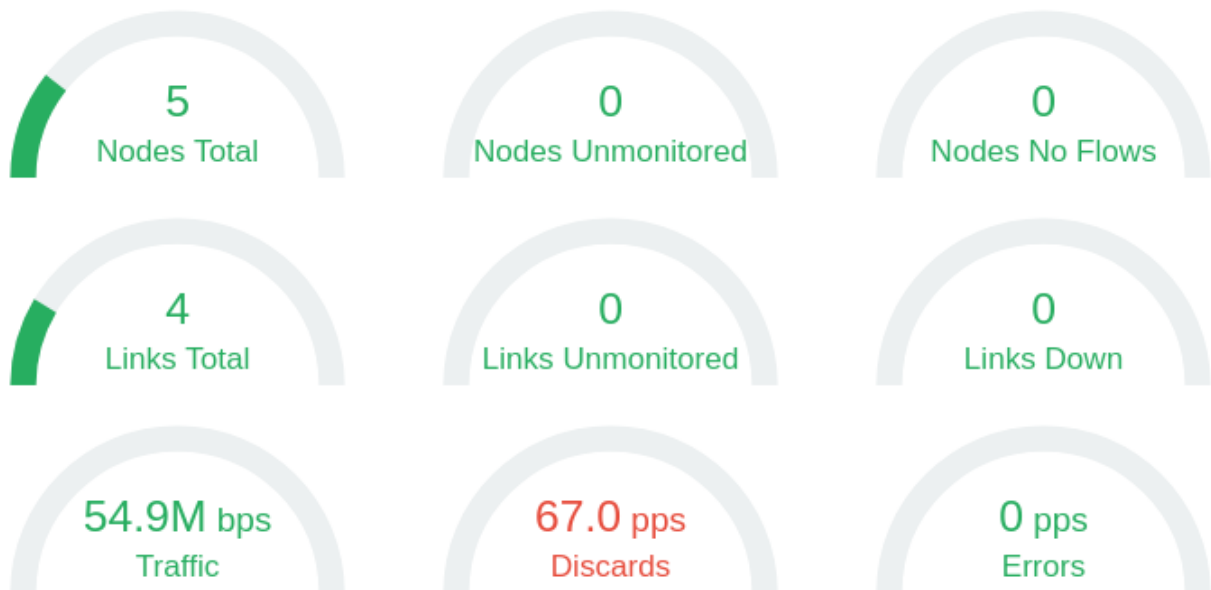
Gambar Lampiran 1.6 Packet Loss dan Latensi Saat Lalu Lintas Normal

```
Terminal
Traffic is mostly Normal.
-----
Traffic Analysis:
Normal traffic count: 264
DDoS traffic count: 0
Total traffic analyzed: 264
Traffic is mostly Normal.
-----
Traffic Analysis:
Normal traffic count: 2
DDoS traffic count: 0
Total traffic analyzed: 2
Traffic is mostly Normal.
-----
Traffic Analysis:
Normal traffic count: 2
DDoS traffic count: 0
Total traffic analyzed: 2
Traffic is mostly Normal.
-----
```

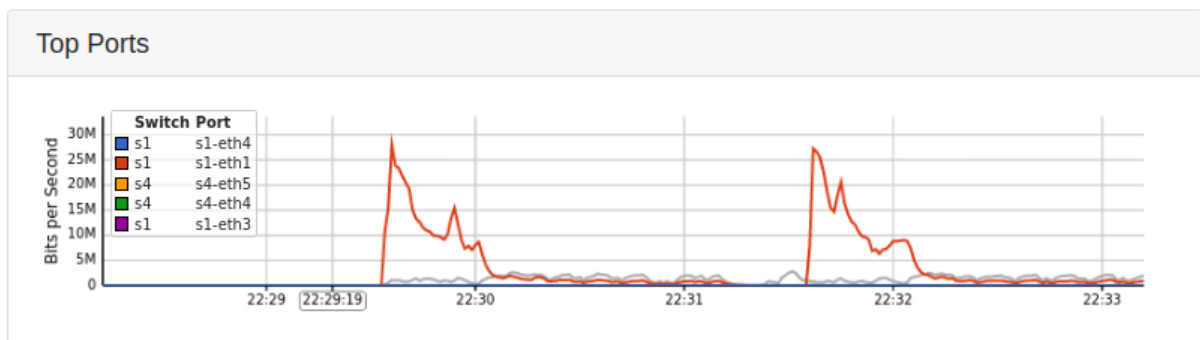
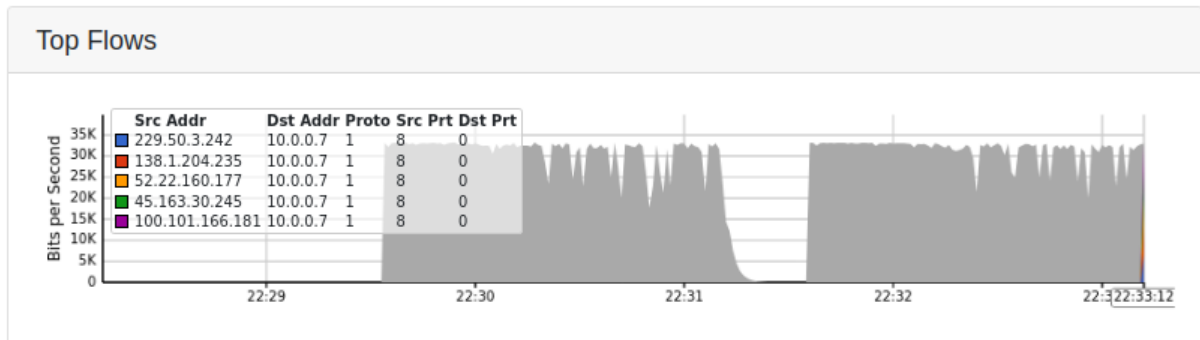
Gambar Lampiran 1.7 Hasil Deteksi Lalu Lintas Normal

## A-2 Lalu Lintas DDoS Selama DDoS Terjadi

### 1. ICMP Flood



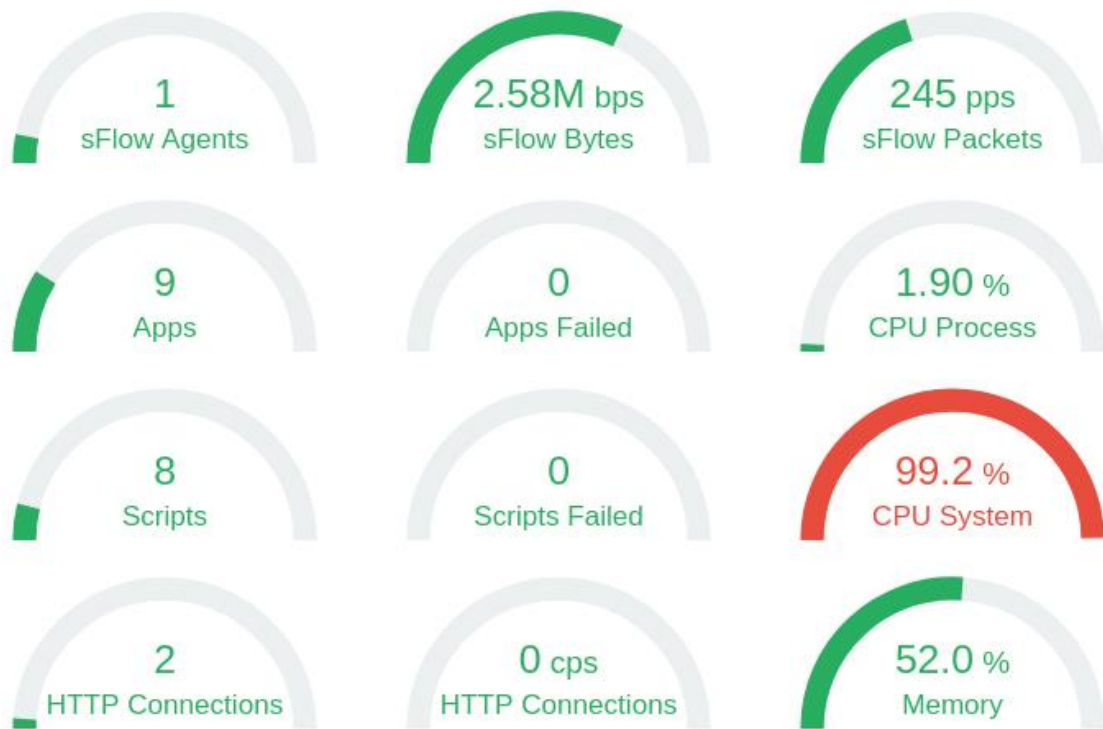
Gambar Lampiran 2.1 Dashboard Sflow RT Pada Lalu Lintas ICMP Flood Selama DDoS Terjadi



Gambar Lampiran 2.2 Flow dan Port Saat Lalu lintas ICMP Flood Selama DDoS Terjadi



Gambar Lampiran 2.3 Jumlah Flow saat Lalu Lintas ICMP Flood Selama DDoS Terjadi



Gambar Lampiran 2.4 Resource Saat Lalu Lintas ICMP Flood Selama DDoS Terjadi

```

"Node: h10"
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Detection# iperf -c 10.0.0.2
tcp connect failed: No route to host
-----
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: -1.00 Byte (default)
-----
[ 1] local 0.0.0.0 port 0 connected with 10.0.0.2 port 5001
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Detection#

```

Gambar Lampiran 2.5 Throughput Lalu Lintas ICMP Flood Selama DDoS Terjadi



```
"Node: h10"
From 10.0.0.10 icmp_seq=54 Destination Host Unreachable
From 10.0.0.10 icmp_seq=55 Destination Host Unreachable
From 10.0.0.10 icmp_seq=56 Destination Host Unreachable
From 10.0.0.10 icmp_seq=57 Destination Host Unreachable
From 10.0.0.10 icmp_seq=58 Destination Host Unreachable
From 10.0.0.10 icmp_seq=59 Destination Host Unreachable
From 10.0.0.10 icmp_seq=60 Destination Host Unreachable
From 10.0.0.10 icmp_seq=61 Destination Host Unreachable
From 10.0.0.10 icmp_seq=62 Destination Host Unreachable
From 10.0.0.10 icmp_seq=63 Destination Host Unreachable
From 10.0.0.10 icmp_seq=64 Destination Host Unreachable
From 10.0.0.10 icmp_seq=65 Destination Host Unreachable
From 10.0.0.10 icmp_seq=66 Destination Host Unreachable
From 10.0.0.10 icmp_seq=67 Destination Host Unreachable
From 10.0.0.10 icmp_seq=68 Destination Host Unreachable
From 10.0.0.10 icmp_seq=69 Destination Host Unreachable
From 10.0.0.10 icmp_seq=70 Destination Host Unreachable
From 10.0.0.10 icmp_seq=71 Destination Host Unreachable
From 10.0.0.10 icmp_seq=72 Destination Host Unreachable

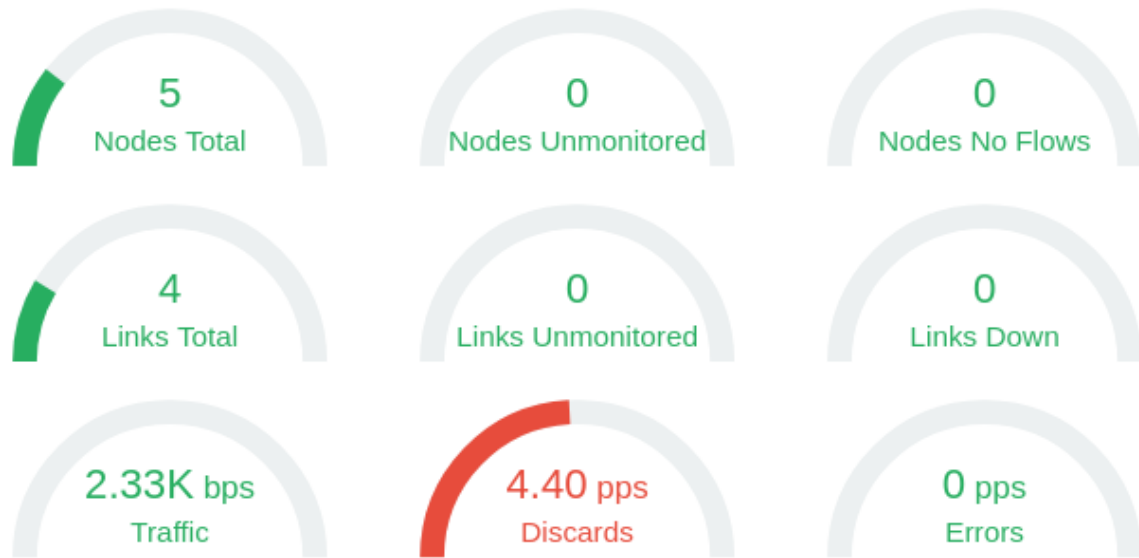
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 0 received, +72 errors, 100% packet loss, time 101358ms
pipe 4
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Detection#
```

Gambar Lampiran 2.6 Packet Loss dan Latensi Saat Lalu Lintas ICMP Flood Selama DDoS Terjadi

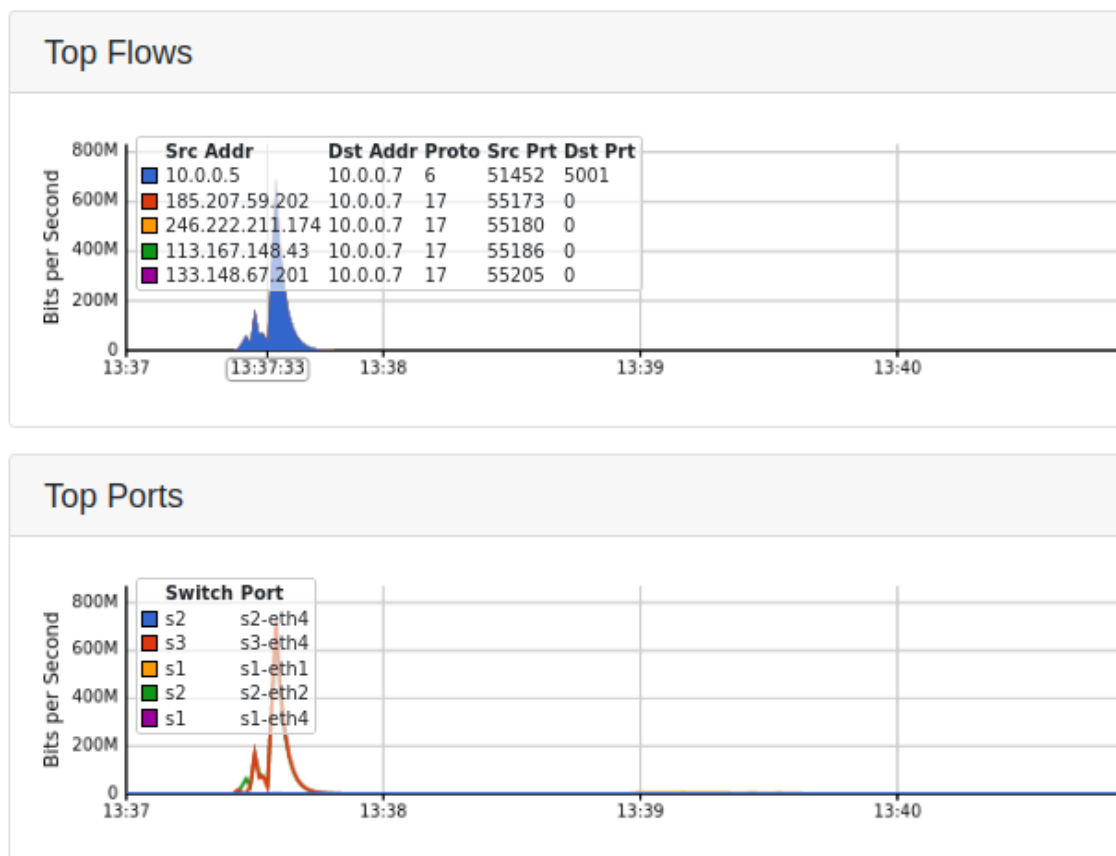
```
Terminal
Terminal
Traffic Analysis:
Normal traffic count: 0
DDoS traffic count: 39
Total traffic analyzed: 39
DDoS traffic detected.
Potential victim host: h7
-----
Traffic Analysis:
Normal traffic count: 0
DDoS traffic count: 190
Total traffic analyzed: 190
DDoS traffic detected.
Potential victim host: h7
-----
Traffic Analysis:
Normal traffic count: 0
DDoS traffic count: 103
Total traffic analyzed: 103
DDoS traffic detected.
Potential victim host: h7
-----
```

Gambar Lampiran 2.7 Hasil Deteksi Lalu Lintas ICMP Flood Selama DDoS Terjadi

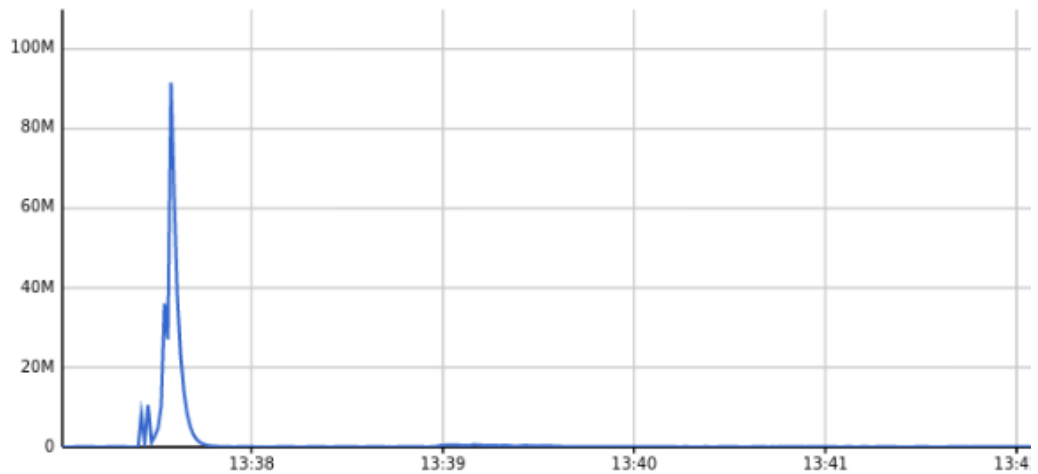
## 2. UDP Flood



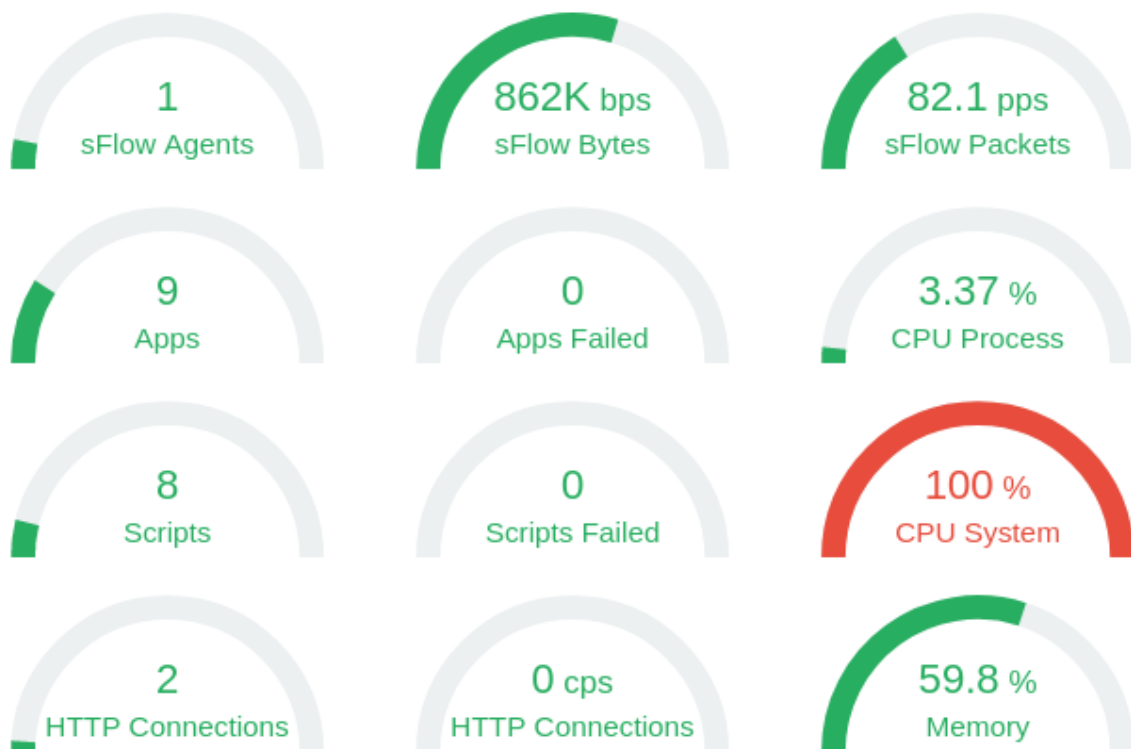
Gambar Lampiran 3.1 Dashboard Sflow RT Pada Lalu Lintas UDP Flood Selama DDoS Terjadi



Gambar Lampiran 3.2 Flow dan Port Saat Lalu lintas UDP Flood Selama DDoS Terjadi



Gambar Lampiran 3.3 Jumlah Flow saat Lalu Lintas UDP Flood Selama DDoS Terjadi



Gambar Lampiran 3.4 Resource Saat Lalu Lintas UDP Flood Selama DDoS Terjadi

```
"Node: h10"
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Detection V3# iperf -c 10.0.0.2
tcp connect failed: No route to host
-----
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: -1.00 Byte (default)
-----
[ 1] local 0.0.0.0 port 0 connected with 10.0.0.2 port 5001
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Detection V3#
```

Gambar Lampiran 3.5 Throughput Lalu Lintas UDP Flood Selama DDoS Terjadi

```
"Node: h10"
From 10.0.0.10 icmp_seq=82 Destination Host Unreachable
From 10.0.0.10 icmp_seq=83 Destination Host Unreachable
From 10.0.0.10 icmp_seq=84 Destination Host Unreachable
From 10.0.0.10 icmp_seq=85 Destination Host Unreachable
From 10.0.0.10 icmp_seq=86 Destination Host Unreachable
From 10.0.0.10 icmp_seq=87 Destination Host Unreachable
From 10.0.0.10 icmp_seq=88 Destination Host Unreachable
From 10.0.0.10 icmp_seq=89 Destination Host Unreachable
From 10.0.0.10 icmp_seq=90 Destination Host Unreachable
From 10.0.0.10 icmp_seq=91 Destination Host Unreachable
From 10.0.0.10 icmp_seq=92 Destination Host Unreachable
From 10.0.0.10 icmp_seq=93 Destination Host Unreachable
From 10.0.0.10 icmp_seq=94 Destination Host Unreachable
From 10.0.0.10 icmp_seq=95 Destination Host Unreachable
From 10.0.0.10 icmp_seq=96 Destination Host Unreachable
From 10.0.0.10 icmp_seq=97 Destination Host Unreachable
From 10.0.0.10 icmp_seq=98 Destination Host Unreachable
From 10.0.0.10 icmp_seq=99 Destination Host Unreachable
From 10.0.0.10 icmp_seq=100 Destination Host Unreachable

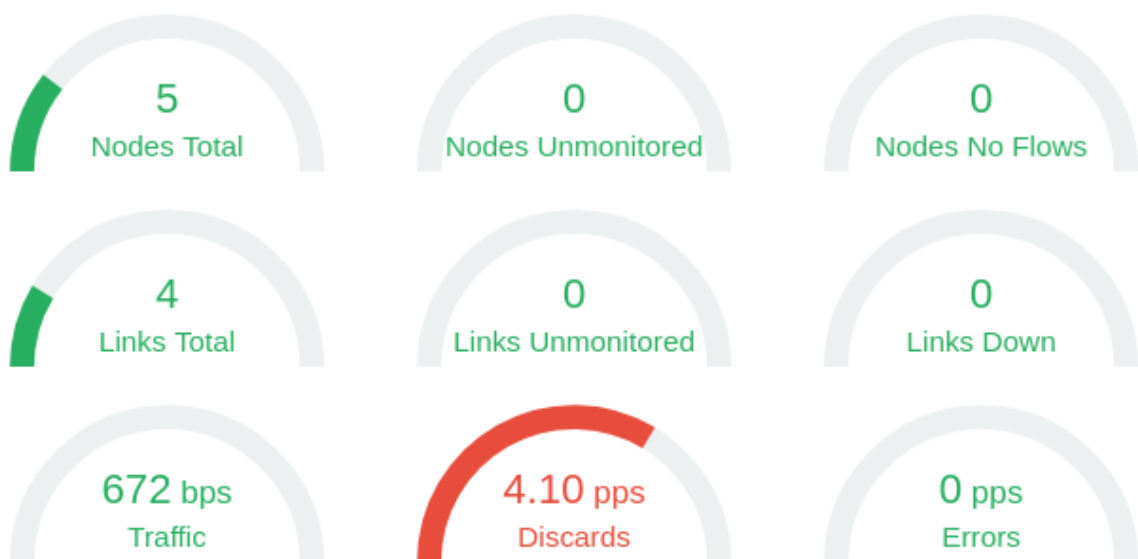
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 0 received, +69 errors, 100% packet loss, time 101370ms
pipe 4
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Detection V3#
```

Gambar Lampiran 3.6 Packet Loss dan Latensi Saat Lalu Lintas UDP Flood Selama DDoS Terjadi

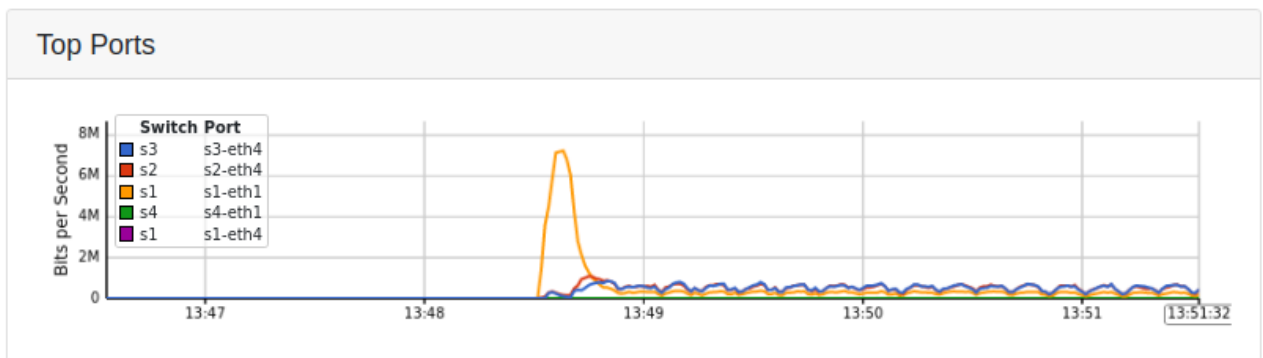
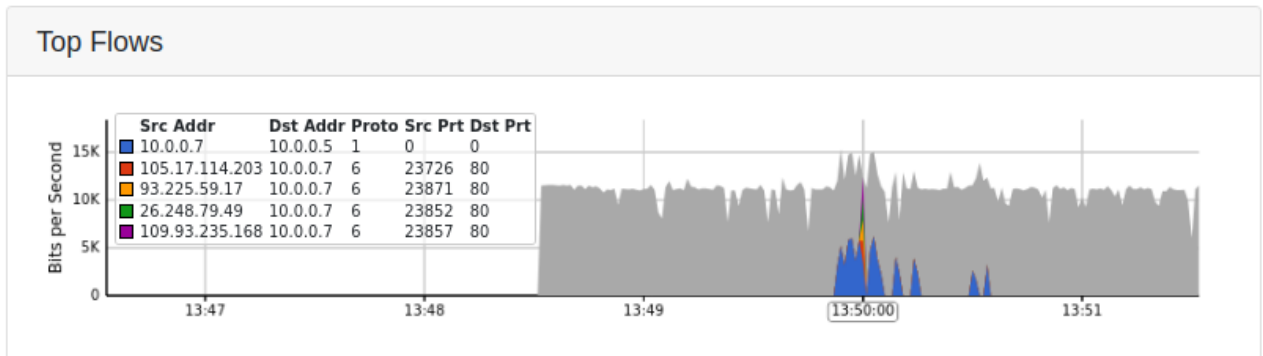
```
Terminal
Terminal x Terminal x v
Traffic Analysis:
Normal traffic count: 0
DDoS traffic count: 91
Total traffic analyzed: 91
DDoS traffic detected.
Potential victim host: h7
-----
Traffic Analysis:
Normal traffic count: 0
DDoS traffic count: 440
Total traffic analyzed: 440
DDoS traffic detected.
Potential victim host: h7
-----
Traffic Analysis:
Normal traffic count: 0
DDoS traffic count: 33
Total traffic analyzed: 33
DDoS traffic detected.
Potential victim host: h7
-----
```

Gambar Lampiran 3.7 Hasil Deteksi Lalu Lintas UDP Flood Selama DDoS Terjadi

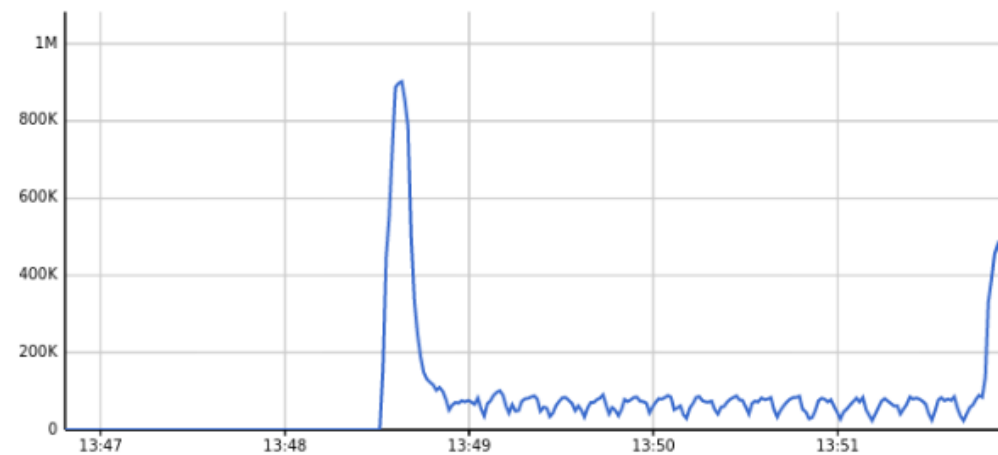
### 3. TCP SYN Flood



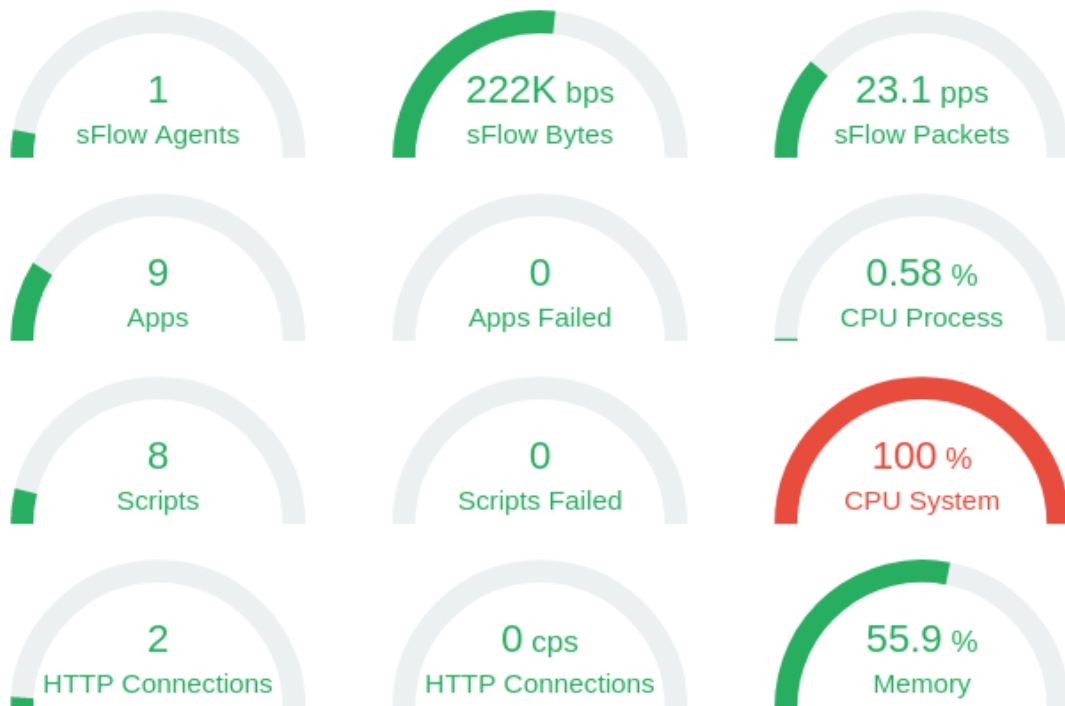
Gambar Lampiran 4.1 Dashboard Sflow RT Pada Lalu Lintas TCP SYN Flood Selama DDoS Terjadi



Gambar Lampiran 4.2 Flow dan Port Saat Lalu lintas TCP SYN Flood Selama DDoS Terjadi



Gambar Lampiran 4.3 Jumlah Flow saat Lalu Lintas TCP SYN Flood Selama DDoS Terjadi



Gambar Lampiran 4.4 Resource Saat Lalu Lintas TCP SYN Flood Selama DDoS Terjadi

```

"Node: h10"
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Detection V3# iperf -c 10.0
.0.2
tcp connect failed: No route to host
-----
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: -1.00 Byte (default)
-----
[ 1] local 0.0.0.0 port 0 connected with 10.0.0.2 port 5001
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Detection V3#

```

Gambar Lampiran 4.5 Throughput Lalu Lintas TCP SYN Flood Selama DDoS Terjadi

```
"Node: h10"
From 10.0.0.10 icmp_seq=81 Destination Host Unreachable
From 10.0.0.10 icmp_seq=82 Destination Host Unreachable
From 10.0.0.10 icmp_seq=83 Destination Host Unreachable
From 10.0.0.10 icmp_seq=84 Destination Host Unreachable
From 10.0.0.10 icmp_seq=85 Destination Host Unreachable
From 10.0.0.10 icmp_seq=86 Destination Host Unreachable
From 10.0.0.10 icmp_seq=87 Destination Host Unreachable
From 10.0.0.10 icmp_seq=88 Destination Host Unreachable
From 10.0.0.10 icmp_seq=89 Destination Host Unreachable
From 10.0.0.10 icmp_seq=90 Destination Host Unreachable
From 10.0.0.10 icmp_seq=91 Destination Host Unreachable
From 10.0.0.10 icmp_seq=92 Destination Host Unreachable
From 10.0.0.10 icmp_seq=93 Destination Host Unreachable
From 10.0.0.10 icmp_seq=94 Destination Host Unreachable
From 10.0.0.10 icmp_seq=95 Destination Host Unreachable
From 10.0.0.10 icmp_seq=96 Destination Host Unreachable
From 10.0.0.10 icmp_seq=97 Destination Host Unreachable
From 10.0.0.10 icmp_seq=98 Destination Host Unreachable
From 10.0.0.10 icmp_seq=99 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
100 packets transmitted, 0 received, +99 errors, 100% packet loss, time 101368ms
pipe 4
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Detection V3#
```

Gambar Lampiran 4.6 Packet Loss dan Latensi Saat Lalu Lintas TCP SYN Flood Selama DDoS Terjadi

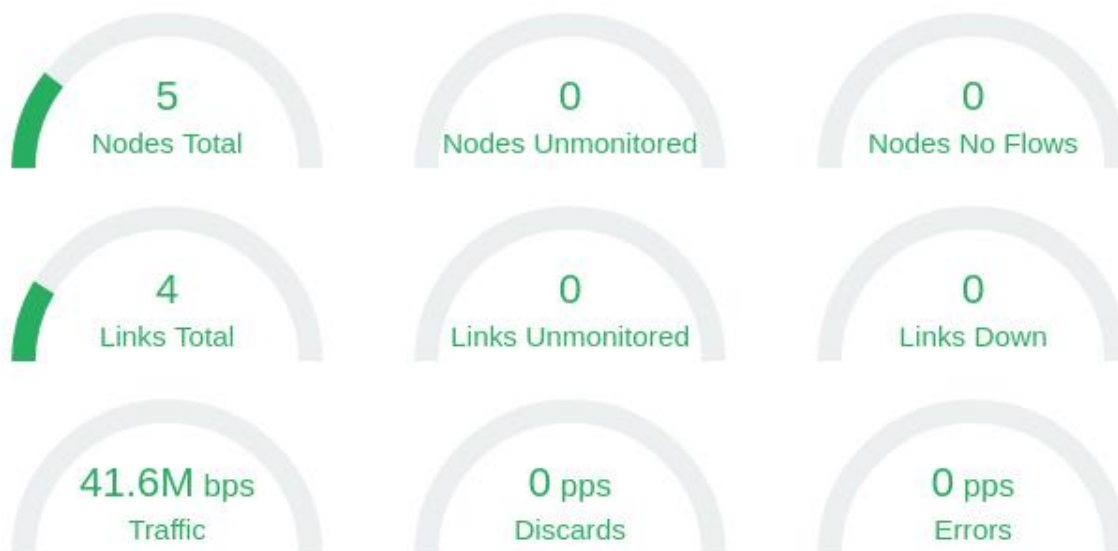
```
Terminal
Traffic Analysis:
Normal traffic count: 0
DDoS traffic count: 41
Total traffic analyzed: 41
DDoS traffic detected.
Potential victim host: h7
-----
Traffic Analysis:
Normal traffic count: 0
DDoS traffic count: 321
Total traffic analyzed: 321
DDoS traffic detected.
Potential victim host: h7
-----
Traffic Analysis:
Normal traffic count: 0
DDoS traffic count: 199
Total traffic analyzed: 199
DDoS traffic detected.
Potential victim host: h7
-----
```

Gambar Lampiran 4.7 Hasil Deteksi Lalu Lintas TCP SYN Flood Selama DDoS Terjadi

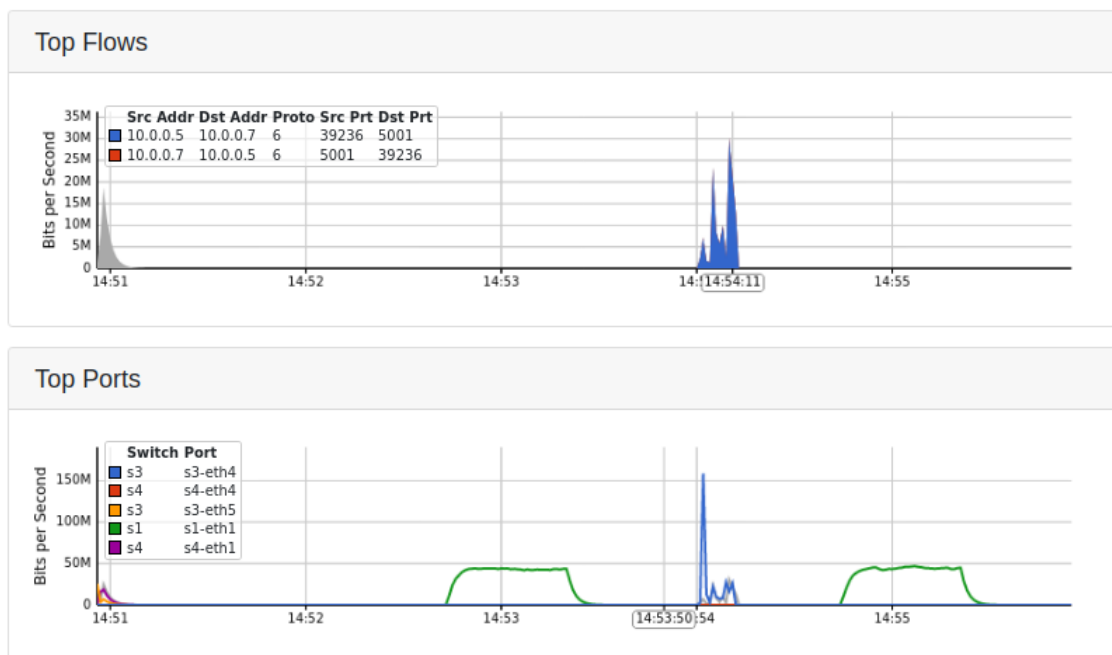


### A-3 Lalu Lintas DDoS Setelah Mitigasi

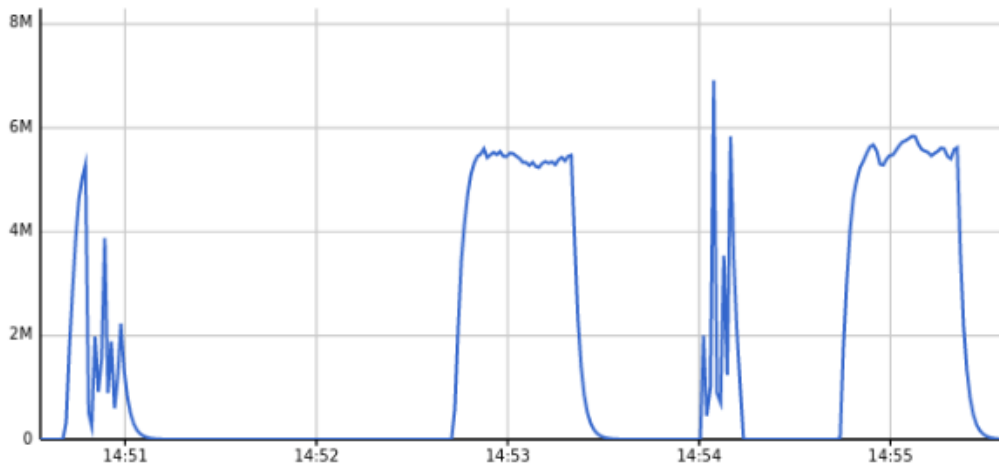
#### 1. ICMP Flood



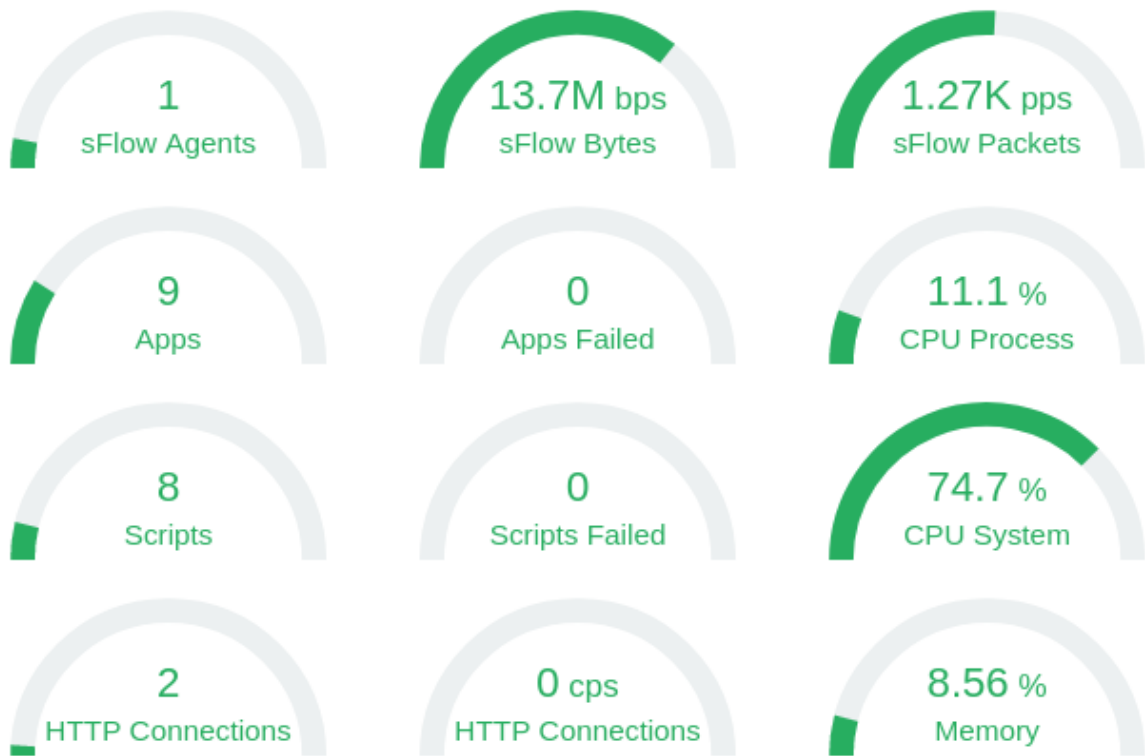
Gambar Lampiran 5.1 Dashboard Sflow RT Pada Lalu Lintas ICMP Flood Setelah Mitigasi



Gambar Lampiran 5.2 Flow dan Port Saat Lalu lintas ICMP Flood Setelah Mitigasi



Gambar Lampiran 5.3 Jumlah Flow saat Lalu Lintas ICMP Flood Setelah Mitigasi



Gambar Lampiran 5.4 Resource Saat Lalu Lintas ICMP Flood Setelah Mitigasi

```

"Node: h10"
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Mitigation V8# iperf -c 10.0.0.2
-----
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 1] local 10.0.0.10 port 51384 connected with 10.0.0.2 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 1] 0.0000-10.0101 sec 3.75 GBytes 3.21 Gbits/sec
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Mitigation V8#

```

Gambar Lampiran 5.5 Throughput Lalu Lintas ICMP Flood Setelah Mitigasi

```

"Node: h10"
64 bytes from 10.0.0.2: icmp_seq=82 ttl=64 time=0.063 ms
64 bytes from 10.0.0.2: icmp_seq=83 ttl=64 time=0.053 ms
64 bytes from 10.0.0.2: icmp_seq=84 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=85 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=86 ttl=64 time=0.041 ms
64 bytes from 10.0.0.2: icmp_seq=87 ttl=64 time=0.035 ms
64 bytes from 10.0.0.2: icmp_seq=88 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=89 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=90 ttl=64 time=0.074 ms
64 bytes from 10.0.0.2: icmp_seq=91 ttl=64 time=0.080 ms
64 bytes from 10.0.0.2: icmp_seq=92 ttl=64 time=0.056 ms
64 bytes from 10.0.0.2: icmp_seq=93 ttl=64 time=0.063 ms
64 bytes from 10.0.0.2: icmp_seq=94 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=95 ttl=64 time=0.094 ms
64 bytes from 10.0.0.2: icmp_seq=96 ttl=64 time=0.103 ms
64 bytes from 10.0.0.2: icmp_seq=97 ttl=64 time=0.132 ms
64 bytes from 10.0.0.2: icmp_seq=98 ttl=64 time=0.066 ms
64 bytes from 10.0.0.2: icmp_seq=99 ttl=64 time=7.97 ms
64 bytes from 10.0.0.2: icmp_seq=100 ttl=64 time=0.066 ms

--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 101295ms
rtt min/avg/max/mdev = 0.028/0.242/8.717/1.162 ms
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Mitigation V8#

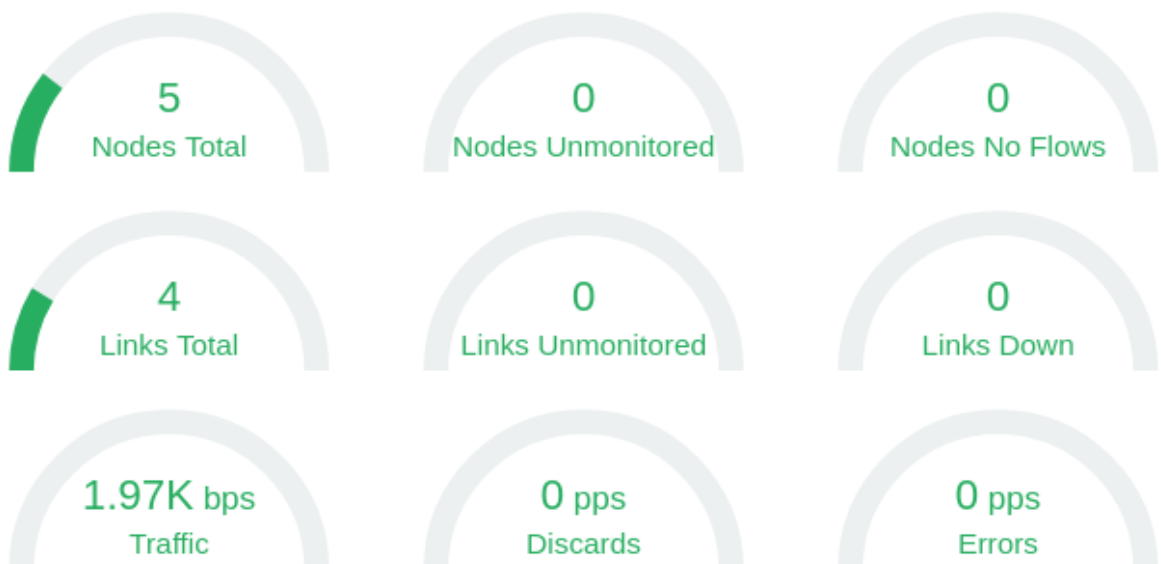
```

Gambar Lampiran 5.6 Packet Loss dan Latensi Saat Lalu Lintas ICMP Flood Setelah Mitigasi

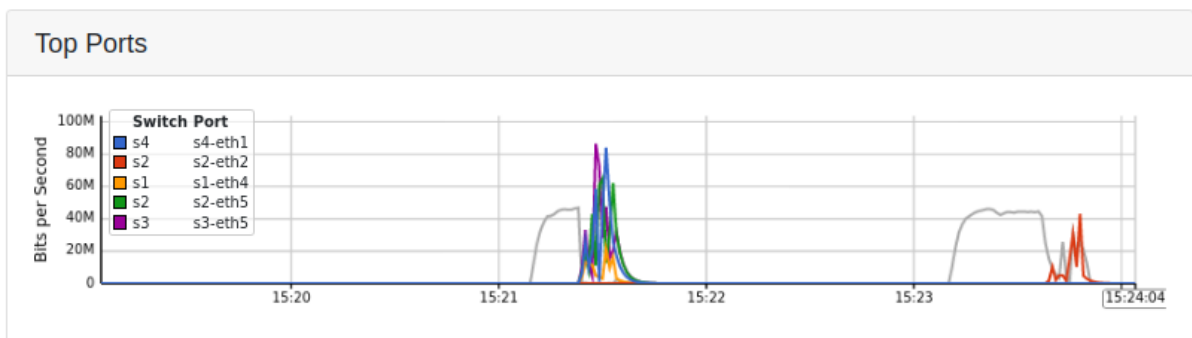
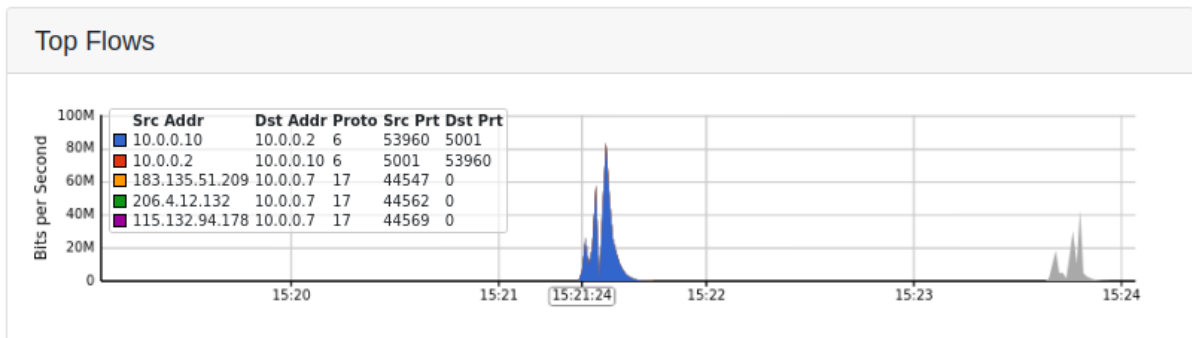
```
Terminal
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
-----
Traffic Analysis:
Legitimate Traffic = 0
DoS Traffic = 470
Total Traffic analyzed: 470
DDoS traffic detected.
Victim Host: h7
Mitigation process in progress!
-----
```

Gambar Lampiran 5.7 Hasil Deteksi Lalu Lintas ICMP Flood Setelah Mitigasi

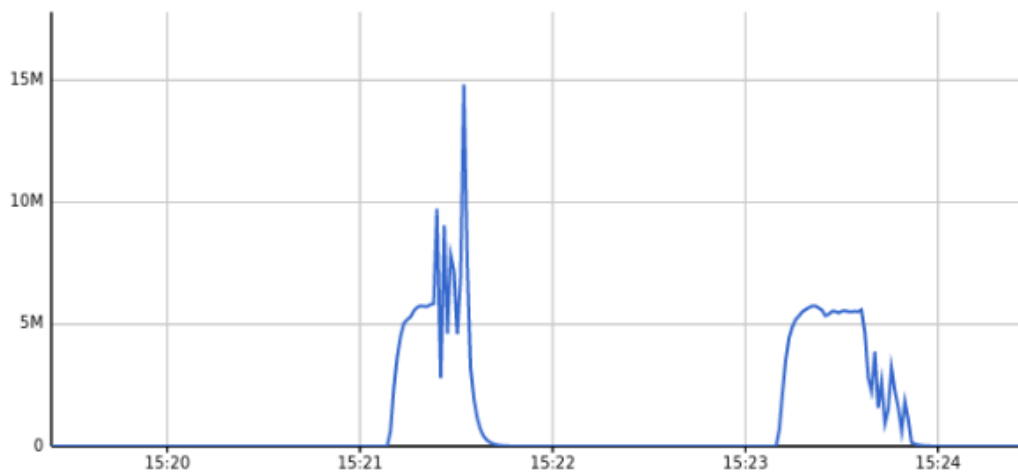
## 2. UDP Flood



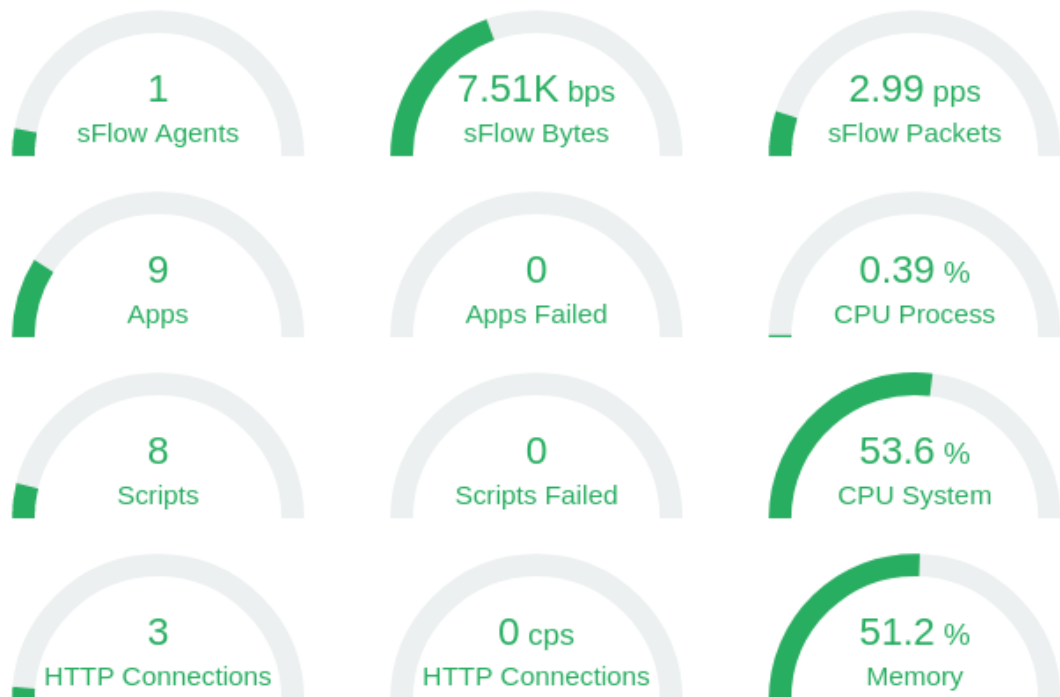
Gambar Lampiran 2.1 Dashboard Sflow RT Pada Lalu Lintas UDP Flood Setelah Mitigasi



Gambar Lampiran 2.2 Flow dan Port Saat Lalu lintas UDP Flood Setelah Mitigasi



Gambar Lampiran 2.3 Jumlah Flow saat Lalu Lintas UDP Flood Setelah Mitigasi



Gambar Lampiran 2.4 Resource Saat Lalu Lintas UDP Flood Setelah Mitigasi

```

"Node: h10"
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Mitigation V8# iperf -c 10.0.0.2
-----
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 kByte (default)
-----
[ 1] local 10.0.0.10 port 53960 connected with 10.0.0.2 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-10.0249 sec  3.61 GBytes  3.09 Gbits/sec
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Mitigation V8#

```

Gambar Lampiran 2.5 Throughput Lalu Lintas UDP Flood Setelah Mitigasi

```
"Node: h10"
64 bytes from 10.0.0.2: icmp_seq=82 ttl=64 time=0,065 ms
64 bytes from 10.0.0.2: icmp_seq=83 ttl=64 time=0,049 ms
64 bytes from 10.0.0.2: icmp_seq=84 ttl=64 time=0,051 ms
64 bytes from 10.0.0.2: icmp_seq=85 ttl=64 time=0,068 ms
64 bytes from 10.0.0.2: icmp_seq=86 ttl=64 time=0,052 ms
64 bytes from 10.0.0.2: icmp_seq=87 ttl=64 time=0,052 ms
64 bytes from 10.0.0.2: icmp_seq=88 ttl=64 time=0,064 ms
64 bytes from 10.0.0.2: icmp_seq=89 ttl=64 time=0,062 ms
64 bytes from 10.0.0.2: icmp_seq=90 ttl=64 time=0,066 ms
64 bytes from 10.0.0.2: icmp_seq=91 ttl=64 time=0,089 ms
64 bytes from 10.0.0.2: icmp_seq=92 ttl=64 time=0,072 ms
64 bytes from 10.0.0.2: icmp_seq=93 ttl=64 time=0,094 ms
64 bytes from 10.0.0.2: icmp_seq=94 ttl=64 time=0,086 ms
64 bytes from 10.0.0.2: icmp_seq=95 ttl=64 time=0,117 ms
64 bytes from 10.0.0.2: icmp_seq=96 ttl=64 time=0,078 ms
64 bytes from 10.0.0.2: icmp_seq=97 ttl=64 time=0,116 ms
64 bytes from 10.0.0.2: icmp_seq=98 ttl=64 time=0,084 ms
64 bytes from 10.0.0.2: icmp_seq=99 ttl=64 time=0,088 ms
64 bytes from 10.0.0.2: icmp_seq=100 ttl=64 time=20,4 ms

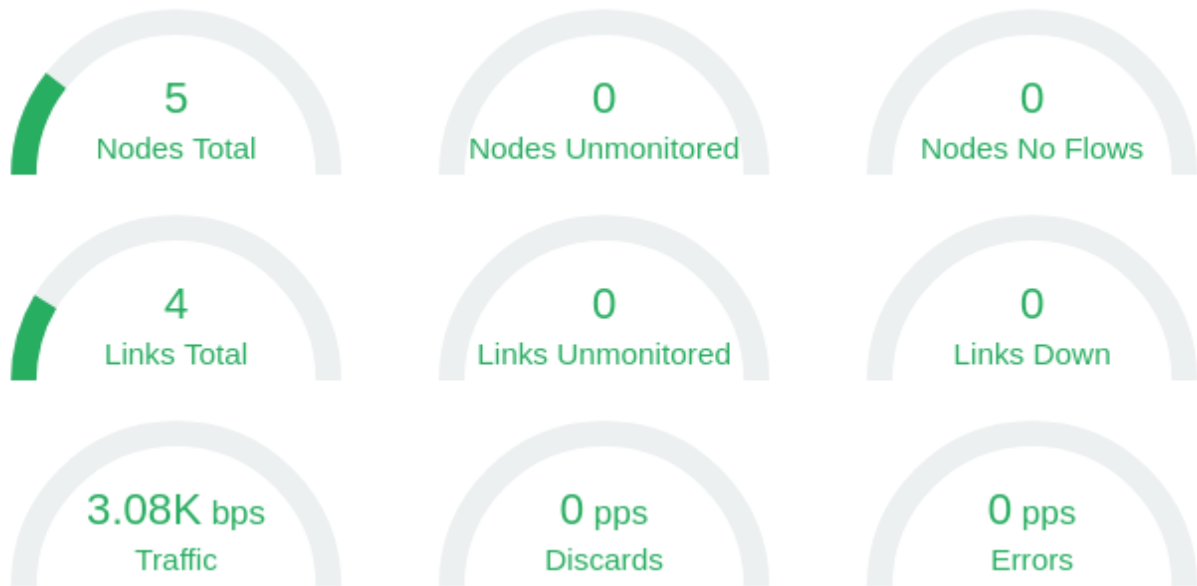
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 101327ms
rtt min/avg/max/mdev = 0,037/0,400/20,409/2,322 ms
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Mitigation V8#
```

Gambar Lampiran 2.6 Packet Loss dan Latensi Saat Lalu Lintas UDP Flood Setelah Mitigasi

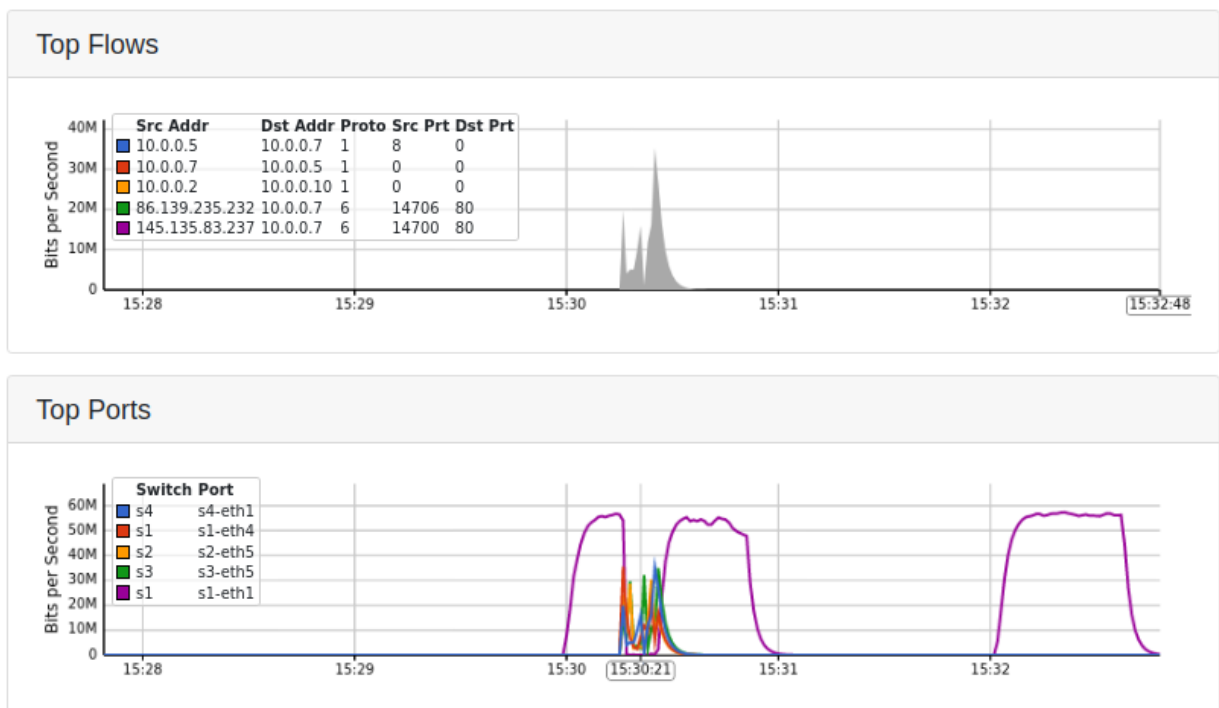
```
Terminal
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
-----
Traffic Analysis:
Legitimate Traffic = 0
DoS Traffic = 526
Total Traffic analyzed: 526
DDoS traffic detected.
Victim Host: h7
Mitigation process in progress!
-----
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
Blocking port 1 in switch 1
```

Gambar Lampiran 2.7 Hasil Deteksi Lalu Lintas UDP Flood Setelah Mitigasi

### 3. TCP SYN Flood

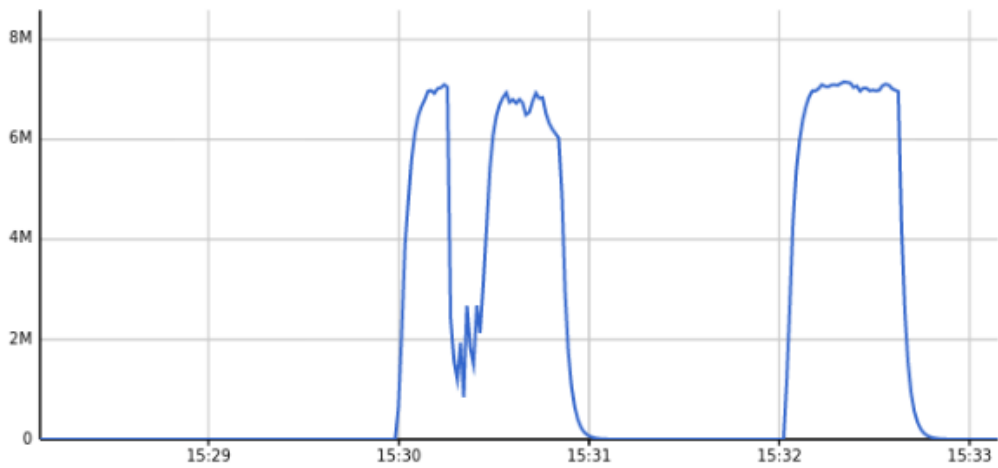


Gambar Lampiran 2.1 Dashboard Sflow RT Pada Lalu Lintas TCP SYN Setelah Mitigasi

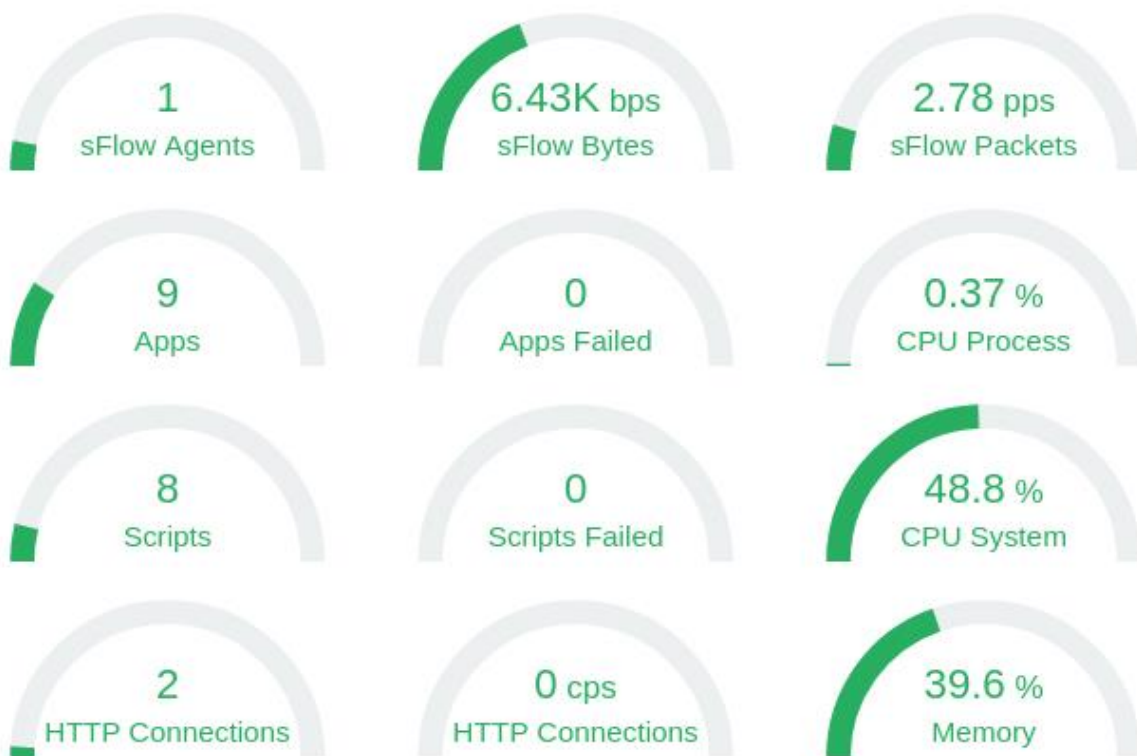


Gambar Lampiran 2.2 Flow dan Port Saat Lalu lintas TCP SYN Setelah Mitigasi





Gambar Lampiran 2.3 Jumlah Flow saat Lalu Lintas TCP SYN Setelah Mitigasi



Gambar Lampiran 2.4 Resource Saat Lalu Lintas TCP SYN Setelah Mitigasi

```
"Node: h10"
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Mitigation V8# iperf -c 10.0.0.2
-----
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 1] local 10.0.0.10 port 52104 connected with 10.0.0.2 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-10.0054 sec 4.06 GBytes 3.49 Gbits/sec
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Mitigation V8#
```

Gambar Lampiran 2.5 Throughput Lalu Lintas TCP SYN Setelah Mitigasi

```
"Node: h10"
64 bytes from 10.0.0.2: icmp_seq=82 ttl=64 time=0.099 ms
64 bytes from 10.0.0.2: icmp_seq=83 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=84 ttl=64 time=0.074 ms
64 bytes from 10.0.0.2: icmp_seq=85 ttl=64 time=0.055 ms
64 bytes from 10.0.0.2: icmp_seq=86 ttl=64 time=0.100 ms
64 bytes from 10.0.0.2: icmp_seq=87 ttl=64 time=0.074 ms
64 bytes from 10.0.0.2: icmp_seq=88 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=89 ttl=64 time=0.081 ms
64 bytes from 10.0.0.2: icmp_seq=90 ttl=64 time=0.074 ms
64 bytes from 10.0.0.2: icmp_seq=91 ttl=64 time=0.098 ms
64 bytes from 10.0.0.2: icmp_seq=92 ttl=64 time=0.068 ms
64 bytes from 10.0.0.2: icmp_seq=93 ttl=64 time=0.055 ms
64 bytes from 10.0.0.2: icmp_seq=94 ttl=64 time=0.054 ms
64 bytes from 10.0.0.2: icmp_seq=95 ttl=64 time=0.054 ms
64 bytes from 10.0.0.2: icmp_seq=96 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=97 ttl=64 time=0.084 ms
64 bytes from 10.0.0.2: icmp_seq=98 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_seq=99 ttl=64 time=11.0 ms
64 bytes from 10.0.0.2: icmp_seq=100 ttl=64 time=0.057 ms

--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 101298ms
rtt min/avg/max/mdev = 0.040/0.375/17.544/2.045 ms
root@ivan-virtual-machine:/home/ivan/Documents/DDoS Mitigation V8#
```

Gambar Lampiran 2.6 Packet Loss dan Latensi Saat Lalu Lintas TCP SYN Setelah Mitigasi



## LAMPIRAN B KODE PROGRAM

### 1. Pembuatan Topologi

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.link import TCLink
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.node import OVSKernelSwitch, RemoteController
# Impor fungsi wrapper dari sflow.py
from sflow import wrapper

class MyTopo(Topo):
    def build(self):
        switches = []
        hosts = []

        for s in range(1, 6):
            switch_name = 's{}'.format(s)
            switch = self.addSwitch(switch_name,
cls=OVSKernelSwitch, protocols='OpenFlow13')
            switches.append(switch)

        for h in range(1, 4):
            host_name = 'h{}'.format(s * 3 + h - 3)
            host_ip = '10.0.0.{}'.format(s * 3 + h - 3)
            host = self.addHost(host_name, cpu=1.0/20,
mac="00:00:00:00:00:{}".format(s * 3 + h - 3), ip=host_ip)
            hosts.append(host)
            self.addLink(host, switch)

        for i in range(len(switches) - 1):
            self.addLink(switches[i], switches[i + 1])

    def startNetwork():
        topo = MyTopo()
        c0 = RemoteController('c0', ip='127.0.0.1', port=6653)
        net = Mininet(topo=topo, link=TCLink, controller=c0)

        # Menggunakan fungsi wrapper dari sflow.py
        wrapper(net)

        net.start()
        CLI(net)
        net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    startNetwork()
```

## 2. Pembuatan dataset

### a. Traffic generator

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.link import TCLink
from mininet.log import setLogLevel
from mininet.node import OVSKernelSwitch, RemoteController
from time import sleep
from datetime import datetime
from random import randrange, choice

class CustomTopo(Topo):
    def build(self):
        switches = []
        hosts = []

        for s in range(1, 6):
            switch = self.addSwitch(f's{s}',
cls=OVSKernelSwitch, protocols='OpenFlow13')
            switches.append(switch)

            for h in range(1, 4):
                host_id = s * 3 + h - 3
                host_ip = f'10.0.0.{host_id}/24'
                host_mac = f"00:00:00:00:00:{host_id:02d}"
                host = self.addHost(f'h{host_id}',
cpu=1.0/20, mac=host_mac, ip=host_ip)
                hosts.append(host)
                self.addLink(host, switch)

            for i in range(len(switches) - 1):
                self.addLink(switches[i], switches[i + 1])

    def generate_ip():
        return f"10.0.0.{randrange(1, 16)}"

    def launch_network():
        topology = CustomTopo()
        controller = RemoteController('c0', ip='127.0.0.1')
```

```

network = Mininet(topo=topology, link=TCLink,
controller=controller)
network.start()

server_host = network.get('h1')
server_host.cmd('cd /home/mininet/webserver')
server_host.cmd('python -m SimpleHTTPServer 80 &')

hosts = [network.get(f'h{i}') for i in range(1, 16)]

for attack, cmd in [("ICMP Flood", "-1"), ("UDP Flood",
"-2"), ("TCP SYN Flood", "-S")]:
    attacker = choice(hosts)
    target_ip = generate_ip()
    print(f"-----")
    print(f"Performing {attack}")
    print(f"-----")

    attacker.cmd(f"timeout 20s hping3 {cmd} -V -d 120
-w 64 -p 80 --rand-source --flood {target_ip}")
    sleep(100)

network.stop()

if __name__ == '__main__':
    start_time = datetime.now()

    setLogLevel('info')
    launch_network()

    end_time = datetime.now()
    print(end_time - start_time)

```

## b. Ekstraksi dataset

```

Cimport topologi_kontroler
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER,
DEAD_DISPATCHER
from ryu.controller.handler import set ev cls

```

```

from ryu.lib import hub

from datetime import datetime

class
StatsCollectionApp(topologi_kontroler.SimpleSwitch13):
    def __init__(self, *args, **kwargs):
        super(StatsCollectionApp, self).__init__(*args,
**kwargs)
        self.datapaths = {}
        self.monitor_thread = hub.spawn(self._monitor)

        @set_ev_cls(ofp_event.EventOFPPStateChange,
[MAIN_DISPATCHER, DEAD_DISPATCHER])
        def _on_state_change(self, ev):
            datapath = ev.datapath
            if ev.state == MAIN_DISPATCHER:
                self._register_datapath(datapath)
            elif ev.state == DEAD_DISPATCHER:
                self._unregister_datapath(datapath)

        def _register_datapath(self, datapath):
            self.logger.debug('Registering datapath: %016x',
datapath.id)
            self.datapaths[datapath.id] = datapath

        def _unregister_datapath(self, datapath):
            self.logger.debug('Unregistering datapath: %016x',
datapath.id)
            if datapath.id in self.datapaths:
                del self.datapaths[datapath.id]

        def _monitor(self):
            while True:
                for dp in self.datapaths.values():
                    self._send_stats_request(dp)
                    hub.sleep(10)

        def _send_stats_request(self, datapath):
            self.logger.debug('Sending stats request: %016x',
datapath.id)
            parser = datapath.ofproto_parser
            req = parser.OFPFlowStatsRequest(datapath)
            datapath.send_msg(req)

        @set_ev_cls(ofp_event.EventOFPFlowStatsReply,
MAIN_DISPATCHER)
        def _handle_flow_stats_reply(self, ev):
            timestamp = datetime.now().timestamp()
            with open("FlowStatsfile.csv", "a+") as file:
                for stat in sorted(ev.msg.body, key=lambda f:
(f.match.get('eth_type'), f.match.get('ipv4_src'),
f.match.get('ipv4_dst'), f.match.get('ip_proto'))):
                    if stat.priority == 1:
                        ip_src = stat.match.get('ipv4_src',
'0.0.0.0')
                        ip_dst = stat.match.get('ipv4_dst',
'0.0.0.0')

```

```

0)         ip_proto = stat.match.get('ip_proto',
icmp_code, icmp_type, tp_src, tp_dst =
-1, -1, 0, 0
         if ip_proto == 1:
             icmp_code =
stat.match.get('icmpv4_code', -1)
             icmp_type =
stat.match.get('icmpv4_type', -1)
         elif ip_proto == 6:
             tp_src = stat.match.get('tcp_src',
0)
             tp_dst = stat.match.get('tcp_dst',
0)
         elif ip_proto == 17:
             tp_src = stat.match.get('udp_src',
0)
             tp_dst = stat.match.get('udp_dst',
0)
         flow_id = f"{ip_src}-{tp_src}-
{ip_dst}-{tp_dst}-{ip_proto}"
         pps = stat.packet_count /
stat.duration_sec if stat.duration_sec else 0
         pns = stat.packet_count /
stat.duration_nsec if stat.duration_nsec else 0
         bps = stat.byte_count /
stat.duration_sec if stat.duration_sec else 0
         bns = stat.byte_count /
stat.duration_nsec if stat.duration_nsec else 0

file.write(f"{timestamp},{ev.msg.datapath.id},{flow_id},{i
p_src},{tp_src},{ip_dst},{tp_dst},{ip_proto},{icmp_code},{
icmp_type},{stat.duration_sec},{stat.duration_nsec},{stat.
idle_timeout},{stat.hard_timeout},{stat.flags},{stat.packe
t_count},{stat.byte_count},{pps},{pns},{bps},{bns},1\n")

```

### 3. Pemilihan Hyperparameter Model

```

from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform

# Tentukan parameter yang ingin Anda cari
param_distributions = {
    'hidden_layer_sizes': [(50, 50), (100, 50), (64, 128), (256,
128)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': uniform(0.0001, 0.001), # Regularization term
    'learning_rate_init': uniform(0.001, 0.01) # Initial
learning rate
}

# Buat MLP Classifier
mlp = MLPClassifier(max_iter=100, early_stopping=True,
random_state=42)

# Buat instance RandomizedSearchCV

```



```

random_search = RandomizedSearchCV(mlp, param_distributions,
n_iter=100, cv=3, verbose=2, random_state=42, n_jobs=-1)

# Lakukan pencarian pada data pelatihan
random_search.fit(X_flow_train, y_flow_train)

# Hasil terbaik
print("Best parameters found: ", random_search.best_params_)
print("Best          cross-validation          score:
{: .2f}".format(random_search.best_score_))

```

#### 4. Pelatihan Model

```

import pandas as pd
import os
import joblib
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
import tensorflow as tf
from sklearn.neural_network import MLPClassifier
from keras.models import load_model
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

flow_dataset = pd.read_csv('dataset.csv')
flow_dataset.head()

flow_dataset.iloc[:, 2] = flow_dataset.iloc[:,
2].str.replace('.', '')
flow_dataset.iloc[:, 3] = flow_dataset.iloc[:,
3].str.replace('.', '')
flow_dataset.iloc[:, 5] = flow_dataset.iloc[:,
5].str.replace('.', '')

X_flow = flow_dataset.iloc[:, :-1].values
X_flow = X_flow.astype('float64')

y_flow = flow_dataset.iloc[:, -1].values

X_flow_train, X_flow_test, y_flow_train, y_flow_test =
train_test_split(X_flow, y_flow, test_size=0.25, random_state=0)

# Membuat dan melatih model MLP yang ditingkatkan
classifier = MLPClassifier(
    hidden_layer_sizes=(256, 128), # Ubah ukuran lapisan
    tersembunyi
    activation='relu',
    alpha=0.001, # Menambahkan regularisasi L2
    random_state=42,
    solver='adam',
    learning_rate_init=0.001,
    verbose=1,
    early_stopping=True, # Menggunakan early stopping
    validation_fraction=0.1 # Porsi data validasi
)

```

```

#Training Model
flow_model = classifier.fit(X_flow_train, y_flow_train)

y_flow_pred = flow_model.predict(X_flow_test)

# Save the scaler
joblib.dump(classifier, 'mlp_model.pkl')

```

## 5. Uji Coba

```

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

import switch
from datetime import datetime

import os
import joblib
from sklearn.neural_network import MLPClassifier

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score

class MonitoringApp(switch.SimpleSwitch13):

    def __init__(self, *args, **kwargs):
        super(MonitoringApp, self).__init__(*args, **kwargs)
        self.datapaths = {}
        self.monitor_thread = hub.spawn(self._monitor)

        start_time = datetime.now()
        self.train_flow_model()
        end_time = datetime.now()

        print("Training duration: ", (end_time - start_time))

        @set_ev_cls(ofp_event.EventOFPPStateChange, [MAIN_DISPATCHER,
DEAD_DISPATCHER])
        def handle_state_change(self, ev):
            datapath = ev.datapath
            if ev.state == MAIN_DISPATCHER:
                if datapath.id not in self.datapaths:
                    self.logger.debug('Registering datapath: %016x',
datapath.id)
                    self.datapaths[datapath.id] = datapath
            elif ev.state == DEAD_DISPATCHER:
                if datapath.id in self.datapaths:
                    self.logger.debug('Unregistering datapath:
%016x', datapath.id)
                    del self.datapaths[datapath.id]

    def _monitor(self):
        while True:
            for dp in self.datapaths.values():

```

```

        self.request_stats(dp)
        hub.sleep(10)
        self.predict_traffic()

    def request_stats(self, datapath):
        self.logger.debug('Sending stats request: %016x',
datapath.id)
        parser = datapath.ofproto_parser
        req = parser.OFPFlowStatsRequest(datapath)
        datapath.send_msg(req)

    @set_ev_cls(ofp_event.EventOFPFlowStatsReply,
MAIN_DISPATCHER)
    def handle_flow_stats_reply(self, ev):
        timestamp = datetime.now().timestamp()

        with open("PredictFlowStatsfile.csv", "w") as file:

file.write('timestamp,datapath_id,flow_id,ip_src,tp_src,ip_dst,t
p_dst,ip_proto,icmp_code,icmp_type,flow_duration_sec,flow_durati
on_nsec,idle_timeout,hard_timeout,flags,packet_count,byte_count,
packet_count_per_second,packet_count_per_nsecond,byte_count_per_
second,byte_count_per_nsecond\n')

        body = ev.msg.body
        for stat in sorted([flow for flow in body if
flow.priority == 1], key=lambda flow: (flow.match['eth_type'],
flow.match['ipv4_src'], flow.match['ipv4_dst'],
flow.match['ip_proto'])):
            ip_src = stat.match['ipv4_src']
            ip_dst = stat.match['ipv4_dst']
            ip_proto = stat.match['ip_proto']
            icmp_code = stat.match.get('icmpv4_code', -1)
            icmp_type = stat.match.get('icmpv4_type', -1)
            tp_src = stat.match.get('tcp_src', 0) if ip_proto
== 6 else stat.match.get('udp_src', 0)
            tp_dst = stat.match.get('tcp_dst', 0) if ip_proto
== 6 else stat.match.get('udp_dst', 0)

            flow_id =
f"{ip_src}{tp_src}{ip_dst}{tp_dst}{ip_proto}"
            packet_count_per_second = stat.packet_count /
stat.duration_sec if stat.duration_sec else 0
            packet_count_per_nsecond = stat.packet_count /
stat.duration_nsec if stat.duration_nsec else 0
            byte_count_per_second = stat.byte_count /
stat.duration_sec if stat.duration_sec else 0
            byte_count_per_nsecond = stat.byte_count /
stat.duration_nsec if stat.duration_nsec else 0

file.write(f"{timestamp},{ev.msg.datapath.id},{flow_id},{ip_src}
,{tp_src},{ip_dst},{tp_dst},{ip_proto},{icmp_code},{icmp_type},{
stat.duration_sec},{stat.duration_nsec},{stat.idle_timeout},{sta
t.hard_timeout},{stat.flags},{stat.packet_count},{stat.byte_coun
t},{packet_count_per_second},{packet_count_per_nsecond},{byte_co
unt_per_second},{byte_count_per_nsecond}\n")

    def train_flow_model(self):

```

```

self.logger.info("Starting flow model training...")

if os.path.isfile('mlp_model.pkl'):
    self.flow_model = joblib.load('mlp_model.pkl')
else:
    dataset = pd.read_csv('dataset.csv')
    dataset.iloc[:, 2] = dataset.iloc[:,
2].str.replace('.', '')
    dataset.iloc[:, 3] = dataset.iloc[:,
3].str.replace('.', '')
    dataset.iloc[:, 5] = dataset.iloc[:,
5].str.replace('.', '')

    X = dataset.iloc[:, :-1].values.astype('float64')
    y = dataset.iloc[:, -1].values

    X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.25, random_state=0)

    classifier = MLPClassifier(
        hidden_layer_sizes=(256, 128),
        activation='relu',
        alpha=0.001,
        random_state=42,
        solver='adam',
        learning_rate_init=0.001,
        verbose=1,
        early_stopping=True,
        validation_fraction=0.1
    )

    self.flow_model = classifier.fit(X_train, y_train)
    joblib.dump(self.flow_model, 'mlp_model.pkl')

    y_pred = self.flow_model.predict(X_test)

    self.logger.info("Confusion Matrix:")
    self.logger.info(confusion_matrix(y_test, y_pred))

    accuracy = accuracy_score(y_test, y_pred)
    self.logger.info(f"Success accuracy: {accuracy *
100:.2f}%")
    self.logger.info(f"Fail accuracy: {(1 - accuracy) *
100:.2f}%")

    def predict_traffic(self):
        try:
            predict_flow_dataset =
pd.read_csv('PredictFlowStatsfile.csv')

            predict_flow_dataset.iloc[:, 2] =
predict_flow_dataset.iloc[:, 2].str.replace('.', '')
            predict_flow_dataset.iloc[:, 3] =
predict_flow_dataset.iloc[:, 3].str.replace('.', '')
            predict_flow_dataset.iloc[:, 5] =
predict_flow_dataset.iloc[:, 5].str.replace('.', '')

            X_predict_flow =
predict_flow_dataset.values.astype('float64')

```

```

        y_pred = self.flow_model.predict(X_predict_flow)

        legitimate_traffic = sum(y_pred == 0)
        ddos_traffic = sum(y_pred != 0)

        self.logger.info("Traffic Analysis:")
        self.logger.info(f"Legitimate Traffic:
{legitimate_traffic}")
        self.logger.info(f"DDoS Traffic: {ddos_traffic}")
        self.logger.info(f"Total Traffic analyzed:
{len(y_pred)}")

        if (legitimate_traffic / len(y_pred)) * 100 > 80:
            self.logger.info("Traffic is mostly normal.")
        else:
            victim = int(predict_flow_dataset.iloc[y_pred !=
0, 5].iloc[0]) % 20
            self.logger.info("DDoS traffic detected.")
            self.logger.info(f"Victim Host: h{victim}")
            print("Mitigation process in progress!")
            self.mitigation = 1

        with open("PredictFlowStatsfile.csv", "w") as file:
            file.write('timestamp,datapath_id,flow_id,ip_src,tp_src,ip_dst,t
p_dst,ip_proto,icmp_code,icmp_type,flow_duration_sec,flow_durati
on_nsec,idle_timeout,hard_timeout,flags,packet_count,byte_count,
packet_count_per_second,packet_count_per_nsecond,byte_count_per_
second,byte_count_per_nsecond\n')

    except Exception as e:
        self.logger.error(f"Error in predicting traffic:
{e}")

```