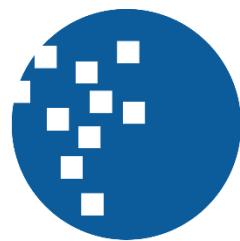


**RANCANG BANGUN SISTEM PROXY MITIGASI DDOS
BERBASIS MACHINE LEARNING DAN BLOCKCHAIN**



UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA

TUGAS AKHIR

Matthew Brandon Dani

00000036391

**PROGRAM STUDI TEKNIK KOMPUTER
FAKULTAS TEKNIK DAN INFORMATIKA
UNIVERSITAS MULTIMEDIA NUSANTARA
TANGERANG
2023**

RANCANG BANGUN SISTEM PROXY MITIGASI DDOS BERBASIS MACHINE LEARNING DAN BLOCKCHAIN



Diajukan sebagai Salah Satu Syarat untuk Memperoleh
Gelar Sarjana Teknik Komputer

Matthew Brandon Dani

00000036391

PROGRAM STUDI TEKNIK KOMPUTER

FAKULTAS TEKNIK DAN INFORMATIKA

UNIVERSITAS MULTIMEDIA NUSANTARA

TANGERANG

2023

HALAMAN PERNYATAAN TIDAK PLAGIAT

Dengan ini saya,

Nama : Matthew Brandon Dani
Nomor Induk Mahasiswa : 00000036391
Program studi : Teknik Komputer

Tugas akhir dengan judul:

RANCANG BANGUN SISTEM PROXY MITIGASI DDOS BERBASIS MACHINE LEARNING DAN BLOCKCHAIN

merupakan hasil karya saya sendiri bukan plagiat dari karya ilmiah yang ditulis oleh orang lain, dan semua sumber, baik yang dikutip maupun dirujuk, telah saya nyatakan dengan benar serta dicantumkan di Daftar Pustaka.

Jika di kemudian hari terbukti ditemukan kecurangan/penyimpangan, baik dalam pelaksanaan skripsi maupun dalam penulisan laporan skripsi, saya bersedia menerima konsekuensi dinyatakan TIDAK LULUS untuk Tugas Akhir yang telah saya tempuh.

Tangerang, 16 Juni 2023



Brandon

(Matthew Brandon Dani)

UNIVERSITAS
MULTIMEDIA
NUSANTARA

HALAMAN PERSETUJUAN

Tugas akhir dengan judul

RANCANG BANGUN SISTEM PROXY MITIGASI DDOS BERBASIS MACHINE LEARNING DAN BLOCKCHAIN

Oleh

Nama : Matthew Brandon Dani
NIM : 00000036391
Program Studi : Teknik Komputer
Fakultas : Fakultas Teknik dan Informatika

Telah disetujui untuk diajukan pada

Sidang Ujian Tugas Akhir Universitas Multimedia Nusantara

Tangerang, 16 Juni 2023

Pembimbing


Samuel Hutagalung, M.T.I.
304038902

Pembimbing


Dareen Kusumah Halim, S.Kom., M.Eng.Sc.
317129202

Ketua Teknik Komputer


Samuel Hutagalung, M.T.I.

HALAMAN PENGESAHAN

Tugas akhir dengan judul

RANCANG BANGUN SISTEM PROXY MITIGASI DDOS BERBASIS MACHINE LEARNING DAN BLOCKCHAIN

Oleh

Nama : Matthew Brandon Dani
NIM : 00000036391
Program Studi : Teknik Komputer
Fakultas : Fakultas Teknik dan Informatika

Telah diujikan pada hari Selasa, 27 Juni 2023

Pukul 15.00 s.d 17.00 dan dinyatakan

LULUS

Dengan susunan penguji sebagai berikut.

Ketua Sidang

Nabila Husna Shabrina, S.T., M. T.
321099301

Penguji

Monica Pratiwi, S.ST., M.T.
325059601

Pembimbing

Samuel Hutagalung, M.T.I.
304038902

Pembimbing

Dareen Kusuma Halim, S.Kom., M.Eng.Sc
317129202

Ketua Teknik Komputer

Samuel Hutagalung, M.T.I.

HALAMAN PERSETUJUAN PUBLIKASI KARYA ILMIAH UNTUK KEPENTINGAN AKADEMIS

Sebagai civitas academica Universitas Multimedia Nusantara, saya yang bertanda tangan di bawah ini:

Nama : Matthew Brandon Dani
NIM : 00000036391
Program Studi : Teknik Komputer
Fakultas : Fakultas Teknik dan Informatika
Jenis Karya : ***Tesis/Skripsi/Tugas Akhir** (*coret salah satu)

Demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Multimedia Nusantara Hak Bebas Royalti Nonekslusif (*Non-exclusive Royalty-Free Right*) atas karya ilmiah saya yang berjudul.

RANCANG BANGUN SISTEM PROXY MITIGASI DDOS BERBASIS MACHINE LEARNING DAN BLOCKCHAIN

Beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Nonekslusif ini, Universitas Multimedia Nusantara berhak menyimpan, mengalihmediakan/mengalihformatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta. Demikian pernyataan ini saya buat dengan sebenarnya.

Tangerang, 16 Juni 2023

Yang menyatakan,

Brandon

(Matthew Brandon Dani)

KATA PENGANTAR

Puji dan syukur kepada Tuhan Yang Maha Esa atas berkat yang diberikan kepada penulis selama proses pengerjaan dan pembuatan tugas akhir sehingga penulis dapat menyelesaikan laporan tugas akhir yang berjudul “RANCANG BANGUN SISTEM PROXY MITIGASI DDOS BERBASIS MACHINE LEARNING DAN BLOCKCHAIN”. Dengan selesainya laporan tugas akhir ini, penulis mengucapkan syukur atas kelulusan penulis atas pendidikan strata satu program studi Teknik Komputer pada Fakultas Teknik dan Informatika Universitas Multimedia Nusantara.

Tugas Akhir ini tidak dapat terselesaikan tanpa adanya dukungan dari berbagai pihak yang telah mendukung dan membimbing penulis. Oleh karena itu penulis ingin mengucapkan terima kasih kepada :

1. Dr. Ninok Leksono, selaku Rektor Universitas Multimedia Nusantara.
2. Dr. Eng. Niki Prastomo, selaku Dekan Fakultas Universitas Multimedia Nusantara.
3. Samuel Hutagalung, M.T.I., selaku Ketua Program Studi Universitas Multimedia Nusantara dan juga sebagai Pembimbing pertama yang telah memberikan bimbingan, arahan, dan motivasi atas terselesainya tugas akhir ini.
4. Dareen Kusuma Halim, S.Kom., M.Eng.Sc., sebagai Pembimbing kedua yang telah memberikan bimbingan, arahan, dan motivasi atas terselesainya tugas akhir ini.
5. Keluarga saya yang telah memberikan bantuan dukungan material dan moral, sehingga penulis dapat menyelesaikan tugas akhir ini.
6. Teman-teman Progam Studi Teknik Komputer Universitas Multimedia Nusantara angkatan 2019, yang telah bersama-sama belajar dan berjuang untuk menyelesaikan pendidikan strata satu program studi teknik komputer.

7. Dan pihak-pihak lainnya yang tidak dapat disebutkan satu persatu yang telah membantu dalam pembuatan tugas akhir

Akhir kata, semoga karya ilmiah ini dapat bermanfaat dan menginspirasi bagi para pembaca sehingga dapat menjadi acuan dan dilanjutkan ke dalam penelitian terkait selanjutnya.

Tangerang, 16 Juni 2023

Brandon

(Matthew Brandon Dani)



RANCANG BANGUN SISTEM PROXY MITIGASI DDOS

BERBASIS MACHINE LEARNING DAN BLOCKCHAIN

(Matthew Brandon Dani)

ABSTRAK

DDoS merupakan serangan pada jaringan komputer yang umum ditemukan. Namun sampai sekarang, masih belum ada solusi *intrusion prevention system* (IPS) yang *open-source* dan dapat memitigasinya secara efektif, keseluruhan, dan akurat. Hal ini juga dukung dengan terus meningkatnya angka serangan DDoS dari berbagai survei sumber referensi. Ketidakefektifannya karena DDoS memiliki metode dan teknik yang berbeda beda sehingga akan menghasilkan serangan berupa pola stokastik. Pola stokastik akan sulit untuk dideteksi menggunakan metode "*threshold*" pada solusi IPS konvensional. Meningkatnya tren arsitektur sistem terdistribusi menggunakan jaringan internet juga akan memperbesar celah sistem untuk terserang jika tidak dirancang secara matang. Oleh karena itu dirancang dan dibangunlah sistem proxy terdistribusi yang dapat mendeteksi dan mengidentifikasi serangan DDoS menggunakan *machine learning* dan *blockchain*. Dilakukan penelitian menggunakan 2 metode *machine learning* yaitu pendekatan secara individual dan *timeseries*. Penelitian memakai beberapa algoritma *machine learning* seperti *Bidirectional LSTM*, SVM, Linear Regresion, dan lain-lain. Akan dibuat setiap model *machine learning* untuk setiap protokol jaringan komputer dan dianalisis menggunakan beberapa matriks pengujian. Untuk mendistribusikan informasi penyerang, digunakan *database* terdistribusi berbasis blockchain BigchainDB dan sistem konsensus. Penelitian menghasilkan model *machine learning* mendapatkan akurasi rata rata diatas 95% dengan waktu deteksi kurang dari 1 detik dengan menggunakan *feature* dan data *pre-processing* yang dipilih. Walaupun terjadi *overfitting* akibat penggunaan dataset yang kecil, sehingga hanya dapat menjadi referensi penggunaan *machine learning* saja. Waktu distribusi data membutuhkan waktu 0,67 detik untuk tersebar ke setiap proxy terdistribusi. Sistem yang dirancang berhasil menunjukkan tingkat efektivitas yang lebih tinggi daripada IPS / IDS Snort yang *open-source* dan banyak dipakai sekarang, terutama pada tingkat deteksi *false positive*. Salah satunya adalah sistem ini dapat memitigasi serangan DDoS yang menggunakan IP *Spoofing* dengan tetap meneruskan paket yang normal.

Kata kunci: DDoS, Proxy, Sistem Terdistribusi, *Machine Learning*, *Blockchain*

RANCANG BANGUN SISTEM PROXY MITIGASI DDOS

BERBASIS MACHINE LEARNING DAN BLOCKCHAIN

(Matthew Brandon Dani)

ABSTRACT (English)

DDoS is attack on computer network that is commonly found. But until now, there is still no intrusion prevention system (IPS) solutions that is open-source and can mitigate it effectively, comprehensively, and accurately. This is also supported by the continued increase in the number of DDoS attacks from various surveys of reference sources. The ineffectiveness is because DDoS has different methods and techniques that will result in the form of a stochastic pattern. Stochastic patterns will be difficult to detect using the "threshold" method on conventional IPS solutions. The increasing trend of distributed system architectures using internet networks will also increase the possibility of the system to be attacked if not designed carefully. Therefore a distributed proxy system was designed and built that can detect and identify DDoS attacks using machine learning and blockchain. Research was carried out using 2 machine learning methods the individual approach and the time series approach. Research uses several machine learning algorithms such as Bidirectional LSTM, SVM, Linear Regression, and others. Each machine learning model will be created for each computer network protocol and analyzed using several test matrices. To distribute attacker information, a distributed database based on blockchain BigchainDB and a consensus algorithm is used. Research shows that machine learning models obtaining an average accuracy of above 95% with a detection time of less than 1 second using selected features and pre-processing data. Even though there is overfitting due to the use of a small dataset, it can only be used as a reference for using machine learning. Data distribution time takes 0.67 seconds to spread to each distributed proxy. The successfully designed system shows a higher level of effectiveness than the open-source and widely used IPS / IDS Snort, especially at the false positive detection rate. One of them is that this system can mitigate DDoS attacks using IP Spoofing while still forwarding normal packets.

Keywords: *DDoS, Proxy, Distributed System, Machine Learning, Blockchain*

DAFTAR ISI

HALAMAN PERNYATAAN TIDAK PLAGIAT.....	iii
HALAMAN PENGESAHAN	iv
HALAMAN PERSETUJUAN PUBLIKASI KARYA ILMIAH UNTUK KEPENTINGAN AKADEMIS	v
KATA PENGANTAR.....	vi
ABSTRAK	viii
ABSTRACT (<i>English</i>).....	ix
DAFTAR ISI.....	x
DAFTAR TABEL	xiii
DAFTAR GAMBAR.....	xiv
DAFTAR LAMPIRAN	xix
BAB I PENDAHULUAN.....	1
1.1 Latar Belakang	1
1.2 Identifikasi Masalah.....	7
1.3 Batasan Penelitian	7
1.4 Tujuan Penelitian	8
1.5 Manfaat Penelitian	8
1.6 Sistematika Penulisan	9
BAB II TINJAUAN PUSTAKA.....	11
2.1 Penelitian Terdahulu.....	11
2.2 Tinjauan Teori.....	35
2.2.1 OSI Layer.....	35
2.2.1.1 TCP	36
2.2.1.2 UDP	38
2.2.1.3 ICMP	39
2.2.1.4 HTTP	39
2.2.2 IP Tables	40
2.2.3 <i>Intrusion Prevention System</i> dan <i>Intrusion Detection System</i>	40
2.2.3.1 Snort	42
2.2.3.4 NetfilterQueue	43

2.2.5	Reverse Proxy	43
2.2.5.1	Concurrent Connection	44
2.2.6	Python Socket	44
2.2.7	Python Multithreading	45
2.2.8	BigchainDB	46
2.2.9	Democracy JS	47
2.2.10	Standar Deviasi dan Variansi	48
2.2.11	Machine Learning	48
2.2.12	Bidirectional LSTM	50
2.2.13	Imbalance dataset.....	51
2.2.13.1	Smote	52
2.2.14	ReactJS.....	52
2.2.15	Express JS	53
2.2.16	Hping3	53
2.2.17	Jmeter.....	54
2.2.18	IP Spoofing.....	54
2.3	Simpulan.....	54
BAB III ANALISIS DAN PERANCANGAN SISTEM.....		57
3.1	Metode Penelitian	57
3.2	Studi literatur	57
3.3	Arsitektur General Sistem Proxy Mitigasi DDoS berbasis Machine Learning dan Blockchain.....	58
3.4	Alur Sistem Proxy Mitigasi DDoS berbasis Machine Learning dan Blockchain.....	59
3.5	Analisis Features yang akan Dipakai	62
3.6	Perancangan Sistem Proxy	72
3.7	Pengambilan Dataset.....	75
3.8	Perancangan Machine Learning	77
3.9	Perancangan Sistem Firewall	85
3.10	Perancangan Sistem Distribusi Data	87
3.10.1	Perancangan database terdistribusi BigchainDB	87
3.10.2	Perancangan Sistem Backend.....	88

3.10.3	Perancangan Sistem Frontend.....	90
3.10.4	Analisa Kecepatan Persebaran Data	91
3.11	Perancangan Deployment Infratruktur Sistem	93
3.12	Perbandingan dengan IDS / IPS Snort.....	93
BAB IV IMPLEMENTASI DAN PENGUJIAN SISTEM		95
4.1	Spesifikasi Sistem dan Testbed	95
4.2	Implementasi Sistem	97
4.2.1	Pembangunan Sistem Proxy	98
4.2.2	Pembangunan Machine Learning	103
4.2.3	Pembangunan Sistem Firewall	109
4.2.4	Pembangunan Sistem Distribusi Data	111
4.2.5	Pembangunan IPS / IDS Snort	121
4.3	Hasil dan analisis Pengujian Sistem	124
4.3.1	Analisis Hasil Evaluasi Performa Machine Learning	124
4.3.2	Analisis Hasil Evaluasi Sistem Reverse Proxy	140
4.3.3	Analisis Hasil Evaluasi Sistem Distribusi Data.....	143
4.3.4	Analisis Hasil Perbandingan dengan IDS / IPS Snort.....	146
4.4	Kendala dan Solusi penelitian	148
BAB V SIMPULAN DAN SARAN		149
5.1	Simpulan.....	149
5.2	Saran.....	151
DAFTAR PUSTAKA		153
LAMPIRAN.....		157

**UNIVERSITAS
MULTIMEDIA
NUSANTARA**

DAFTAR TABEL

Tabel 2.1 Features yang Digunakan dalam Deteksi Serangan DDoS menggunakan SVM.....	16
Tabel 2.2 Tabel Snort Rule Syntax.....	42
Tabel 2.3 Operasi Data BigchainDB	46
Tabel 3.1 Ilustrasi Normalisasi Dataset	79
Tabel 3.2 List API Endpoint Backend	89
Tabel 4.1 Hasil Dataset yang Terkumpul	104
Tabel 4.2 Perbandingan Dataset Sebelum dan Sesudah Balancing	105
Tabel 4.3 Perbandingan Dataset Normalisasi	108
Tabel 4.4 Perbandingan Hasil Performa Machine Learning dengan Classification Report.....	128
Tabel 4.5 Perbandingan Hasil Performa Machine Learning dengan Time Prediction	134
Tabel 4.6 Perbandingan Hasil Performa Machine Learning Identifikasi dengan Classification Matrix	135
Tabel 4.7 Perbandingan Hasil Performa Machine Learning Identifikasi dengan Waktu Prediksi	137
Tabel 4.8 Perbandingan Hasil Performa Machine Learning dengan Metode yang Berbeda	139
Tabel 4.9 Perbandingan Rata Rata Penggunaan CPU antara Sistem yang Dibuat dengan Aplikasi Snort.....	147
Tabel 4.10 Perbandingan Waktu TTM Sistem yang Dibuat dengan Aplikasi Snort	147

UNIVERSITAS
MULTIMEDIA
NUSANTARA

DAFTAR GAMBAR

Gambar 1.1 Gambaran Umum Serangan DDoS.....	2
Gambar 1.2 Grafik Trend Global Serangan DDoS	3
Gambar 1.3 Ilustrasi Perbedaan Sistem Distributed, Centralized, dan Decentralized	5
Gambar 1.4 Ilustrasi Proxy Terdistribusi	6
Gambar 2.1 Ilustrasi serangan TCP SYN	12
Gambar 2.2 Proses Deteksi Serangan DDoS secara individual	14
Gambar 2.3 Ilustrasi Klasifikasi Algoritma Machine Learning SVM.....	16
Gambar 2.4 Proses Deteksi menggunakan LSTM.....	19
Gambar 2.5 Proses Transformasi Dataset LSTM.....	20
Gambar 2.6 Korelasi antara time window dengan akurasi model machine learning LSTM	21
Gambar 2.7 Arsitektur Pembagian Informasi Penyerang menggunakan Blockchain dan IPFS	23
Gambar 2.8 Sequence Diagram Pembagian Informasi Identitas Penyerang menggunakan Blockchain dan IPFS	24
Gambar 2.9 Perbandingan input 1000 data dengan BigchainDB dan Amazon QLDB	26
Gambar 2.10 Perbandingan membaca 1000 data dengan BigchainDB dan Amazon QLDB	26
Gambar 2.11 Karakteristik Serangan DDOS.....	30
Gambar 2.12 Hasil Pengujian dan Analisis penggunaan features MEFF	30
Gambar 2.13 Hasil Pengujian dan Analisis Penggunaan Multicore dan Multithread pada Proxy Server	32
Gambar 2.14 Ilustrasi Ketergantungan OSI Layer	35
Gambar 2.15 Alur handshaking TCP	36
Gambar 2.16 TCP Segmen	37
Gambar 2.17 UDP Segmen.....	38

Gambar 2.18 ICMP Segmen	39
Gambar 2.19 Implementasi IDS dan IPS.....	41
Gambar 2.20 Implementasi Reverse Proxy	43
Gambar 2.21 Contoh Implementasi Socket untuk Reverse Proxy	45
Gambar 2.22 Contoh Implementasi Multithreading Socket untuk Reverse Proxy.....	45
Gambar 2.23 Penggunaan DemocracyJS.....	47
Gambar 2.24 Struktur LSTM.....	50
Gambar 2.25 Struktur Bidirectional LSTM.....	51
Gambar 2.26 Ilustrasi Teknik SMOTE untuk menangani masalah dataset tidak seimbang.....	52
Gambar 2.27 Diagram State of The Art	56
Gambar 3.1 Diagram tahapan penelitian	57
Gambar 3.2 Diagram Arsitektur Sistem secara General	58
Gambar 3.3 Sequence Diagram untuk Deteksi Paket.....	59
Gambar 3.4 Sequence Diagram untuk Persebaran Data Informasi Penyerang	60
Gambar 3.5 Sequence Diagram untuk Admin Membuat Akun dan Melakukan Operasi Database.....	61
Gambar 3.6 Alur Kerja Sistem Proxy Mitigasi DDoS berbasis Machine Learning dan DDoS	62
Gambar 3.7 Alur Analisis Features yang akan Dipakai.....	63
Gambar 3.8 Potongan Kode Program Random Request UDP	64
Gambar 3.9 Potongan Kode Program Video Streaming UDP.....	65
Gambar 3.10 Potongan Kode Bash Script ICMP Normal	65
Gambar 3.11 Potongan Kode Program HTTP GET Flood	66
Gambar 3.12 Potongan Kode Program HTTP GET Flood 2	66
Gambar 3.13 Potongan Kode Program HTTP GET Flood 3	66
Gambar 3.14 Potongan Kode Firewall Sniffer	67
Gambar 3.15 Contoh Observasi Menggunakan Software Wireshark	69

Gambar 3.16 Kode SVM features Importance	69
Gambar 3.17 Hasil Plot SVM features Importance.....	70
Gambar 3.18 Alur Kerja Analisis features importance SVM.....	70
Gambar 3.19 Arsitektur Sistem Reverse Proxy	72
Gambar 3.20 Arsitektur Multithreading Socket.....	73
Gambar 3.21 Sequence diagram untuk proses pengambilan dataset individual	74
Gambar 3.22 Perbedaan Metode Individual dan Timeseries	74
Gambar 3.23 Arsitektur Pengambilan Dataset.....	76
Gambar 3.24 Cara Kerja Dataset Individu	76
Gambar 3.25 Cara Kerja Dataset Timeseries	77
Gambar 3.26 Potongan Kode SMOTE untuk Balancing Dataset	78
Gambar 3.27 Ilustrasi Proses Balancing SMOTE	78
Gambar 3.28 Arsitektur Machine Learning DNN	81
Gambar 3.29 Arsitektur Machine Learning LSTM	82
Gambar 3.30 Perbedaan Atomic dan Composite.....	83
Gambar 3.31 Ilustrasi Metode Karakteristik Paket Terbanyak	84
Gambar 3.32 Arsitektur Sistem Firewall dan Proxy	86
Gambar 3.33 Diagram 2 Fase Machine Learning.....	86
Gambar 3.34 Database Schema	88
Gambar 3.35 Rancangan Frontend	91
Gambar 3.36 Alur Evaluasi Sistem Distribusi Data	92
Gambar 3.37 Ilustrasi Deployment Sistem Proxy Mitigasi DDoS	93
Gambar 4.1 VMware Synchronize time.....	95
Gambar 4.2 Potongan Kode Webserver	96
Gambar 4.3 Arsitektur Reverse Proxy.....	97
Gambar 4.4 Arsitektur Modul Proxy	98
Gambar 4.5 Potongan Pembuatan Thread TCP	99
Gambar 4.6 Potongan Kode dan Contoh File Konfigurasi Proxy Server	99
Gambar 4.7 Potongan Kode Thread Safe Logging.....	100

Gambar 4.8 Potongan Kode Data Parser	100
Gambar 4.9 Tampilan Proxy Server	101
Gambar 4.10 Arsitektur Firewall Server.....	101
Gambar 4.11 Potongan Kode fungsi Filter Firewall.....	102
Gambar 4.12 Alur Kerja Pembangunan Model Machine Learning	102
Gambar 4.13 Potongan Kode Perubah String menjadi Numerik Bytes	103
Gambar 4.14 Ilustrasi Proses Transformasi Dataset LSTM	104
Gambar 4.15 Potongan Kode Transformasi Dataset menjadi 3 Dimensional Matriks	104
Gambar 4.16 Potongan Kode Teknik SMOTE dan Perbedaan Jumlah Dataset.....	105
Gambar 4.17 Potongan Kode Proses Normalisasi Dataset	106
Gambar 4.18 Potongan Kode Eksport Instance.....	106
Gambar 4.19 Potongan Kode Pembagian Dataset Training dan Testing....	106
Gambar 4.20 Potongan Kode Pembuatan Model Machine Learning.....	107
Gambar 4.21 Potongan Kode Training Machine Learning	107
Gambar 4.22 Arsitektur Sistem Firewall.....	107
Gambar 4.23 Potongan Kode Implementasi Machine Learning pada Firewall Controller.....	108
Gambar 4.24 Potongan Kode Objek Signature.....	108
Gambar 4.25 Potongan Kode Implementasi Firewall	108
Gambar 4.26 Tampilan Program Firewall saat Bekerja.....	109
Gambar 4.27 Arsitektur Sistem Distribusi Data.....	109
Gambar 4.28 Alur Instalasi BigchainDB Network	110
Gambar 4.29 Ilustrasi Infrastruktur BigchainDB Network	110
Gambar 4.30 Tampilan Tendermint Init.....	111
Gambar 4.31 Data public key node dan Node Id	111
Gambar 4.32 Isi File genesis.json Tendermint	112
Gambar 4.33 Konfigurasi Tendermint Config.toml	113
Gambar 4.34 Konfigurasi BigchainDB	114

Gambar 4.35 Konfigurasi Firewall untuk Tendermint	114
Gambar 4.36 Tampilan BigchainDB setelah Berjalan	114
Gambar 4.37 Struktur Folder Backend	115
Gambar 4.38 Arsitektur Modul Backend	115
Gambar 4.39 Potongan Kode Database untuk Operasi Data	116
Gambar 4.40 Potongan Kode Konfigurasi DemocracyJS	116
Gambar 4.41 Potongan Kode Pub/Sub democracyJS	117
Gambar 4.42 Potongan Kode Backend CronJob	117
Gambar 4.43 Tampilan Sistem Backend	118
Gambar 4.44 Arsitektur Sistem Backend	118
Gambar 4.45 Tampilan Sistem Frontend	119
Gambar 4.46 Instalasi Snort	120
Gambar 4.47 Prompt Instalasi Snort	120
Gambar 4.48 Aplikasi Snort.....	120
Gambar 4.49 Isi dari snort.conf pada Aplikasi Snort	121
Gambar 4.50 Custom Ruleset Aplikasi Snort	121
Gambar 4.51 Tampilan Ketika Program Snort Dijalankan	122
Gambar 4.52 Tampilan Pengambilan Data Rata Rata Penggunaan CPU ..	129
Gambar 4.53 Hasil Penelitian Uji Proxy 1.....	129
Gambar 4.54 Hasil Penelitian Uji Proxy 2.....	130
Gambar 4.55 Hasil Penelitian Uji Proxy 3.....	130
Gambar 4.56 Hasil Penelitian Uji Proxy 4.....	131
Gambar 4.57 Hasil Penelitian Uji Proxy 5.....	131
Gambar 4.58 Hasil Penelitian Uji Proxy 6.....	132
Gambar 4.59 Hasil Penelitian Uji Sistem Distribusi 1.....	133
Gambar 4.60 Hasil Penelitian Uji Sistem Distribusi 2.....	133
Gambar 4.61 Hasil Penelitian Uji Sistem Distribusi 3.....	134

DAFTAR LAMPIRAN

Lampiran A. Hasil Turnitin	156
Lampiran B. Form Konsultasi Skripsi Pembimbing 1	159
Lampiran C. Form Konsultasi Skripsi Pembimbing 2.....	160



BAB I

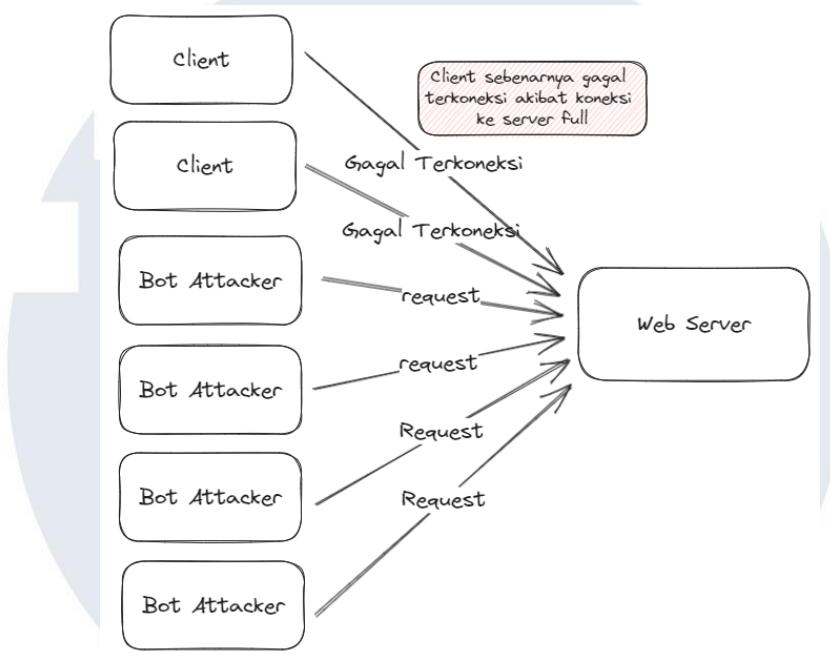
PENDAHULUAN

1.1 Latar Belakang

Penggunaan jaringan komputer sekarang ini semakin luas dan berkembang, ditambah dengan adanya berbagai teknologi layanan web (*web service*) di berbagai bidang kehidupan seperti contohnya adalah *financial technology*, *online game*, *Internet of Things*, *supply chain technology*, dan berbagai bentuk situs web. Layanan web adalah suatu perangkat lunak yang dirancang untuk mendukung interoperabilitas dan interaksi pertukaran data antar beberapa perangkat komputer yang berbeda melalui jaringan internet [1]. Layanan web menggunakan beberapa protokol jaringan komputer sebagai standar spesifikasi web server terlepas dari bahasa pemrogramannya dan sistem operasi yang digunakan oleh setiap perangkat yang berkomunikasi. Biasanya layanan web akan memakai protokol internet seperti layer 4 *transport layer* yaitu TCP / UDP dan layer 7 *application layer* yaitu HTTP / HTTPS. Layer ini dijabarkan dalam 7 OSI layer dalam jaringan komputer. Layanan web sering digunakan untuk integrasi aplikasi, akses jarak jauh, dan berbagi data antar sistem komputer.

Namun disisi lain jika semakin besar jaringan komputer yang dipakai maka akan semakin besar juga risiko yang dapat ditemukan dari suatu implementasi layanan web. Hal ini karena *hacker* dapat menyerang jaringan komputer yang memanfaatkan kelemahan dari jaringan komputer di antara ke 7 OSI layer. Yaitu suatu perangkat server yang menjalankan aplikasi layanan web dapat dengan mudah terserang dengan berbagai metode untuk menghabiskan *resource* atau sumber daya dari perangkat tersebut. Penyerangan ini dikenal dengan teknik *hacking Denial of Service* (DDoS). Seiring perkembangan serangan ini, semakin meluas kapasitas serangannya, dimana sebelumnya menggunakan 1 komputer penyerang menjadi beberapa “bot” komputer penyerang dengan bertindak sebagai beberapa *client* terdistribusi yang me-request ke server tersebut dengan berulang kali dan bersamaan. Serangan ini dikenal dengan *Distributed Denial of Service*

(DDoS) [2]. Sebagai ilustrasi umum serangan DDoS, diilustrasikan pada gambar 1.1.



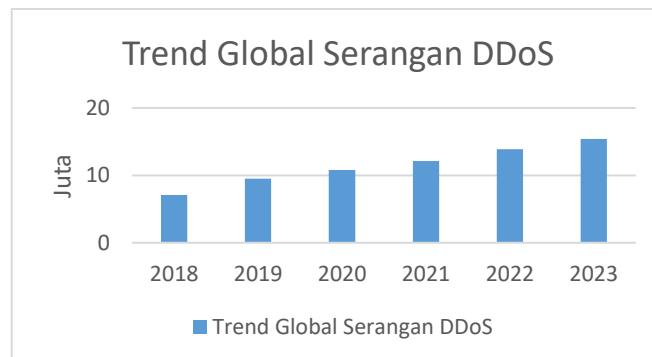
Gambar 1.1 Gambaran Umum Serangan DDoS

Dampak dari serangan DDoS ini menyebabkan aplikasi layanan web mengalami penurunan kinerja sampai gagalnya sistem yang mengakibatkan adanya *downtime* dan tidak bisa diaksesnya sistem tersebut. Oleh karena itu serangan DDoS merupakan ancaman dan masalah utama keamanan internet terlebih pada sistem yang memiliki *single point of failure* seperti penggunaan proxy server yang menjadi satu-satunya perantara antara *client* dan *server*. Jika proxy server atau web server yang memiliki *single point of failure* terserang DDoS maka akan melumpuhkan semua sistem di belakangnya. Hal ini karena akan bukan hanya melumpuhkan layanan web sehingga tidak bisa digunakan namun juga integritas dan kerahasiaan data yang tersimpan di server dapat menjadi rentan akibat serangan DDoS ini [3]. Misalnya seperti teknik *hacking brute-force*. Terlebih serangan DDoS cenderung mudah untuk diluncurkan dari pada serangan siber lainnya yaitu dengan melakukan akses secara berulang kali ke target yang dituju.

Secara umum tujuan dari serangan DDoS adalah menghabiskan *resource* atau sumber daya dari perangkat yang dituju, bisa sumber daya CPU, RAM, atau

network interface. Oleh karena itu metode yang digunakan bisa bermacam-macam, penyerang akan melakukan serangan pada salah satu protokol komunikasi dalam jaringan komputer yaitu antara TCP, UDP, ICMP, HTTP, SSH, FTP, dan sebagainya. Karena metode dan teknik penyerangan DDoS sangat banyak dan terus berkembang, maka suatu serangan DDoS akan membentuk suatu pola stokastik. Pola stokastik ini adalah pola yang menghasilkan suatu kejadian yang acak dan tidak bersifat stasioner namun mempunyai *unit root* [4]. Terlebih dengan metode *IP Spoofing*, penyerang bisa saja menggunakan identitas atau alamat IP palsu sehingga akan cukup kompleks untuk mendeteksi dan mengidentifikasi serangan DDoS.

Kasus penyerangan jaringan komputer dengan metode DDoS sekarang ini semakin meningkat. Berdasarkan data statistik dari beberapa sumber seperti perusahaan Cloudflare dan Cisco, menunjukkan bahwa pada kuartal 2 tahun 2022 terjadi peningkatan serangan DDoS pada layer 7 (*application layer*) sebesar 72% dan pada layer 4 (*transport layer*) meningkat hingga 109% per tahunnya [5]. Berikut merupakan grafik tren serangan DDoS yang ditampilkan pada gambar 1.2.



Gambar 1.2 Grafik Trend Global Serangan DDoS [5]

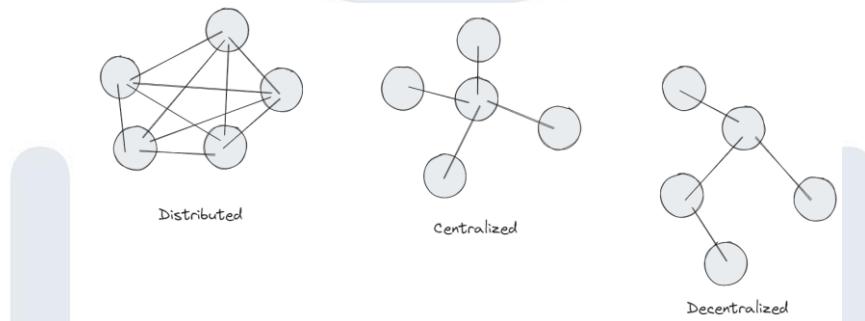
Ada beberapa solusi yang sudah ada untuk memitigasi serangan DDoS yaitu dengan menggunakan sistem *intrusion detection system* (IDS) dan *intrusion prevention system* (IPS). IPS dan IDS beberapa menggunakan *signature based* dan *behavior based* dalam mendeteksi suatu anomali [6]. Dapat ditemukan banyak aplikasi IDS / IPS yang *open-source* dan sudah banyak dipakai, salah satunya adalah aplikasi yang bernama Snort. Namun untuk aplikasi ini metode

pendeteksiannya masih menggunakan metode konvensional yaitu menggunakan algoritma statistika sederhana yaitu sistem “*Threshold*”. Sistem *threshold* adalah membatasi suatu aktivitas jaringan komputer dengan menerapkan batasan tertentu pada interval waktu dan identitas tertentu secara statis. Misal sistem akan melimit suatu IP address hanya boleh memiliki *packet* berjumlah 100 buah setiap 10 detik. Ketika suatu IP address memiliki *packet* lebih dari 100 maka *packet* yang melebihi *threshold*-nya akan dianggap sebagai anomali dan dimitigasi sesuai dengan pengaturannya. Dengan metode tradisional ini memiliki beberapa kekurangan yaitu bisa saja terjadi kesalahan pendeksiyan sehingga menghasilkan *false positive* dan *false negative*. Seperti jika lalu lintas jaringan memang sedang ramai pengguna yang normal akan menjadi masalah karena pengaturannya yang statis.

Teknologi *Machine learning* dapat digunakan untuk menangani masalah peristiwa pola stokastik. Dengan *machine learning* dapat membuat sistem IPS yang menggunakan metode *behavior based*. Setiap serangan DDoS memiliki pola dan variasi tersendiri yang bersifat kompleks dan sulit untuk didapatkan jika hanya menggunakan teori statistik dan observasi manusia. Dengan *machine learning* maka akan didapatkan *features* (informasi penting) tersendiri dari suatu data serangan DDoS. Beberapa metode *machine learning* yang dapat dilakukan yaitu untuk pendekatan secara individual dan *timeseries*, dan juga pendekatan secara *atomic* untuk identifikasi *packet* dan *composite* untuk deteksi DDoS pada *stream* (*flow*) packet jaringan [7]. 7 algoritma *machine learning* dan deep learning yang dipilih oleh penulis berdasarkan hasil tinjauan teori untuk klasifikasi data dapat digunakan dan dibandingkan untuk mendapatkan model *machine learning* yang sesuai untuk deteksi secara *realtime* dengan beberapa matriks pengujian. Hal ini karena setiap protokol memiliki *features*, karakteristik, dan penggunaan yang berbeda beda sehingga diperlukan perlakuan *machine learning* yang berbeda pula karena setiap algoritma *machine learning* memiliki metode kalkulasi matematis yang berbeda.

Seiring perkembangan teknologi, dibutuhkan arsitektur layanan web yang mendukung *redundancy*. Hal ini dapat dicapai dengan arsitektur terdistribusi dan terdesentralisasi yang akan bertumpu pada jaringan komputer. Dengan metode dan

arsitektur ini, layanan web memiliki kumpulan aplikasi dan perangkat yang berbeda beda dan memiliki tugasnya masing-masing. Setiap aplikasi dan perangkatnya saling bergantung satu sama lainnya dan digunakan untuk hal yang krusial dan penting sehingga dibutuhkan keamanan yang lebih agar tingkat ketersediaannya tinggi. Dengan ini, dimungkinkan bahwa suatu layanan web memiliki beberapa perangkat server yang terpisah satu dengan yang lainnya namun dikoneksikan dengan jaringan internet dikarenakan memiliki geolokasi yang berbeda beda. Contoh implementasi dari arsitektur ini adalah seperti *content delivery network*, *API services*, dan *regional game server*. Berdasarkan penelitian yang telah dilakukan oleh IBM Market Research yang diikuti oleh 1200 tenaga IT yang berpengalaman, bahwa lebih dari 80% perusahaan yang ada sudah beralih pada sistem *microservice* dan *decentralized computing* [8]. Adapun arsitektur sistem terdistribusi (*distributed*). Dimana walaupun sistem dibagi bagi menjadi beberapa *service / node* tertentu namun tetap ada *central control* untuk mengatur semuanya seperti melakukan konsensus dan lain lain [9]. Berikut merupakan ilustrasi arsitektur *centralized*, *decentralized*, dan *distributed* yang ditampilkan pada gambar 1.3.

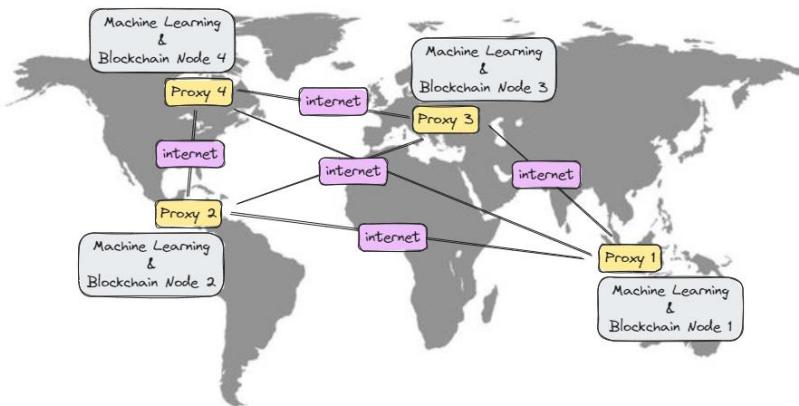


Gambar 1.3 Ilustrasi Perbedaan Sistem *Distributed*, *Centralized*, dan *Decentralized*

Oleh karena arsitektur terdesentralisasi dan terdistribusi maka antar IPS perlu membagi informasi penyerang agar menambah efektivitas dari keuntungan sistem terdesentralisasi. Agar jika suatu penyerang menyerang sistem 1, maka sistem 2, 3, dan 4 sudah memblokir penyerang sehingga *time to mitigate*-nya adalah 0. Namun karena antar IPS bisa saja menggunakan jalur komunikasi *public* internet karena perbedaan geolokasi maka perlu digunakan *database* yang terdistribusi dan

memiliki kemampuan *immutability* yaitu salah satunya adalah *database* berbasis *blockchain*. Kemampuan *immutability* penting agar tidak terjadi penyerangan pada sistem *database* dan mengubah konfigurasi *firewall*. Dibutuhkan juga sistem konsesus yang cepat agar informasi penyerang tersebar ke seluruh IPS sedini mungkin.

Oleh karena beberapa dasar permasalahan tersebut maka penelitian ini akan merancang dan membangun sistem proxy untuk memitigasi serangan DDoS dengan teknologi *machine learning* dan *blockchain*. Proxy akan bertindak sebagai penengah antara *client* dengan *webserver*, sehingga dapat menjadi *listener* dan juga *firewall*. Dengan begitu *webserver* tidak secara langsung terhubung dengan *public internet*. Proxy akan dirancang menjadi proxy terdistribusi yang berbeda secara geolokasi untuk memenuhi kebutuhan arsitektur *decentralized computing* dan *distributed computing*. Ilustrasi proxy terdistribusi dapat dilihat pada gambar 1.4.



Gambar 1.4 Ilustrasi Proxy Terdistribusi

Berdasarkan masalah dan rancangan solusi yang sudah dipaparkan, maka dengan latar belakang ini penulis akan melakukan penelitian dengan judul “Rancang Bangun Sistem Proxy Mitigasi DDoS berbasis Machine Learning dan Blockchain”.

1.2 Identifikasi Masalah

Berdasarkan latar belakang yang sudah dipaparkan, rumusan masalah pada penelitian ini terdiri dari beberapa poin, yaitu :

- 1.2.1 Apakah sistem proxy mitigasi DDoS berbasis *machine learning* dapat mengamankan layanan web dari serangan DDoS secara efektif, keseluruhan, dan akurat yang diukur dengan metrik pengujian tertentu jika dibandingkan dengan solusi IPS / IDS yang *open-source* dan banyak dipakai sekarang?
- 1.2.2 Model dan metode *machine learning* apakah yang cocok untuk diimplementasikan pada deteksi dan identifikasi serangan DDoS secara *realtime*?
- 1.2.3 Apakah dengan membuat sistem proxy mitigasi DDoS yang menggunakan *database* terdistribusi berbasis *blockchain* dapat menambah efektivitas pada sisi *time to mitigate* dari proxy terdistribusi?

1.3 Batasan Penelitian

Berdasarkan identifikasi masalah yang sudah disebutkan, berikut merupakan batasan masalah dari penelitian ini :

- 1.3.1 Jenis serangan DDoS yang diteliti hanya berjenis TCP Syn dan XMAS flood, UDP flood, ICMP ping dan Smurf flood, dan HTTP Get flood pada jaringan IPV4. Beberapa jenis metode DDoS ini dan aktivitas jaringan normal disimulasikan dengan *testbed*, kode, dan *tools* yang dipilih oleh penulis.
- 1.3.2 Keseluruhan sistem yang dibangun hanya diuji coba pada *testbed* dan uji skenario yang sudah ditetapkan oleh penulis yang dijabarkan pada bab analisis dan perancangan sistem untuk mendapatkan hasil dan analisa pengujian.

- 1.3.3 *Hyper-parameter* model *machine learning* yang digunakan merupakan pilihan penulis berdasarkan penelitian terdahulu, dasar teori, dan hasil observasi penulis.

1.4 Tujuan Penelitian

Berikut beberapa tujuan dari penelitian ini, yaitu :

- 1.4.1 Merancang dan membangun sistem proxy mitigasi DDoS berbasis *machine learning* dan *blockchain*.
- 1.4.2 Membandingkan sistem proxy mitigasi DDoS yang sudah dibangun dengan solusi IPS / IDS *open-source* dan banyak digunakan sekarang dengan uji skenario dan metrik pengujian tertentu.
- 1.4.3 Menguji coba beberapa metode dan algoritma *machine learning* untuk mengklasifikasi data deteksi dan identifikasi serangan DDoS berdasarkan protokol TCP, UDP, ICMP, dan HTTP pada jaringan IPV4.
- 1.4.4 Merancang, membangun, dan menganalisis proxy yang terdistribusi untuk menunjang perkembangan sistem *distributed computing*.

1.5 Manfaat Penelitian

Manfaat dari penelitian ini adalah sebagai berikut :

- 1.5.1 Menyediakan sistem proxy mitigasi DDoS berbasis *machine learning* dan *blockchain* sebagai solusi dan referensi untuk mengatasi masalah serangan DDoS yang semakin meningkat.
- 1.5.2 Dapat digunakan sebagai sumber referensi dan acuan dalam hasil penelitian dan analisis untuk modul *machine learning* yang digunakan untuk mendeteksi dan mengidentifikasi serangan DDoS secara *realtime*.
- 1.5.3 Dapat digunakan sebagai sumber informasi dalam pembuatan sistem terdistribusi dengan *distributed database* berbasis *blockchain*

1.5.4 Mendorong perkembangan dan penggunaan teknologi internet, *machine learning*, dan *blockchain* pada berbagai bidang.

1.6 Sistematika Penulisan

Laporan penelitian ini disusun dengan beberapa bagian untuk mempermudah pembacaan dan pemahaman pada bahasan penelitian ini.

1.6.1 Bab 1 Pendahuluan

Pada bab ini akan membahas tentang latar belakang, rumusan masalah, batasan penelitian, tujuan penelitian, dan manfaat penelitian dari penelitian yang akan dilakukan oleh penulis.

1.6.2 Bab 2 Tinjauan Pustaka

Pada bab ini akan membahas tentang penulis melakukan pencarian dan mempelajari penelitian terdahulu serta teori terkait pada penelitian yang akan dilakukan guna untuk referensi penelitian dan perbaikan untuk mengoptimalkan sistem yang akan dibuat oleh penulis seperti karakteristik serangan DDoS, *features* dataset, dan metode *machine learning* apa saja yang dapat digunakan untuk deteksi dan identifikasi serangan DDoS. Selain itu akan dibahas juga deskripsi dari teknologi yang akan dipakai penulis seperti NetfilterQueue, Python, Socket, BigchainDB, Bidirectional LSTM, SVM, Linear Regresion, dan lain lain.

1.6.3 Bab 3 Analisis dan Perancangan Sistem

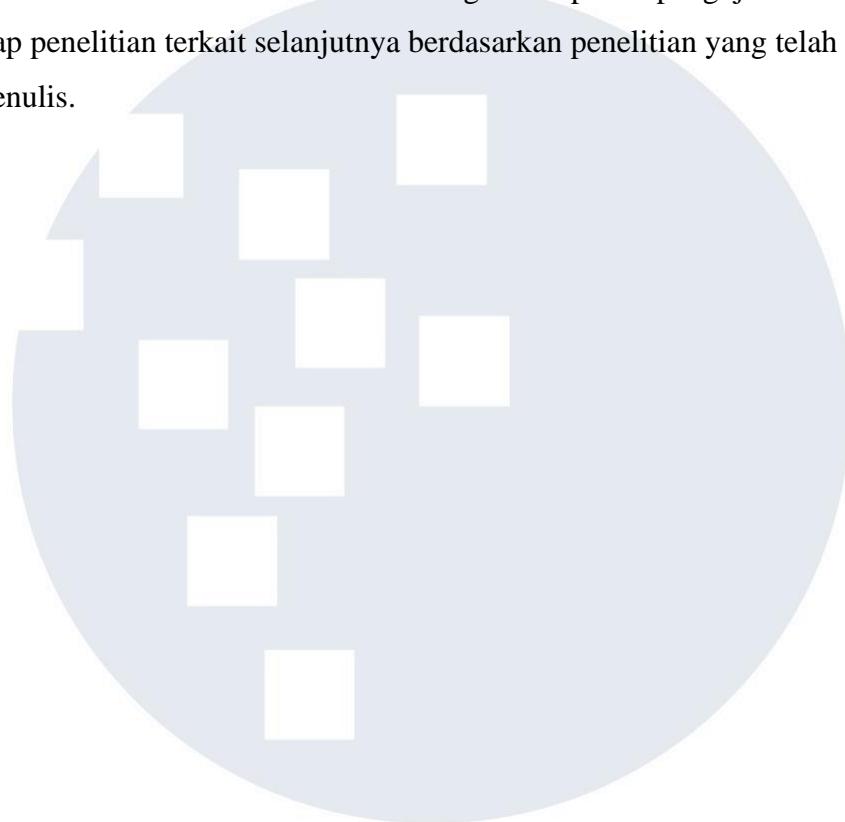
Pada bab ini akan membahas tentang penulis melakukan perancangan umum dari keseluruhan sistem yang akan dibuat oleh penulis, mulai dari arsitektur sistem, cara kerja, sub sistem dan modul yang akan diimplementasikan, skenario uji coba, dan penentuan metrik pengujian.

1.6.4 Bab 4 Implementasi dan Pengujian Sistem

Pada bab ini akan membahas tentang penulis melakukan implementasi dan pengujian berupa hasil penelitian dan analisa berdasarkan skenario uji coba dan metrik pengujian pada sistem yang dibangun. Berisi juga tentang kendala dan solusi terhadap masalah saat proses implementasi yang dilakukan oleh penulis.

1.6.5 Bab 5 Simpulan dan Saran

Pada bab ini akan membahas tentang kesimpulan pengujian beserta saran terhadap penelitian terkait selanjutnya berdasarkan penelitian yang telah dilakukan oleh penulis.



UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA

BAB II

TINJAUAN PUSTAKA

2.1 Penelitian Terdahulu

Terdapat beberapa penelitian terkait dengan arsitektur dan implementasi teknologi proxy, *machine learning*, dan *blockchain* dan definisi dan karakteristik serangan DDoS sebagai referensi perancangan dan pembangunan sistem proxy mitigasi serangan DDoS berbasis *machine learning* dan *blockchain*.

2.1.1 *A Comprehensive Survey on DDoS Attacks on various Intelligent Systems and It's Defense Techniques [10]*

Penelitian dengan judul “*A Comprehensive Survey on DDoS Attacks on various Intelligent Systems and It's Defense Techniques*” yang dilakukan oleh Akshat Gaurav dilakukan dengan tujuan untuk memberikan penjabaran lengkap tentang analisis arsitektur serangan DDoS secara umum dengan memberikan beberapa contoh serangan yang umum dilakukan. Di penelitian ini juga diangkat beberapa solusi yang bisa dilakukan untuk memitigasi serangan DDoS. Selain itu peneliti juga menyebutkan bahwa belum ditemukan cara yang sesuai dan efektif untuk mendeteksi dan memfilter serangan DDoS.

DDoS adalah merupakan salah satu tipe dari teknik hacking yang memiliki tujuan untuk menghabiskan sumber daya dari perangkat targetnya dengan mengirimkan terus menerus paket yang tidak sesuai. DDoS memiliki beberapa motif dan fase dalam penyerangannya, seperti memilih target, menyiapkan komputer bot, dan memilih teknik DDoS. Komputer bot yang dipakai bisa saja akan berbeda beda secara geolokasi.

Beberapa tipe dalam serangan DDoS

1. *Bandwidth Attack*

Jenis ini memiliki tujuan untuk mengambil seluruh *bandwidth network interface* dan kekuatan komputasi perangkat targetnya.

Metode yang cukup sering ditemukan adalah UDP Flood, HTTP Flood, dan ICMP Flood.

a. UDP Flood

Penyerang akan mengirimkan jumlah paket UDP yang sangat banyak dan *payload* yang besar ke salah satu atau beberapa *port* perangkat target sekaligus. Jika paket UDP diterima oleh perangkat targetnya, maka jika tidak ada aplikasi yang memproses paket tersebut maka akan direspon dengan paket ICMP. Jika dilakukan terus menerus akan mengkonsumsi sumber daya dari perangkat targetnya untuk terus merespons penyerang.

b. HTTP Flood

Penyerang akan mengirimkan paket *request* berupa HTTP GET atau POST, dan web server akan merespons dengan mengirimkan data yang sesuai dengan apa yang diminta. Akan diulangi terus menerus tanpa ada selang waktu. HTTP merupakan OSI layer protokol layer 7 (*application layer*) yang menggunakan TCP sebagai layer 4 nya (*transport layer*).

c. ICMP Flood

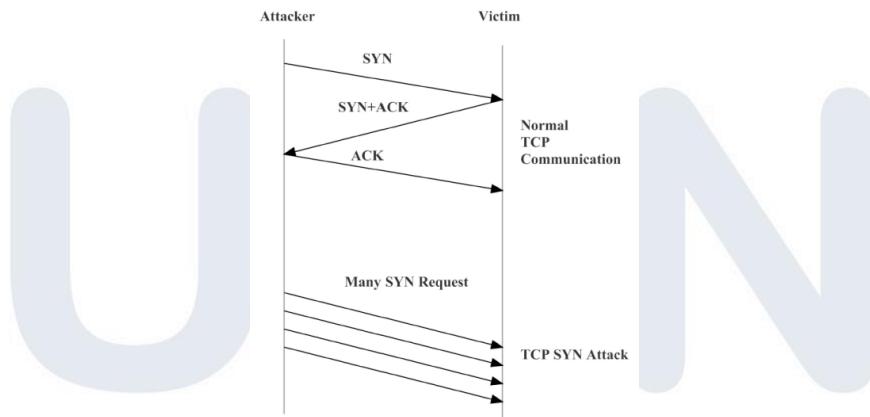
Tipe ICMP yang dipakai adalah ICMP_ECHO_REPLY, jika paket tersebut dikirimkan maka perangkat tujuan akan mengirimkan kembali paket tersebut. Namun penyerang akan mengirimkan paket ini terus menerus dengan selang waktu yang sangat kecil dan jumlah yang banyak.

2. *Resource Depletion Attack*

a. TCP-SYN Flood (*protocol exploitation attack*)

Penyerang akan memanfaatkan cara kerja protokol TCP yaitu *three-way handshake*. Secara normalnya *client* pertama kali akan mengirimkan paket SYN ke server, lalu

server akan menyimpan *IP address* sumber ke dalam tabel antrean. Setelah *IP address* sumber berada di antrean paling depan, server akan meresponsnya dengan ACK + SYN beserta *payload* data yang diminta. Lalu iterasi berikutnya adalah Ketika *client* menerima paket ACK + SYN dari server maka *client* harus merespons dengan paket ACK. Selama *client* tidak mengirimkan paket ACK, server akan terus menunggu dan tidak melanjutkan tabel antrean, sampai batas waktu (*timeout*) yang sudah dikonfigurasi. Hal ini yang dimanfaatkan oleh penyerang, penyerang akan mengirimkan paket SYN terus menerus ke server untuk menghabiskan tabel antrean tersebut. Selain dari TCP SYN flood ada PUSH + ACK flood yang akan meng-*overflow* / membanjiri *memory buffer* server dengan *payload* dari ACK *client* yang berukuran besar. Untuk ilustrasi dari serangan TCP SYN diilustrasikan pada gambar 2.1.



Gambar 2.1 Ilustrasi serangan TCP SYN [10]

b. *Ping of Death (Malformed Packet Attack)*

Penyerang akan mengirimkan paket ICMP dengan ukuran yang besar secara berulang kali dan tanpa adanya selang waktu. Ukuran normal dari paket ICMP adalah 64

bytes. Hal ini akan membebani perangkat target dan sumber daya perangkatnya akan habis.

Penelitian ini juga memaparkan beberapa cara untuk melakukan tindakan preventif untuk menangani serangan DDoS. Beberapanya adalah load balancing, throttling, honeypots, firewalls, IP Hopping, dan SOS. Honeypots adalah menggunakan metode *logging* sebagai tempat untuk menyimpan informasi agar nantinya dapat digunakan untuk mengidentifikasi penyerang yang nantinya dianalisis dan diproses oleh IPS / IDS. *Firewall* dapat digunakan untuk memfilter paket yang tidak diinginkan namun tidak dapat secara langsung membedakan mana paket yang berbahaya atau normal.

Lalu dipaparkan juga salah satu rencana *attack filtering* yang dapat dilakukan adalah *Reactive & Cooperative*. Yaitu proses filter paket yang berbahaya hanya akan dilakukan jika hanya terdeteksi adanya serangan DDOS dan tetap melakukan *logging* ketika adanya serangan. Dan 2 atau lebih *router* atau *network controller*-nya akan melakukan pembagian informasi terkait identifikasi penyerang.

Beberapa poin penting yang dapat diambil oleh penulis adalah sebagai berikut :

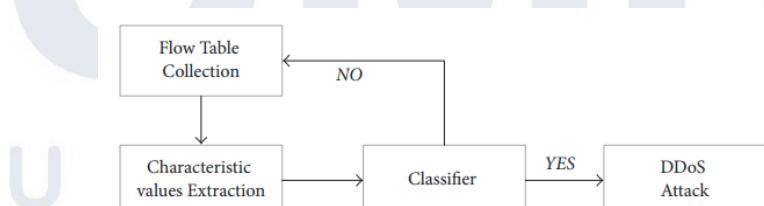
1. Penulis mengetahui tentang jenis serangan DDoS yang umum dilakukan dan cara kerja serangannya. Yaitu UDP flood, HTTP flood, ICMP flood, TCP Syn flood, TCP PUSH + ACK flood, Ping of Death Flood.
2. Dapat diimplementasi teknik honeypots, dimana melakukan *logging* setiap paket yang berlalu untuk nantinya dianalisis dan diproses oleh IDS atau IPS.
3. Dapat diimplementasi teknik *firewall*, dimana melakukan *filtering packet* serangan DDoS dengan mengatur *firewall rule* yang sesuai dengan identifikasi serangan DDoS.

- Dapat diimplementasi teknik *Reactive & Cooperative* untuk melakukan proses *filtering* jika hanya sedang keadaan DDoS dan selalu melakukan *logging* terhadap suatu paket. Lalu dilakukan pertukaran informasi berupa *logs* dan identitas penyerang pada setiap *router* atau *network controller*-nya.

2.1.2 A DDoS Attack Detection Method Based on SVM in Software Defined Network [11]

Penelitian dengan judul “A DDoS Attack Detection Method Based on SVM in Software Defined Network” yang dilakukan oleh Jin Ye, Xiangyang Cheng, Jian Zhu, Luting Feng, dan Ling Song akan mempraktikkan *machine learning* berupa algoritma SVM ke dalam deteksi serangan DDoS secara *realtime*. Didapatkan 6 *features* yang menjadi dataset dari *machine learning*. Data *features* tersebut di dapatkan dari *controller (control plane)* SDN (*Software Defined Network*) yaitu mininet, openFlow, dan floodlight. Pendekripsi disimulasikan dengan jumlah dataset yang sedikit, namun model *machine learning* SVM mendapatkan akurasi rata-rata sebesar 95.24 % pada 3 protokol yaitu TCP, UDP, dan ICMP.

Penelitian ini melakukan pendekripsi serangan DDoS dengan pendekatan individual yaitu menemukan suatu anomali pada *stream packet* dengan rentang waktu tertentu. Dilakukan beberapa proses pendekripsi yang dapat dilihat pada gambar 2.2.



Gambar 2.2 Proses Deteksi Serangan DDoS secara individual [11]

Penjabaran dari proses deteksi serangan DDoS pada gambar 2.2 adalah sebagai berikut :

1. Flow Status Collection

Data paket diambil dari *control plane* SDN yang berfungsi untuk menyalurkan paket dari sumber paket ke tujuan paket, SDN yang digunakan yaitu OpenFlow SDN. Akan dilakukan pengambilan data secara periodik setiap 5 detik untuk nantinya akan diproses informasinya.

2. Characteristic Extraction

Pada penelitian ini dipakai 6 *features* yang dijadikan data untuk mengklasifikasi suatu anomali. Untuk ke 6 *featuresnya* dijabarkan pada tabel 2.1.

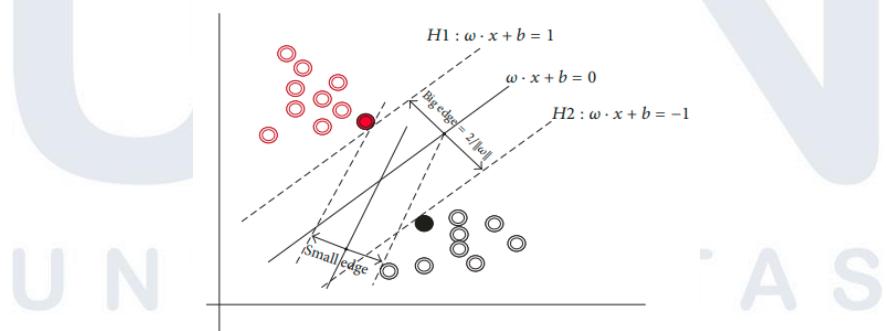
Tabel 2.1 Features yang Digunakan dalam Deteksi Serangan DDoS menggunakan SVM [11]

No	Nama	Rumus	Deskripsi
1	SSIP (<i>Speed of source IP</i>)	$SSIP = \frac{\text{Sum_IP src}}{T}$	Ketika serangan jumlah <i>IP address</i> akan semakin banyak
2	SSP (<i>Speed of source port</i>)	$SSP = \frac{\text{Sum_port src}}{T}$	Ketika serangan jumlah sumber <i>port</i> akan semakin banyak
3	SDFP (<i>Standar Deviation of Number of Packet</i>)	$SDFP = \sqrt{\frac{1}{N} \sum_{i=1}^N (\text{packets } i - \text{mean_packet})^2}$	Ketika serangan variasi jumlah paket akan kecil daripada normalnya karena biasanya serangan akan menggunakan data yang sama
4	SDFB (<i>Standar Deviation of the Flow Bytes</i>)	$SDFB = \sqrt{\frac{1}{N} \sum_{i=1}^N (\text{bytes } i - \text{mean_bytes})^2}$	Ketika serangan variasi besar paket akan kecil daripada normalnya karena biasanya serangan akan menggunakan data yang sama

5	SFE (<i>Speed of Flow Entries</i>)	$SFE = \frac{N}{T}$	Jumlah keseluruhan paket dalam rentang satuan waktu
6	RPF (<i>Ratio of Pair Flow</i>)	$RPF = \frac{2 * \text{Pair_Sum}}{N}$	Jumlah paket yang bersifat 2 arah. Untuk menentukan paket yang 2 arah atau tidak, dapat dilihat pada paket lain yang memiliki destinasi IP paket tersebut.

3. Classifier

Pada bagian ini dijabarkan juga alasan penelitian ini menggunakan algoritma *machine learning* SVM. SVM digunakan karena menggunakan teori statistik untuk pembelajarannya, bisa mendapatkan akurasi yang bagus ketika jumlah dataset yang sedikit, kernel SVM dapat mengatasi masalah *high-dimensional mapping*, dan hemat untuk proses komputasinya. Kernel SVM yang digunakan yaitu linear, sehingga untuk ilustrasi pengklasifikasianya dapat dilihat pada gambar 2.3.



Gambar 2.3 Ilustrasi Klasifikasi Algoritma *Machine Learning* SVM [11]

Jika dibandingkan dengan melakukan klasifikasi data menggunakan *deep learning* seperti model ANN, maka akan

cenderung terlalu kompleks dan tidak efisien dalam proses komputasi.

Peneliti penelitian ini menyimulasikan keadaan serangan DDoS dan normal menggunakan *tools* Hping3 dan mendapatkan dataset berjumlah 200, 600, dan 1000 *sample*. Pada hasil penelitian ini didapatkan bahwa untuk protokol TCP mendapatkan akurasi sebesar 96.83%, UDP sebesar 95.24%, dan ICMP sebesar 93.65%. ICMP memiliki akurasi yang paling kecil karena dari 6 *features* yang digunakan, hanya tersisa efektif 4 *features* karena paket ICMP tidak menggunakan *features* SSP karena tidak menggunakan *port* dan SDFP karena setiap 1 kali komunikasi ICMP pasti hanya menggunakan 1 paket saja.

Beberapa poin penting yang dapat diambil oleh penulis adalah sebagai berikut:

1. Menggunakan pendekatan individual untuk menemukan suatu anomali pada *stream packet* dengan rentang waktu 5 detik.
2. Menggunakan 6 *features* yaitu SSIP, SSP, SDFP, SDFB, SFE, dan RPF yang didasari oleh kalkulasi statistik standar deviasi.
3. Untuk mengklasifikasikan suatu anomali dapat menggunakan algoritma *machine learning* SVM dengan kernel linear karena dapat mengatasi masalah *high-dimensional mapping*, dan hemat untuk proses komputasinya. Di sisi lain untuk algoritma *deep learning* seperti model ANN tidak diteliti karena akan terlalu kompleks dan tidak efisien secara komputasi.
4. Menggunakan SDN OpenFlow sebagai *listener* dan *logger* paket yang akan diteliti.

2.1.3 *LSTM-BA: DDoS Detection Approach Combining LSTM and Bayes* [12]

Penelitian dengan judul “*LSTM-BA: DDoS Detection Approach Combining LSTM and Bayes*” yang dilakukan oleh Yan Li dan Yifei Lu memiliki tujuan untuk mendeteksi serangan DDoS menggunakan 2 model

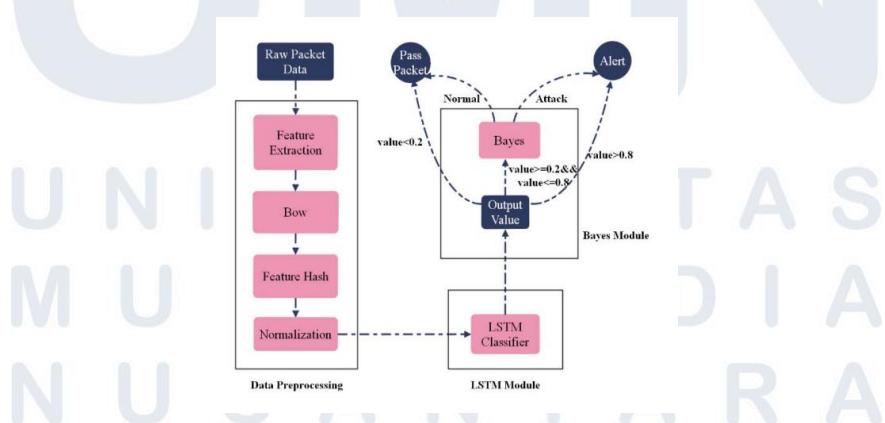
machine learning yaitu LSTM dan Bayes. Digunakan 2 algoritma *machine learning* karena jika saat fase deteksi pertama yang menggunakan LSTM memiliki keluaran level *confident* yang kecil maka akan dilakukan deteksi kedua menggunakan Bayes. Penelitian ini menggunakan publik dataset yaitu ISCX2012. Hasil penelitian ini menghasilkan model *machine learning* dengan akurasi sebesar 98.15%.

Penelitian ini menggunakan pendekatan metode *machine learning* timeseries yaitu beberapa paket jaringan komputer dikumpulkan sesuai dengan lebar *time window machine learning* yang digunakan lalu dideteksi menggunakan model LSTM.

LSTM adalah salah satu algoritma *deep learning* untuk *recurrent* model, terdapat *repeating cell chain* yang nantinya suatu keluaran *cell* akan menjadi input dari *cell* lainnya. LSTM dipakai karena bagus dalam mendeteksi secara *realtime* dan juga memiliki akurasi yang bagus karena dapat menyimpan informasi atau *features* dari waktu yang lama ke belakang.

Bayes disisi lain digunakan karena merupakan algoritma *machine learning* yang cukup sederhana namun bisa mendapatkan akurasi yang tinggi namun cukup buruk jika harus mendeteksi secara *realtime*.

Penelitian ini memiliki proses deteksi yang dapat dilihat pada gambar 2.4.



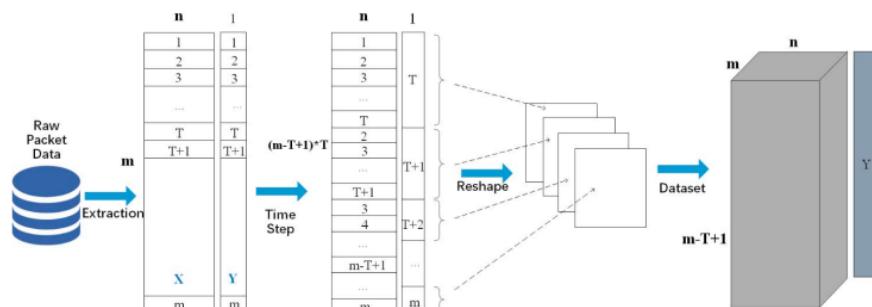
Gambar 2.4 Proses Deteksi menggunakan LSTM [12]

Penjabaran dari proses deteksi serangan DDoS pada gambar 2.4 adalah sebagai berikut :

1. Data preprocess

Pada dataset yang digunakan yaitu ISCX2012, diambil 10 *features* yaitu *Source IP Address*, *Destination IP Address*, *Source Port*, *Destination Port*, *Protocol Type*, *Timestamp*, *Duration*, *Type of service*, *Length*, *Time to live*.

Karena untuk sebagai *input* ke model LSTM memerlukan *input 3 dimensional matrix* yaitu *batch size*, *timestep*, dan *input dimension*, maka perlu dilakukan transformasi dataset. Untuk proses transformasi dataset, diambil paket sejumlah *time window* tertentu lalu diambil label paket terakhir dan dijadikan 1 sample dataset. Begitu seterusnya hingga paket terakhir. Untuk ilustrasinya ditampilkan pada gambar 2.5.



Gambar 2.5 Proses Transformasi Dataset LSTM [12]

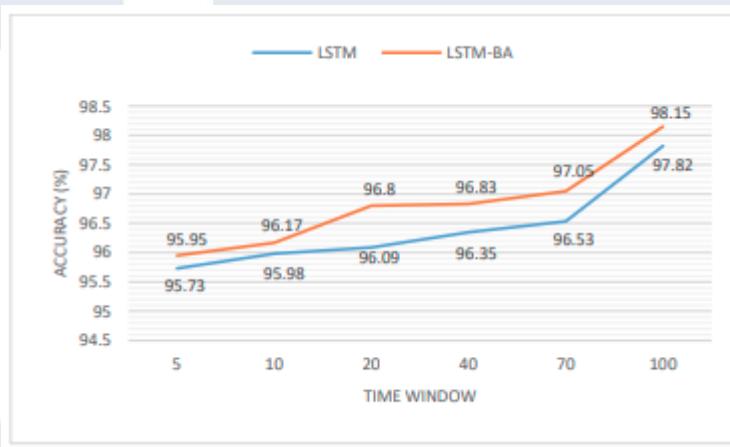
Jika dilihat pada gambar 2.5 diatas, contoh untuk packet 1,2,3 dijadikan 1, lalu label yang dipakai adalah label paket 3. Selanjutnya diulangi untuk paket 2, 3, dan 4 dan menggunakan label paket 4. Pada penelitian ini dipakai 6 buah konfigurasi *time window* yang berbeda yaitu 5, 10, 20, 40, 70, dan 100.

2. LSTM Module

Layer model LSTM yang dipakai adalah untuk layer pertama menggunakan aktivasi tanh, menggunakan 2 *hidden layer* yang

berjumlah 256 layer, *fully connected layer* berjumlah 256 neuron dan aktivasi ReLu, dan 1 *output* layer dengan aktivasi Sigmoid. Aktivasi terakhir berupa Sigmoid digunakan karena bagus untuk *binary classification*, yaitu kelas 0 atau 1.

Dengan menggunakan 120.000 data normal dan 120.000 data serangan DDoS dengan pembagian dataset *training* dan *testing* sebesar 80:02, penelitian ini mendapatkan akurasi tertinggi yaitu 98.15% dengan *time window* 100. Korelasi antara besar *time window* dengan akurasi yaitu semakin besar *time window* maka semakin bagus akurasi yang didapatkan. Untuk grafik perbandingan akurasi dengan *time window* dapat dilihat pada gambar 2.6.



Gambar 2.6 Korelasi antara *time window* dengan akurasi model *machine learning* LSTM [12]

Beberapa poin penting yang dapat diambil oleh penulis adalah sebagai berikut:

1. Menggunakan pendekatan *machine learning* imeseries untuk melakukan pendeketian serangan DDoS menggunakan algoritma *deep learning* yaitu LSTM karena bagus dalam mendeteksi secara *realtime* dan juga memiliki akurasi yang bagus karena dapat menyimpan informasi atau *features* dari waktu yang lama ke belakang.

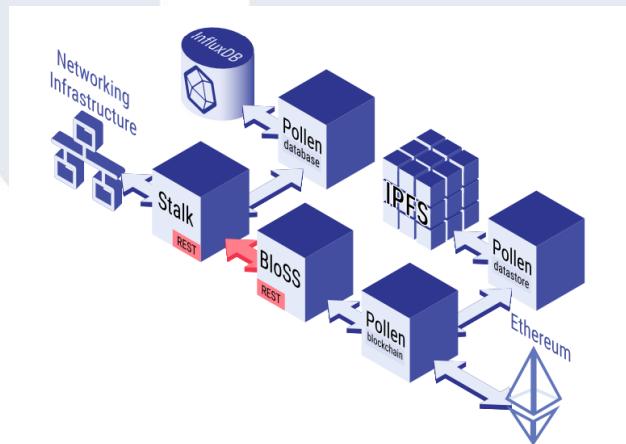
2. Digunakan 10 *features* dari dataset yaitu *Source IP Address*, *destination IP address*, *source port*, *destination port*, *protocol type*, *timestamp*, *duration*, *type of service*, *length*, dan *time to live*.
3. Model LSTM memerlukan *input* berupa 3 *dimensional matrix* yaitu *batch size*, *time step*, dan *input dimension*. Oleh karena itu perlu dilakukan data transformasi pada dataset *timeseries*.
4. Digunakan aktivasi layer terakhir berupa Sigmoid yang bagus untuk *binary classification*, dimana kelas label dataset berupa 0 atau 1.
5. Menggunakan perbandingan dataset *training* dengan dataset *testing* berupa 80:20.
6. Terdapat korelasi antara besar *time window* dengan akurasi model yaitu semakin besar *time window* dari model LSTM maka akurasi yang didapat akan semakin besar.

2.1.4 *Blockchain Signaling System (BloSS): Cooperative Signaling of Distributed Denial-of-Service Attacks* [13]

Penelitian dengan judul “*Blockchain Signaling System (BloSS): Cooperative Signaling of Distributed Denial-of-Service Attacks*” yang dilakukan oleh Bruno Rodrigues, Eder Scheid, Christian Killer, Muriel Franco, dan Burkhard Stiller memiliki tujuan untuk melakukan pembagian data penyerang antar SDN dengan menggunakan teknologi *blockchain* secara aman, automasi, dan efektif. Untuk menangani masalah DDoS secara efektif dibutuhkan metode *cooperative* antar SDN dan saling berkomunikasi untuk memaksimalkan pemblokiran koneksi penyerang. Ada beberapa faktor yang diangkat pada sistem distribusi SDN sekarang yaitu :

1. tingginya kompleksitas dari operasi dan koordinasi antar SDN karena tidak adanya automasi (perlu ditambahkan manual oleh user).
2. Membutuhkan koneksi yang aman dan terpercaya antar SDN

Untuk teknologi *blockchain* yang dipakai adalah *private network* berbasis Etherium yaitu GETH dengan menggunakan *blocksize* sebesar 1MB dengan *throughput* sebesar 5,76GB per hari dan *blockGenTime* sebesar 15 detik setiap *block*-nya. Hal ini dilakukan untuk mendapatkan *time delay* sekecil mungkin namun masih bisa digunakan untuk menyimpan informasi dari penyerang. Informasi penyerang yang diambil dari SDN Stalk dan dibagikan ke *blockchain* yaitu *IP address ranges, packet captures, request headers, notes*, dan pola lainnya. Semua informasi identitas penyerang akan dimasukkan ke dalam IPFS. Secara general arsitektur yang dibangun digambarkan dengan ilustrasi pada gambar 2.7.



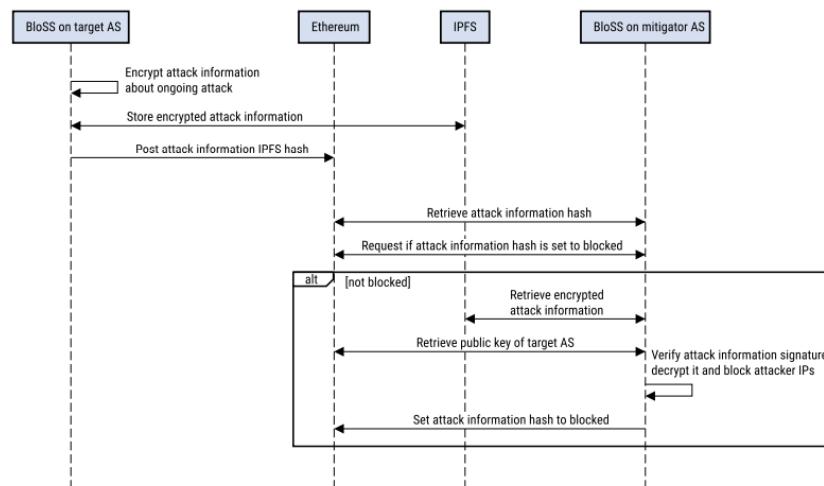
Gambar 2.7 Arsitektur Pembagian Informasi Penyerang menggunakan Blockchain dan IPFS [13]

Penjelasan dari arsitektur pada gambar 2.7 adalah, digunakan IPFS untuk menyimpan informasi penyerang seperti *IP Address* yang sudah diencrypt sebelumnya, lalu *hash* dari IPFS akan disimpan pada *blockchain* menggunakan *PublicKey* dan *PrivateKey* akun yang melapor, lalu akan disalurkan pada backend BLOSS dan digunakan pada firewall infrastruktur *network*. Dalam hal ini *firewall rule* disimpan pada database influxDB. Dengan ini semua individu yang memiliki *smart contract* yang sama dengan *blockchain* akan mendapatkan informasi penyerang yang barusan dilaporkan oleh pelapor dan akan diimplementasikan pada *firewall rule*.

masing masing individu. Untuk lebih jelasnya *sequence diagram* dari penelitian ini ditampilkan pada gambar 2.8.

BLOSS disini adalah sebagai backend dan kontrol sistem yang digunakan untuk mempublish data penyerang ke IPFS dan *blockchain* dan mengkonfigurasi *firewall rule* yang didapatkan dari IPFS dan blockchain ketika individu lain mempublish *firewall rule* baru.

Hasil penelitian ini yaitu dengan 11 skenario pengujian mendapatkan waktu rata-rata untuk *processing time* yaitu 96,95 detik atau 1,5 menit untuk menginput ke dalam *blockchain*. Dan membutuhkan dana sebesar 1 GWEI untuk prioritas standar dan 20 GWEI untuk prioritas tinggi.



Gambar 2.8 *Sequence Diagram* Pembagian Informasi Identitas Penyerang menggunakan *Blockchain* dan IPFS [13]

Beberapa point penting yang dapat diambil oleh penulis adalah sebagai berikut:

1. Data informasi penyerang yang dikirim dan disimpan pada IPFS adalah berupa *IP Address* penyerang.
2. Dengan menggunakan teknologi *blockchain private network* berbasis Etherium menggunakan *tools* GETH, didapatkan hasil *time processing* selama 96,95 detik dan membutuhkan 1 – 20 GWEI untuk 1 transaksi.

3. Dibutuhkan backend sebagai sentral sistem yang akan mempublish informasi penyerang ke IPFS dan *blockchain* dan menerapkan *firewall rule* yang diambil dari IPFS dan *blockchain*
4. Dapat diimplementasikan alur dan cara kerja sistem untuk membuat proxy terdistribusi yang saling bertukar informasi identitas penyerang

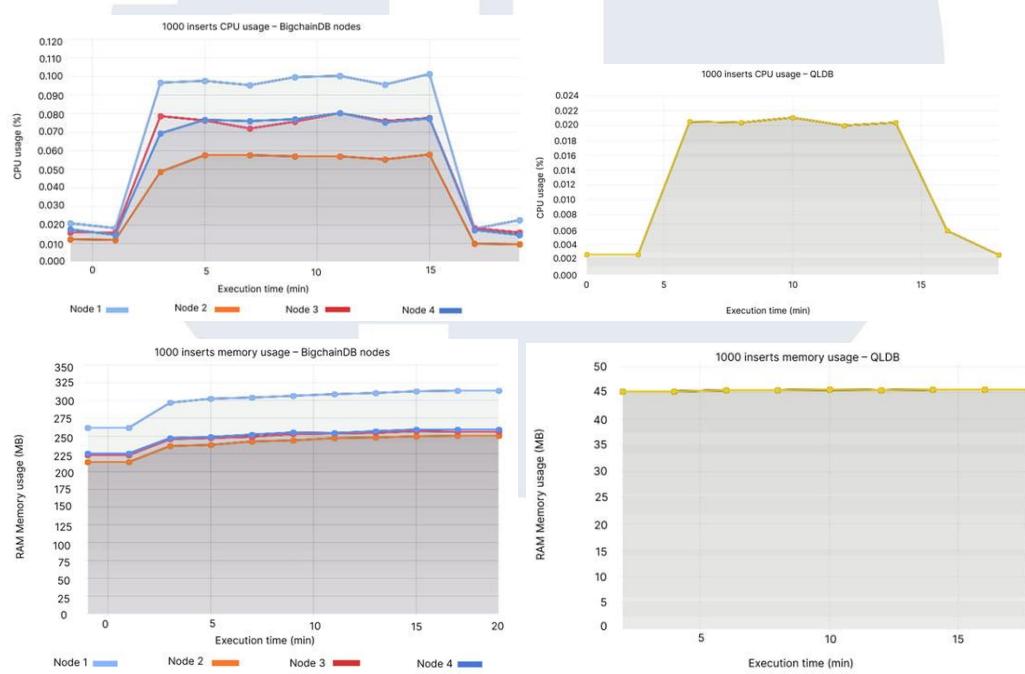
2.1.5 *Centralized vs Decentralized: Performance Comparison between BigchainDB and Amazon QLDB [14]*

Penelitian dengan judul “*Centralized vs Decentralized: Performance Comparison between BigchainDB and Amazon QLDB*” yang dilakukan oleh Sergiu Lupăiescu, Petru Cioata, Cristina Elena Turcu, Ovidiu Gherman, Cornelius Octavian Turcu, dan Gabriela Paslaru memiliki tujuan untuk menemukan karakteristik dari 2 arsitektur *database* yang berbeda, yaitu perbandingan antara BigchainDB yaitu *database* terdesentralisasi berbasis *blockchain* dan mongoDB dan Amazon QLDB yaitu *database* tersentralisasi namun memiliki kemampuan *immutability*. Dari hasil penelitian ini ditemukan bahwa Amazon QLDB memiliki rata-rata performa yang lebih tinggi dari pada BigchainDB, namun BigchainDB di satu sisi memiliki fleksibilitas yang lebih tinggi walaupun faktanya secara implementasi BigchainDB lebih kompleks. Namun kedua *database* ini merupakan *production ready* dan siap untuk digunakan dalam penggunaan *throughput* yang besar.

BigchainDB menggunakan salah satu teknologi yang paling aman yaitu dengan menambahkan fitur *blockchain*. BigchainDB menjanjikan bahwa memiliki *high throughput, low latency, immutable data*, dan *Open-source* dimana sudah *support* untuk beberapa bahasa pemrograman seperti Java, Python, dan JavaScript. Terlebih walaupun menggunakan fitur *blockchain*, BigchainDB dapat dijalankan hanya menggunakan 1 *node* saja. BigchainDB menggunakan konsensus Tendermint yang memiliki waktu konsesus yang cepat dan efisien dalam penggunaan sumber daya komputasi.

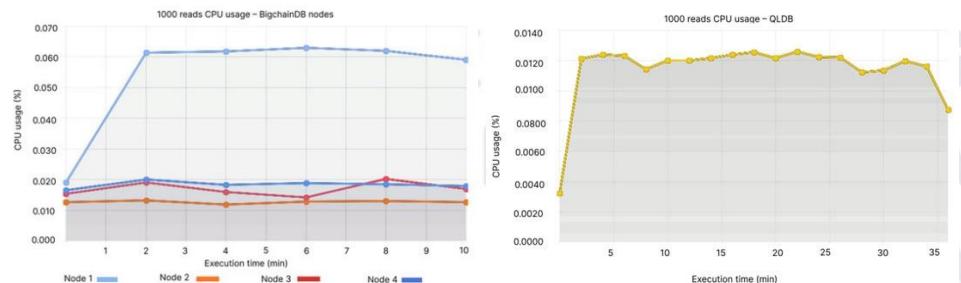
Disisi lain, Amazon QLDB menggunakan fitur *cryptography* agar datanya memiliki kemampuan *immutability*. namun sistemnya masih menggunakan arsitektur *centralize*. Dan Amazon QLDB tidak *open-source*.

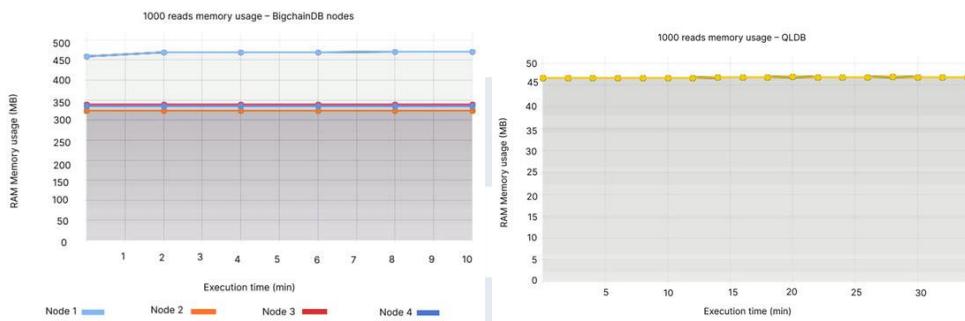
Pada penelitian ini ditemukan bahwa dengan BigchainDB dengan melakukan *input* 1000 data ke dalam *database*, ditampilkan penggunaan CPU dan *memory* pada gambar 2.9.



Gambar 2.9 Perbandingan *input* 1000 data dengan BigchainDB dan Amazon QLDB [14]

Dan penggunaan CPU dan *memory* untuk membaca data sebanyak 1000 data, ditampilkan pada gambar 2.10.





Gambar 2.10 Perbandingan membaca 1000 data dengan BigchainDB dan Amazon QLDB [14]

Dari hasil grafik penelitian, disimpulkan bahwa jika dibandingkan antara BigChainDB dan Amazon QLDB dengan matriks pengujian CPU *usage* kedua *database* memiliki rata-rata penggunaan CPU yang sama. Namun untuk penggunaan *memory*, BigchainDB lebih besar dari pada Amazon QLDB. Dan jika dianalisis lebih lanjut, dalam 1 menit BigchainDB bisa menghasilkan 50 *insert* data ke seluruh 4 node dan 100 *read* data ke dalam *blockchain*. Amazon QLDB memiliki *throughput* yang lebih kecil karena harus melakukan proses *cryptography* yang akan memakan sumber daya komputasi.

Beberapa poin penting yang dapat diambil oleh penulis adalah sebagai berikut :

1. BigchainDB adalah *database* terdesentralisasi yang memiliki fitur *blockchain* sehingga memiliki kemampuan *immutability*.
2. BigchainDB memiliki *throughput* setidaknya dalam 1 menit berhasil melakukan 50 *insert* data dan 100 *read* data
3. BigchainDB relatif memiliki CPU *usage* yang sedang namun memiliki *memory usage* yang cukup tinggi
4. BigchainDB menggunakan Tendermint untuk sistem konsensus yang cepat dan hemat dalam penggunaan sumber daya komputasi jika dibandingkan dengan konsensus POS atau POW Etherium.

5. BigchainDB walaupun implementasinya cukup rumit namun memiliki fleksibilitas yang lebih dan bisa dijalankan pada 1 *node* saja
6. BigchainDB merupakan *database* yang *open-source*.
7. Dapat diimplementasi menggunakan *database* BigchainDB untuk membagi data informasi penyerang pada proxy terdistribusi.

2.1.6 *A DDoS Attack Information Fusion Method Based on CNN for Multi-Element Data [15]*

Penelitian dengan judul “*A DDoS Attack Information Fusion Method Based on CNN for Multi-Element Data*” yang dilakukan oleh Jieren Cheng, Canting Cai, Xiangyan Tang, Victor S. Sheng, Wei Gui, dan Mengyang Li memiliki tujuan untuk melakukan efisiensi proses komputasi, penggunaan *memory*, *running time*, dan meningkatnya akurasi dengan meneliti *features* yang berbeda dari dataset serangan DDoS. *Features* yang digunakan adalah *Source IP Address*, *Destination IP Address*, *source port*, *destination port*, *packet size*, dan *number of IP address*.

Penelitian ini juga dijabarkan analisis dari karakteristik serangan DDoS, yaitu sebagai berikut :

1. *Distribution*

Ketika serangan DDoS dilakukan tujuan utama adalah menghabiskan *resource* dari perangkat target. Oleh karena itu penyerang akan memakai *IP address* palsu, acak, dan banyak, dan *source port* yang acak dan banyak

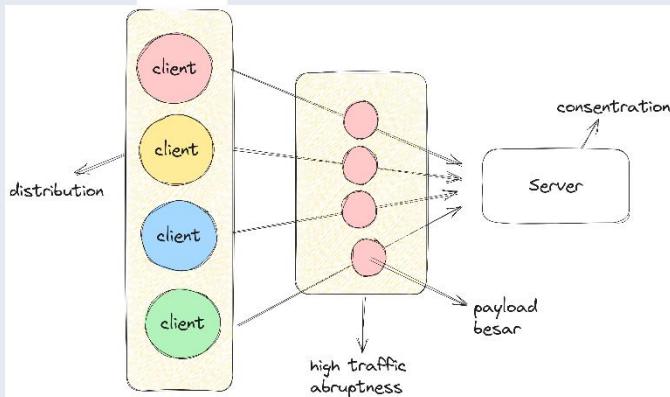
2. *Concentration*

Ketika serangan DDOS dilakukan, penyerang ingin secara spesifik menyerang perangkat target dan aplikasinya. Oleh karena itu destinasi *IP Address* dan *port* pasti terkonsentrasi hanya 1 dan menggunakan *packet size* yang sama semua

3. *High traffic abruptness*

Penyerang akan menggunakan *payload request* dengan ukuran yang besar, oleh karena itu saat terjadi serangan jumlah *number of packet* akan semakin banyak juga.

Untuk ilustrasinya karakteristik *distribution*, *concentration*, dan *High traffic abruptness* dapat dilihat pada gambar 2.11.



Gambar 2.11 Ilustrasi Karakteristik Serangan DDoS

Penelitian ini memperkenalkan MEFF *features* menggunakan teori statistika union untuk mendapatkannya, uraian MEFF *features* adalah sebagai berikut:

1. SIPAF : merupakan kepanjangan dari *Source IP Address Feature*, dimana mengalkulasi variasi dari *IP address* sumber yang berbeda. Disimpulkan bahwa pada lalu lintas normal, jumlah alamat IP sumber yang berbeda dalam aliran jaringan harus lebih sedikit dan stabil dalam satu periode waktu dalam keadaan normal.

$$\text{Rumus : } \text{SIPAF} = |\cup_{i=1}^n \{\text{source ip } i\}|$$

2. DIPAF : merupakan kepanjangan dari *Destination IP Address Feature*, dimana mengalkulasikan variansi dari *IP address* tujuan yang berbeda. Disimpulkan bahwa pada lalu lintas yang normal, *IP address* tujuan yang berbeda lebih banyak dari biasanya, dan jika pada serangan maka *IP address* tujuan akan lebih sedikit dari biasanya.

Rumus : $DIPAF = |\cup_{i=1}^n \{destination ip i\}|$

3. SPF : merupakan kepanjangan dari *Source Port Feature*, dimana mengalkulasikan variansi dari *port* sumber yang berbeda. Disimpulkan bahwa pada lalu lintas yang normal, *port* sumber yang berbeda lebih sedikit dan stabil dalam satu periode.

Rumus : $SPF = |\cup_{i=1}^n \{Source port i\}|$

4. DPF : merupakan kepanjangan dari *Destination Port Feature*, dimana mengalkulasikan variansi dari *port* tujuan yang berbeda. Disimpulkan bahwa pada lalu lintas yang normal, *port* destinasi yang berbeda lebih sedikit dan stabil dalam satu periode.

Rumus : $DPF = |\cup_{i=1}^n \{Destination port i\}|$

5. PNF : merupakan kepanjangan dari *Packet Number Feature*, disimpulkan bahwa jumlah paket yang dikirimkan pada lalu lintas normal akan lebih sedikit jika dibandingkan saat terjadi serangan DDoS.

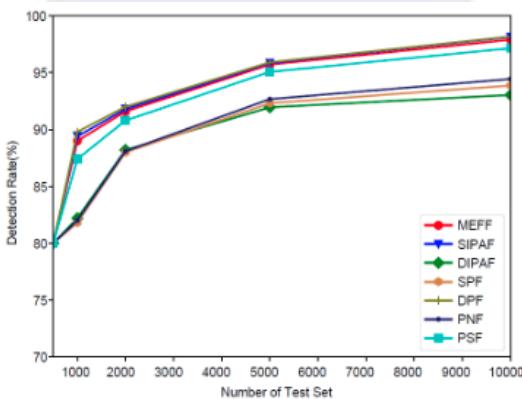
Rumus : $PNF = n$

6. PSF : merupakan kepanjangan dari *Packet Source Features*, disimpulkan bahwa besar paket pada lalu lintas yang normal akan berbeda beda saat misalnya mengunjungi website yang berbeda-beda. namun pada kondisi serangan DDoS, besar paket akan sama semua.

Rumus : $PSF = |\cup_{i=1}^n \{Packet Size i\}|$

Pada penelitian ini diteliti bahwa penggunaan 6 features yang disebut dengan MEFF atau *Multi-element Fusion Feature* dimana merupakan dataset pendekatan individual dari dataset timeseries dalam rentang satuan waktu tertentu. Penelitian ini menunjukan untuk algoritma *machine learning* SVM dengan kernel radian basis function, dibandingkan *features* MEFF dengan pilihan *features* yang lain menghasilkan *running time* untuk prediksi 1000 paket dengan *features* MEFF hanya 9,6 detik

dengan *error rate* 1,25% sementara untuk *features* lainnya yaitu 101.44 detik. Grafik hasil penelitian dapat dilihat pada gambar 2.12.



Gambar 2.12 Hasil Pengujian dan Analisis penggunaan *features* MEFF [15]

Beberapa poin penting yang dapat diambil oleh penulis adalah sebagai berikut:

1. Karakteristik dari serangan DDoS memiliki 3 fokus yaitu *distribution*, *consentration*, dan *high traffic abruptness*.
2. Pemilihan dan penggunaan *features* yang berbeda sangat mempengaruhi hasil akurasi, waktu prediksi, dan penggunaan *memory* dari model *machine learning*.
3. Dapat digunakan 6 *features* yaitu SIPAF, DIPAF, SPF, DPF, PNF, dan PSF. Ke semua *features* yang dipakai berdasarkan statistika union dan variasi.
4. Penggunaan *machine learning* dengan metode “individual” dengan algoritma CNN dan SVM dengan kernel *radian basis function*
5. Semakin tinggi penggunaan jumlah *test set* pada dataset semakin bagus akurasi yang didapat.

2.1.7 *Performance Evaluation of Multi-Core, Multi-Threaded SIP Proxy Servers* [16]

Penelitian dengan judul “*Performance Evaluation of Multi-Core, Multi-Threaded SIP Proxy Servers*” yang dilakukan oleh Ramesh

Krishnamurthy dan George N. Rouskas memiliki tujuan untuk menemukan korelasi antara penggunaan jumlah *cores* dan *thread* processor yang berbeda antara performa proxy untuk menjadi perantara jaringan komputer. Digunakan *software open-source* proxy yaitu OpenSIPS dan digunakan fitur *load balancer*-nya. Penelitian ini dilakukan pada *operating sistem* :inux dan mengatur untuk CFS yaitu *scheduling policy* untuk menciptakan *multicore environment*. Karena *operating sistem* memiliki sebuah *pool core* dan *scheduler* untuk membagi *cores* ke dalam proses yang membutuhkan *processing power* lebih tinggi. Karena sekarang ini processor kebanyakan memiliki jumlah *cores* dan *thread* yang banyak. Ditemukan bahwa dengan menggunakan jumlah *cores* yang lebih banyak akan meningkatkan performa dari proxy server.

Penelitian ini diteliti bahwa setiap jumlah *cores* dan *thread* memiliki jumlah *call per second* (CPS) yang bermacam macam. Pada umumnya jika jumlah cores dinaikkan maka nilai CPS juga akan ikut naik, sehingga akan bagus untuk *concurrent connection*. Begitu juga untuk metrik pengujian PDR atau *packet drop rate*, Jika dinaikkan jumlah *cores* dan *thread* maka nilai PDR akan turun yang menandakan peningkatan performa. Namun perlu diperhatikan juga konfigurasi *scheduler* karena akan ada *migration cost*. Migration cost adalah proses *overhead* yang tidak efisien ketika cores akan berpindah ke proses lain secara sementara akibat kurangnya *cores* untuk proses selain proxy server. Oleh karena itu perlu diatur untuk jumlah *cores* yang khusus mengani proxy server. Untuk hasil penelitian dapat dilihat pada gambar 2.13.

TABLE II
MEASURED SPS PERFORMANCE, ENHANCED MULTI-CORE SERVER MODE, SPS ON 2-CORE

Model Parameters	Number of Server Threads									
	2 Threads		4 Threads		6 Threads		8 Threads			
	3800cps	4000cps	4400cps	4600cps	3800cps	4000cps	3200cps	3400cps	2600cps	2800cps
Arrival rate (packets/sec)	22800	24000	26400	27600	22800	24000	19200	20400	15600	16800
T_{avg} (μs)	54.41	53.55	83.62	85.34	101.28	116.15	125.78	139.26	218.07	262.32
K_{avg} (μs)	241.28	283.81	469.17	591.99	348.17	569.58	358.59	492.15	398.73	622.95
RCV Errors	3087	8017	3119	4245	3348	5366	3798	5586	3986	6830
Call-setup Drops	2682	7002	2736	3734	2879	4646	3205	4781	3228	5624
Call-setup Messages	320141	635594	317922	315676	321645	316459	324962	318851	334935	326550
Total Messages	368496	727720	362463	358886	373972	365455	385030	372492	413491	396564
PDR	0.0084	0.011	0.0086	0.0118	0.0089	0.0147	0.0098	0.0149	0.0096	0.0172

Gambar 2.13 Hasil Pengujian dan Analisis Penggunaan Multicore dan Multithread pada Proxy Server [16]

Beberapa poin penting yang dapat diambil oleh penulis dari penelitian ini adalah :

1. Membuat proxy dengan arsitektur *multicore* dan *multithreading* akan meningkatkan performa proxy namun tetap memperhatikan *migration cost* untuk konsekuensi perpindahan *thread* antar proses.
2. Semakin banyak *cores* dan *thread* yang digunakan maka akan semakin bagus performa proxynya terutama dalam menangani masalah *concurrent connection*, namun tetap harus dikonfigurasi secara baik.

Berdasarkan ke 7 penelitian terdahulu yang diambil penulis sebagai bahan referensi perancangan dan pembuatan sistem yang akan dibangun, berikut merupakan rangkuman poin yang diambil dan dijadikan acuan penelitian oleh penulis :

1. Pengetahuan tentang definisi dan cara kerja serangan UDP flood, HTTP flood, ICMP flood, TCP SYN flood, TCP PUSH + ACK flood, dan *Ping of Death flood*. Sehingga dapat dijadikan acuan untuk mempertimbangkan *features* yang digunakan dan pembuatan program simulasi serangan DDoS.
2. Karakteristik dari serangan DDoS memiliki 3 fokus yaitu, *distribution*, *consentration*, dan *high traffic abruptness*. Informasi ini menjadi acuan saat menentukan *features* yang akan dipakai.
3. Dapat diimplementasi teknik honeypots, firewall, dan *Reactive & Cooperative* yang dijabarkan pada penelitian terdahulu nomor 2.1.1. Yaitu teknik melakukan *logging*, *attack filtering*, dan pembagian informasi identitas penyerang.
4. Diteliti pendekatan metode *machine learning* secara individual dengan menemukan *features* dari rangkuman paket dengan rentang waktu 5 detik dan teknik kalkulasi statistika standar deviasi, union, dan variasi.
5. Menggunakan algoritma *machine learning* SVM dengan kernel linear untuk mengatasi masalah *high-dimensional mapping* dan menghemat proses

komputasi. Selain itu dapat juga dipakai SVM dengan kernel *radian basis function*.

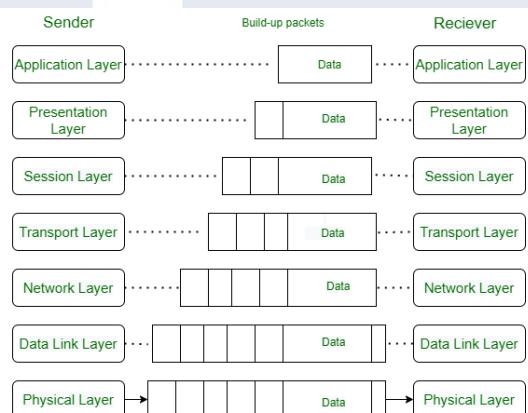
6. Menggunakan *control plane* SDN sebagai *listener* dan *logger* paket yang berlalu.
7. Diteliti pendekatan metode *machine learning* secara *timeseries* menggunakan algoritma *deep learning* LSTM. Digunakan LSTM karena bagus dalam mendeteksi secara *realtime* dan memiliki akurasi yang bagus karena dapat menyimpan informasi atau *features* dari waktu yang lama ke belakang.
8. Penelitian terdahulu nomor 2.1.3 menjadi acuan saat melakukan proses transformasi dataset menjadi 3 *dimensional matrix* yaitu *batch size*, *time step*, dan *input dimension*. Dan acuan penggunaan *time window* yang memiliki korelasi terhadap akurasi model LSTM.
9. Menggunakan layer *output* pada *deep learning* berupa Sigmoid yang bagus untuk *binary classification*. Yaitu kelas label dataset berupa 0 untuk normal dan 1 untuk serang.
10. Menggunakan perbandingan dataset *training* dengan dataset *testing* bernilai 80:20
11. Pemilihan dan penggunaan *features* yang berbeda akan sangat berpengaruh pada hasil akurasi, waktu prediksi, dan penggunaan *memory* dari model *machine learning* yang digunakan.
12. Mengimplementasi arsitektur dan alur kerja modul backend, *blockchain*, dan *firewall* yang serupa dengan penelitian terdahulu nomor 2.1.4.
13. Informasi identitas penyerang yang disebarluaskan ke *blockchain* berupa *IP Address* penyerang.
14. Dapat digunakan *database* terdesentralisasi yaitu BigchainDB yang memiliki banyak keunggulan. Salah satu fokusnya adalah kemampuan *immutability*-nya namun masih memiliki *throughput* besar dan *latency* yang kecil. Selain itu memiliki fleksibilitas yang tinggi untuk diimplementasikan pada sistem yang akan dibangun.

15. Saat merancang dan membangun modul proxy perlu diperhatikan penggunaan arsitektur *multicores* dan *multithreading* agar meningkatkan performa proxy terutama pada *concurrent connection*.

2.2 Tinjauan Teori

2.2.1 OSI Layer [17]

OSI atau *Open System Interconnection* adalah sebuah model untuk mereferensikan hubungan antar berbagai protokol jaringan komputer. Model OSI terdiri dari tujuh layer yang saling berhubungan. OSI mendefinisikan kerangka kerja jaringan komputer untuk memahami dan merancang protokol jaringan. Model OSI digunakan agar tidak ada keterkaitan terhadap suatu teknologi atau protokol tertentu dalam jaringan komputer seperti *software*, *operating system*, dan lain lain. Setiap lapisan memiliki fungsi dan ketergantungan khusus dan bertanggung jawab untuk menyediakan layanan kepada layer di atasnya dan menggunakan layanan dari layer di bawahnya. Untuk ilustrasinya ditampilkan pada gambar 2.14 :



Gambar 2.14 Ilustrasi Ketergantungan OSI Layer

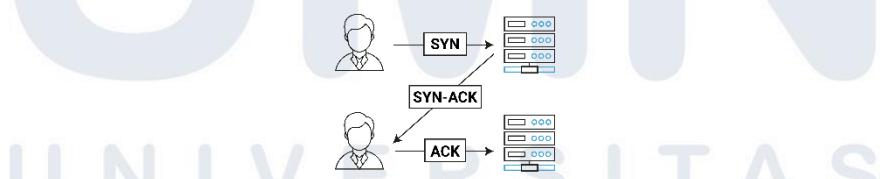
Ketujuh lapisan OSI layer beserta fungsinya adalah :

1. *Physical Layer* : digunakan untuk transmisi data sinyal fisik dan menggunakan bit sebagai obyek komunikasinya.
2. *Data Link Layer* : digunakan sebagai wadah bit-bit data yang dikirimkan oleh *physical layer*. Digunakan juga sebagai pendekripsi kesalahan pada *physical layer*.

3. *Network Layer* : digunakan sebagai pengatur rute data di dalam jaringan komputer yang kompleks, misal terdapat *mac address* sebagai identifikasi unik setiap perangkat yang berkomunikasi di jaringan komputer.
4. *Transport Layer* : digunakan sebagai mekanisme pengiriman data *end-to-end* yang andal dan memastikan data dikirimkan dan diterima secara benar
5. *Session Layer* : digunakan untuk mengatur dan menjaga dialog antar aplikasi di kedua ujung komunikasi sehingga tidak terjadi tabrakan
6. *Presentation Layer* : digunakan untuk mengubah format data agar dapat dipahami oleh aplikasi penerima. Yaitu yang sebelumnya berupa bit-bit data, diubah menjadi sesuatu data dengan format tertentu seperti html, jpg, dan lain lain.
7. *Application layer* : digunakan untuk menyediakan akses ke aplikasi jaringan bagi aplikasi pengguna akhir. Seperti protokol FTP, HTTP, SMTP, dan lain lain.

2.2.1.1 TCP [18] [19]

TCP atau *Transmission Control Protocol* adalah salah satu protokol pada layer 4 (*transport layer*) yang digunakan dalam komunikasi jaringan komputer. TCP menjanjikan untuk menjadi protokol yang andal untuk mengirim data yang terjamin, pemulihan data yang hilang atau rusak, pengaturan lalu lintas untuk efisiensi komunikasi. Karena TCP menggunakan mekanisme *handshaking* untuk mengawali dan mengakhiri komunikasi 2 arah. Ilustrasi dari TCP *handshaking* dapat dilihat pada gambar 2.15.

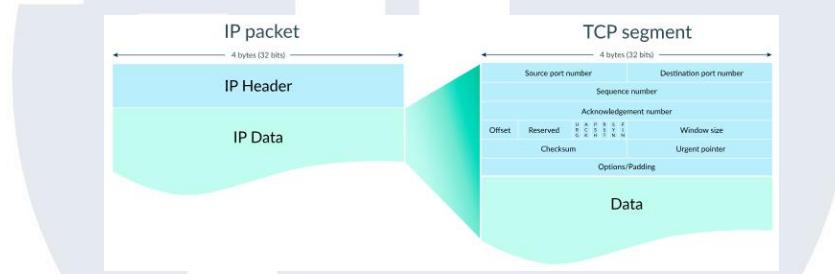


Gambar 2.15 Alur handshaking TCP [19]

Jadi pertama kali *client* akan mengirimkan TCP SYN lalu server akan menyimpan *IP address client* ke tabel antrian. Saat giliran *client* tersebut di layani oleh server, server akan merespons dengan TCP SYN-ACK, dan *client* harus

menjawab TCP ACK untuk memastikan data yang diterima tidak ada yang hilang atau rusak. TCP juga mengatur urutan pengiriman data dengan membuat tabel antrean sehingga dapat mengontrol laju transfer data dan melakukan pemulihan data yang hilang dan rusak. Oleh karena itu jika *client* tidak merespons dengan TCP ACK maka server akan menunggu *client* sampai rentang waktu tertentu.

Secara arsitektur protokol TCP terdiri dari 2 segmen yaitu *header* dan *payload* yang dapat dilihat pada gambar 2.16.



Gambar 2.16 TCP Segmen [19]

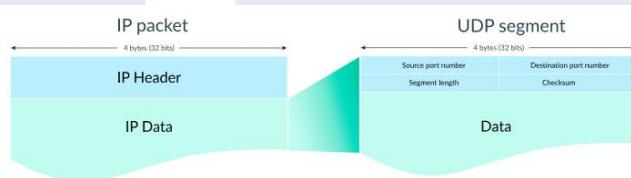
Untuk melakukan *handshake*, diperlukan beberapa data yang disimpan pada *IP Header*.

1. Port sumber dan port tujuan: digunakan untuk mengidentifikasi aplikasi yang berkomunikasi.
2. *Sequence number*: digunakan untuk mengatur urutan pengiriman data dan mengidentifikasi segmen yang dikirimkan.
3. *Acknowledgment number*: berisi nomor urutan terakhir yang telah diterima dengan benar oleh penerima. Digunakan untuk mengonfirmasi penerimaan data dan mengatur mekanisme pengiriman ulang data jika ada data yang hilang.
4. Panjang *Header*: Menunjukkan panjang *header* segmen TCP sehingga pembagian antara segmen *header* dan *payload* jelas.
5. *Flag Kontrol*: berisi bit yang mengidentifikasi perilaku TCP seperti SYN, ACK, dan FIN. SYN (*synchronization*) untuk memulai koneksi, ACK (*acknowledgment*) untuk mengkonfirmasi penerimaan data, dan FIN (*finish*) untuk mengakhiri koneksi.

6. *Window Size*: digunakan untuk memberitahu jumlah data yang dapat diterima oleh penerima.
7. Data: berupa sejumlah data dari aplikasi yang menggunakan TCP.

2.2.1.2 UDP [20]

UDP atau *User Datagram Protocol* adalah salah satu protokol pada layer 4 (*transport layer*) yang digunakan dalam komunikasi jaringan komputer. Berbeda dengan TCP yang berorientasi komunikasi yang andal dan terurut sehingga tidak ada data yang hilang atau rusak, namun UDP menawarkan komunikasi yang cepat dan efisien. UDP dapat dimanfaatkan untuk pengiriman data yang cepat dan *realtime* seperti *streaming media*, VoIP, dan *game online* yang sensitif terhadap besarnya *latency*. Karena UDP tidak memakai mekanisme *handshake* seperti TCP, maka tidak dibutuhkan beberapa informasi khusus untuk komunikasi UDP. Arsitektur komunikasi UDP dapat dilihat pada gambar 2.17.



Gambar 2.17 UDP Segmen [20]

Penjabaran tentang isi dari *header* dari komunikasi UDP adalah sebagai berikut

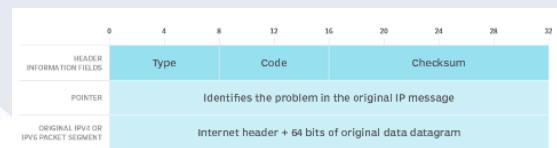
:

1. Port sumber dan port tujuan: digunakan untuk mengidentifikasi aplikasi yang berkomunikasi.
2. Panjang Datagram: digunakan untuk mengidentifikasi panjang keseluruhan datagram UDP, termasuk *header* dan *payload*.
3. *Checksum*: digunakan untuk mendeteksi kesalahan dalam datagram UDP selama pengiriman dengan nilai *checksum* tertentu. Namun tidak ada mekanisme pengiriman ulang pada protokol ini, sehingga perlu diimplementasikan secara terpisah pada aplikasi yang menggunakan UDP.

2.2.1.3 ICMP [21]

ICMP atau *Internet Control Message Protocol* adalah salah satu protokol pada layer 3 (*Internet Protocol / Network layer*) yang digunakan dalam komunikasi jaringan komputer. Pesan ICMP dikemas dalam *payload* datagram IP dan menggunakan identifikasi nilai tipe dan kode untuk menentukan jenis dan tujuan pesan yang dikirimkan. ICMP berperan penting dalam pengoperasian dan pemecahan masalah jaringan komputer dengan memberikan informasi tentang kondisi dalam jaringan komputer seperti *traceroute* untuk mengetahui jalur yang digunakan dalam komunikasi antar komputer dan *ping* untuk menandakan suatu perangkat dapat dicapai oleh perangkat lain.

Secara umum arsitektur dari komunikasi ICMP diilustrasikan pada gambar 2.18.



Gambar 2.18 ICMP Segmen [21]

Pada protokol ICMP terdapat 3 informasi penting yang disimpan dalam *header* datagramnya. Yaitu :

1. *Type* : berupa angka untuk menentukan tipe pesan yang dikirimkan. Contohnya adalah 0 untuk *echo reply*, 3 untuk *destination unreachable*, 8 *echo*, dan 5 untuk *redirect* [22].
2. *Code* : berupa angka untuk mengidentifikasi lebih lanjut paket apa berdasarkan tipe pada *header* sebelumnya.
3. *Checksum* : digunakan untuk mengecek apakah paket yang diterima ada kerusakan atau tidak.

2.2.1.4 HTTP [23]

HTTP atau *Hypertext Transfer Protocol* adalah salah satu protokol pada layer 7 (*application layer*). HTTP menjadi dasar dari komunikasi web dan digunakan secara luas untuk mengakses halaman web dari aplikasi browser dan web

server, mengirim formulir, mengunduh file, berinteraksi dengan API, dan banyak lagi. HTTP memiliki banyak tipe metode permintaan seperti GET, POST, PUT, DELETE, dan lain lain yang memiliki tujuannya masing-masing. HTTP juga memiliki status kode untuk informasi keberhasilan atau kegagalan permintaan (bukan status komunikasi).

Dalam komunikasinya HTTP menggunakan protokol layer 4 TCP untuk mengirimkan data yang andal. Setelah koneksi TCP terbentuk, data HTTP dikemas dalam segmen TCP dan dikirim melalui jaringan komputer. Dalam 1 *request* HTTP bisa saja berisi beberapa segmen TCP sesuai besar data HTTP yang dikirimkan oleh TCP. Oleh karena itu untuk menggunakan protokol HTTP harus terlebih dahulu menggunakan protokol TCP sesuai dengan alurnya.

2.2.2 IP Tables

IP tables adalah *software* pada sistem operasi Linux yang digunakan untuk mengontrol lalu lintas jaringan yang masuk, keluar, dan melewati sistem. Dengan IP tables kita dapat mengatur sebuah *firewall rule* yang spesifik untuk mengizinkan atau memblokir paket data berdasarkan berbagai kriteria seperti *IP address* sumber dan tujuan, port, protokol, dan lain lain. IP tables berjalan pada tingkat *kernel* di sistem operasi Linux, sehingga akan sangat efisien untuk memproses lalu lintas jaringan di sistem tersebut. IP tables dapat digunakan untuk mengamankan jaringan dengan mengatur akses ke layanan tertentu, melindungi sistem dari serangan jaringan, mencegah lalu lintas yang tidak diinginkan, dan mengatur kebijakan keamanan jaringan. IP tables dapat diintegrasikan ke berbagai macam *software* pihak ketiga untuk mengatur keamanan jaringan pada sistem tersebut. IP tables akan digunakan oleh penulis sebagai dasar modul *firewall controller*.

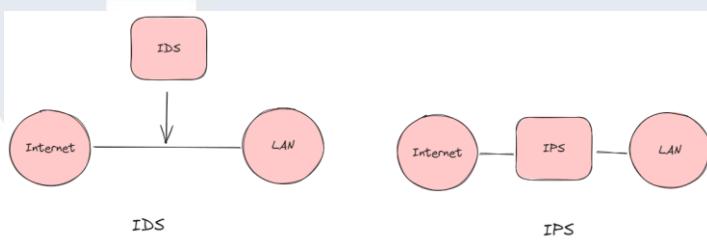
2.2.3 Intrusion Prevention System dan Intrusion Detection System [24]

IPS (*Intrusion Prevention System*) dan IDS (*Intrusion Detection System*) adalah *tools* yang penting dalam keamanan jaringan komputer khususnya serangan DDoS yang berfungsi untuk mendekksi dan mencegah serangan terhadap sistem dan jaringan. Keduanya memiliki kemiripan dan perbedaan yang spesifik.

IDS adalah sistem yang dirancang untuk mendeteksi suatu anomali atau pola yang mencurigakan di dalam jaringan atau sistem. IDS akan mendapatkan data paket yang berlalu di sistem, lalu mencocokkan pola serangan yang diketahui dengan data jaringan yang diamati, dan menghasilkan peringatan atau laporan jika ada indikasi serangan.

Di sisi lain, IPS adalah evolusi dari IDS yang tidak hanya mendeteksi serangan, tetapi juga mengambil tindakan langsung terhadap serangan tersebut. Ini bisa berupa memblokir alamat IP penyerang, menutup port yang diserang, mengubah kebijakan *firewall*, dan lain lain.

Pada gambar 2.19 merupakan ilustrasi penggunaan IDS dan IPS :



Gambar 2.19 Implementasi IDS dan IPS

Untuk metode deteksi yang digunakan ada bermacam macam metode tergantung vendor yang membuat IPS atau IDS tersebut. Namun secara dasar ada 3 metode yaitu :

1. Deteksi berbasis *signature*

Dengan deteksi berbasis *signature*, IPS tidak akan bisa mendeteksi suatu serangan yang baru. Hal ini karena IPS akan mencocokkan *traffic* yang ada dengan objek *signature* yang disimpan dalam *database*. *Signature* yang disimpan bisa berbagai macam bentuk seperti *ip address*, port, dan berbagai macam.

2. Deteksi berbasis anomali (*behavior*)

Deteksi berbasis anomali adalah dengan membandingkan keadaan normal dengan keadaan terserang DDoS. Dalam hal ini misal terdiri dari unit threshold

untuk menentukan batasan jumlah paket yang diterima dalam satuan dan lain lain.

Biasanya vendor akan menggabungkan beberapa macam metode untuk mendapatkan akurasi yang tinggi.

2.2.3.1 Snort

Snort adalah *software* IDS dan IPS yang dikembangkan oleh Sourcefire namun kini dialih kembangkan oleh Cisco Talos. Snort merupakan *software open-source* yang dapat secara gratis diinstall di sistem dengan OS Linux atau Windows. Snort dirancang untuk memantau lalu lintas jaringan dan mendeteksi aktivitas yang mencurigakan atau berbahaya. Snort merupakan IDS dan IPS yang popular dan memiliki ulasan penggunaan yang sangat bagus dan sudah banyak untuk digunakan sebagai objek penelitian [25].

Snort akan menganalisis paket-paket jaringan komputer secara *realtime* dan membandingkannya dengan sekumpulan aturan yang telah ditentukan sebelumnya. Aturan tersebut bisa dikonfigurasi untuk ditambahkan atau dinonaktifkan secara manual oleh usernya. Pada tabel 2.2 merupakan tabel Snort *rule syntax* yang dapat dijadikan acuan untuk membuat snort rule secara mandiri.

Tabel 2.2 Tabel Snort Rule Syntax

Action	Protocol	Source Address	Source Port	Direction	Dest Address	Dest Port	Rule option
Alert	TCP			<			Msg
Log	UDP			>			Ttl
Pass	ICMP			<>			Seq
Drop							Ack
Reject							Dsize
							Icmp_seq
							dll

Jika dilihat pada tabel Snort *rule syntax* diatas, Snort memiliki fleksibilitas yang sangat tinggi. Kita bisa secara terpisah melakukan protokol apa yang akan kita

monitor, alamat jaringan tertentu, arah komunikasi, dan melakukan tindakan tertentu. Kita juga diberi pilihan untuk menggunakan secara gratis dan diberi kumpulan Snort *rule* dari komunitas atau membayar sebesar \$29.99 per tahunnya untuk mendapatkan pembaharuan Snort *rule* langsung dari Cisco Talos (kelompok yang ahli dalam bidang *cybersecurity*).

Snort dapat diterapkan sebagai sensor mandiri atau sebagai bagian dari infrastruktur keamanan jaringan yang lebih besar.

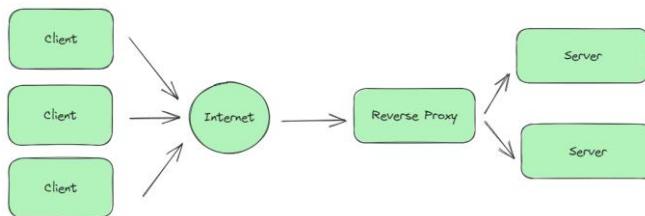
2.2.4 NetfilterQueue

NetfilterQueue adalah suatu library untuk bahasa pemrograman Python, sehingga dimungkinkan untuk berinteraksi langsung dengan IP tables yang sudah dijelaskan penulis diatas. Dapat dilakukan *packet filtering*, *packet mangling*, dan lain lain. Dengan ini dimungkinkan untuk dibuat suatu automasi dengan membuat suatu *callback functions* tertentu untuk menganalisis, memodifikasi, atau bahkan memblokir paket jaringan secara dinamis. Jadi fungsi tersebut dapat menangkap paket yang melewati sistem (dari IP tables), memeriksa dan mengubah data paket, lalu memutuskan apakah paket tersebut akan diterima (*accept*) atau dibuang (*drop*).

Pemanfaatan NetfilterQueue sangat luas dalam bidang keamanan jaringan dan analisis jaringan. Contoh penggunaan NetfilterQueue termasuk pembuatan IPS atau IDS, program sebagai *listener* dan *logging* paket, pembuatan *software firewall*, dan lain lain. Oleh karena itu penulis memutuskan untuk menggunakan NetfilterQueue sebagai *firewall controller* untuk menentukan suatu paket diterima atau ditolak dan menjadi *listener* atau *logger* yang nantinya akan diproses oleh *machine learning*.

2.2.5 Reverse Proxy [26]

Reverse proxy adalah sebuah server yang bertindak sebagai perantara antara *client* dan server tujuan. Ketika *client* mengirimkan *request* ke server, *request* tersebut akan dikirimkan terlebih dahulu ke reverse proxy yang kemudian meneruskannya ke server tujuan. Server tujuan kemudian mengirimkan responsnya kepada reverse proxy, dan respons tersebut diteruskan kembali kepada client. Untuk ilustrasi implementasi dari reverse proxy dapat dilihat pada gambar 2.20.



Gambar 2.20 Implementasi Reverse Proxy

Reverse proxy dapat bertindak sebagai lapisan pertahanan antara *client* dan server tujuan. Sehingga dimungkinkan diimplementasikan pengaturan kebijakan keamanan, termasuk penyaringan paket berdasarkan aturan tertentu, deteksi serangan, dan perlindungan terhadap DDoS. Selain itu menyembunyikan informasi tentang server dari *public internet* sehingga mengurangi risiko serangan langsung ke server.

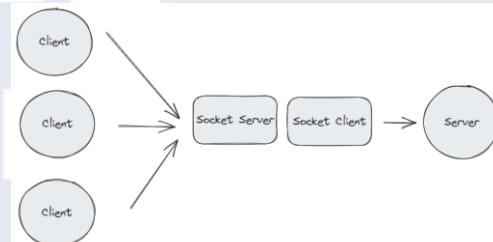
2.2.5.1 Concurrent Connection [27]

Concurrent connection adalah suatu kondisi saat beberapa koneksi atau sesi secara bersamaan terjadi. Dalam konteks jaringan komputer, ini mengacu pada jumlah koneksi yang dapat dilayani oleh server atau perangkat jaringan dalam satu waktu tertentu. Ketika 1 pengguna terhubung ke suatu server dianggap sebagai satu koneksi terjadi. Jika server atau perangkat jaringan memiliki kemampuan untuk menangani banyak koneksi secara bersamaan, maka dikatakan memiliki "*concurrent connection*" yang tinggi. Misalnya, jika sebuah server memiliki kemampuan untuk menangani 100 *concurrent connection*, itu berarti server tersebut dapat melayani hingga 100 *client* yang terhubung ke server secara bersamaan tanpa mengalami penurunan kinerja. Kemampuan ini perlu dipunyai oleh *reverse proxy* karena suatu server pasti akan menangani banyak sekali *client* dan dimungkinkan secara bersamaan. Jumlah koneksi bersamaan yang dapat ditangani oleh suatu sistem tergantung pada berbagai faktor, termasuk sumber daya perangkat keras (seperti CPU, memori, dan *bandwidth* jaringan) dan bagaimana arsitektur perangkat lunak atau proxy tersebut dibuat.

2.2.6 Python Socket

Python socket adalah salah satu *library* pada bahasa pemrograman Python yang menyediakan *interface* untuk komunikasi jaringan komputer secara *low level*.

Library ini dapat digunakan untuk membuat program yang dapat berkomunikasi melalui protokol TCP atau UDP. Dengan menggunakan *library* ini, kita dapat membuat program *client* dan server untuk berkomunikasi melalui jaringan komputer. Klien untuk mengirim permintaan ke server, dan server dapat merespons permintaan tersebut. *Library* ini juga menyediakan banyak fungsi dan pengaturan koneksi jaringan seperti nilai *timeout*, pengolahan *IP address* dan port, serta *custom callback error* yang terkait dengan komunikasi jaringan. Hal ini lah yang dibutuhkan untuk membuat suatu program *reverse proxy* dengan bahasa pemrograman Python. Pada gambar 2.21 ditampilkan contoh implementasinya.

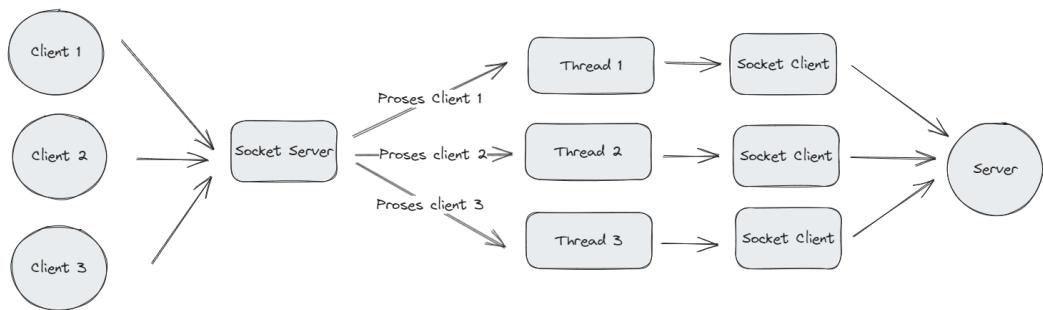


Gambar 2.21 Contoh Implementasi Socket untuk Reverse Proxy

2.2.7 Python Multithreading

Untuk membuat Socket Python dapat berkerja secara parallel untuk mengatasi masalah *concurrent connection* maka perlu membuat arsitektur program menjadi *multithreading*. Bahasa pemrograman Python sendiri sudah mendukung untuk menjalankan instruksi secara *multithreading* dimana dapat digunakan library bernama “*threading*”. *Multithreading* digunakan untuk melakukan instruksi secara paralel dalam satu proses Python. *Multiple thread* pada processor akan berjalan di dalam satu proses yang sama, menggunakan *resource* yang sama (seperti memori). Hal ini akan menyebabkan kondisi perlombaan (*race conditions*) jika tidak ditangani dengan benar.

Untuk keperluan *reverse proxy* maka diperlukan metode *multithreading*, dimana membagi *object* Socket menjadi beberapa *thread handler*. Untuk gambaran implementasinya ditampilkan pada gambar 2.22.



Gambar 2.22 Contoh Implementasi Multithreading Socket untuk Reverse Proxy

2.2.8 BigchainDB

BigchainDB adalah software *database* terdesentralisasi yang memiliki fitur blockchain di dalamnya dan pertama kali liris pada tahun 2016. BigchainDB sendiri merupakan proyek *open-source* yang terbuka untuk kontribusi dan pengembangan oleh komunitas. Keunggulan utama BigchainDB adalah kemampuannya untuk memiliki throughput besar namun tetap memiliki semua keuntungan dari teknologi blockchain yaitu *immutability* [28]. BigchainDB dapat memperoleh keuntungan dari konsensus yang aman dan cepat yang disediakan oleh Tendermint. BigchainDB memiliki sifat untuk *instant finality*, dimana ketika block baru dibuat maka tindakan tidak bisa kembali. Tendermint adalah sebuah platform konsensus Byzantine Fault Tolerant (BFT) yang dirancang untuk melakukan replikasi dan konsensus pada jaringan blockchain. Semenjak BigchainDB menjadi versi 2, sekarang tidak ada lagi yang bernama master node. Namun terdapat primary node yang terus bergilir dan berubah-ubah tiap waktu agar mendukung sistem terdistribusi.

Untuk bekerja dengan data di BigchainDB, ada penyesuaian sedikit dari operasi *database* konvensional. Yaitu karena BigchainDB memiliki fitur *immutability* maka proses manipulasi data yaitu CRUD berubah menjadi CRAB. Untuk perbedaannya dapat dilihat pada tabel 2.3.

Tabel 2.3 Operasi Data BigchainDB

Database : CRUD	BigchainDB : CRAB
Create	Create
Read	Retrieve

Update	Append (membuat transaksi / data baru untuk memperbarui suatu data)
Delete	Burn (membuat transaksi / data baru untuk memperbarui suatu data dengan menambahkan status “burned”, sehingga data yang terhapus masih ada di dalam blockchain)

2.2.9 Democracy JS

Democracy JS adalah salah satu *library* untuk bahasa pemrograman Javascript yang digunakan untuk melakukan konsensus *leader election*. Leader akan berpindah pindah pada setiap *nodes* yang bergabung. Dibuat suatu sistem *pub sub messaging* sederhana untuk membagi data antar beberapa *nodes* yang terdistribusi. DemocracyJS memiliki fitur yang cukup lengkap dan mudah dalam implementasinya.

Penulis menggunakan democracy.JS untuk memberitahu backend ketika ada data baru yang masuk ke dalam *database* BigchainDB, sehingga backend akan segera memperbarui *firewall rule* pada sistem *node*-nya sendiri. Hal ini karena tidak adanya sistem seperti *smart contract* pada BigchainDB untuk memberi tahu jika ada transaksi baru yang dibuat pada *blockchain*-nya. Untuk ilustrasinya dapat dilihat pada gambar 2.23.



Gambar 2.23 Ilustrasi Penggunaan DemocracyJS

2.2.10 Standar Deviasi dan Variansi

Standar deviasi dan variansi adalah teknik statistik yang digunakan untuk mengukur sebaran data dalam suatu data populasi. Kedua ukuran ini sering digunakan dalam analisis statistik dan pengambilan keputusan.

Variansi adalah ukuran statistik yang menggambarkan seberapa jauh data-data yang kita miliki tersebar dari nilai rata-ratanya. Variansi digunakan untuk mengukur tingkat variasi yang ada dalam suatu dataset. Rumus untuk menghitung variansi pada suatu populasi adalah sebagai berikut:

$$\text{Variansi populasi} = \Sigma(x_i - \bar{x})^2 / (n)$$

di mana x_i adalah setiap titik data, \bar{x} adalah rata-rata populasi, dan n adalah jumlah total titik data dalam populasi.

Standar deviasi adalah akar kuadrat dari variansi. Standar deviasi menggambarkan berapa banyak nilai atau jumlah data yang berbeda dari rata-rata, hal ini dapat digunakan untuk mengukur heterogenitas dari suatu kumpulan data.

$$\text{Standar deviasi populasi} = \sqrt{(\Sigma(x_i - \mu)^2 / n)}$$

di mana μ adalah rata-rata populasi dan n adalah jumlah total titik data dalam populasi.

Standar deviasi yang lebih besar menunjukkan bahwa titik data cenderung lebih tersebar atau lebih bervariasi / heterogen dari rata-ratanya, sementara standar deviasi yang lebih kecil menunjukkan bahwa titik data cenderung lebih dekat atau lebih homogen terhadap rata-ratanya. Oleh karena itu penulis menggunakan standar deviasi untuk menentukan karakteristik *concentration* dan *distribution* agar dapat menjadi *features* numerik yang dapat diprediksi menggunakan *machine learning* dari serangan DDoS.

2.2.11 Machine Learning

Machine learning adalah suatu algoritma yang dapat belajar dari dataset dan membuat prediksi atau pengklasifikasian berdasarkan pola yang teridentifikasi dalam dataset tersebut. *Machine learning* bagus untuk digunakan dalam

pengklasifikasian pola stokastik. Karena walaupun pola stokastik berasal dari suatu fungsi acak, namun memiliki sentral struktur dan aturan yang sama tiap pola stokastiknya. Hal ini bisa diidentifikasi menggunakan teknologi *machine learning*. Klasifikasi adalah *supervised learning* yang mengategorikan data ke dalam kelas-kelas. Sehingga dibutuhkan dataset yang sudah dilabeli kebenarannya. Setiap algoritma *machine learning* menggunakan metode kalkulasi matematis yang berbeda beda sehingga akan mempengaruhi akurasi, karakteristik terhadap suatu data, dan kecepatan dalam memprediksi dan melatih suatu model *machine learning*.

Ada beberapa algoritma *machine learning* untuk klasifikasi yang populer:

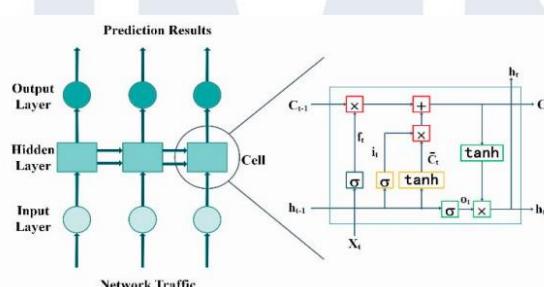
- K-Nearest Neighbors (K-NN): Algoritma ini mengklasifikasikan data dengan melihat tetangga terdekatnya dari suatu titik dengan mengalkulasikan *Euclidean distance*. K-NN bagus digunakan pada set *features* yang kecil dengan dataset yang sedikit, namun cocok juga pada set *features* yang besar dengan dataset yang banyak.
- Naive Bayes: Algoritma ini didasarkan pada teorema Bayes dan mengasumsikan independensi antara *features* yang ada. Dengan menghitung probabilitas kelas berdasarkan fitur-fitur yang diamati lalu mengklasifikasikan kepada data baru.
- Random Forest: Termasuk ke dalam *ensemble learning* yang terdiri dari banyak pohon keputusan acak yang bekerja bersama-sama untuk mengklasifikasikan data. Setiap pohon memberikan suara pada kelas yang dihasilkan, dan mayoritas suara digunakan untuk memutuskan kelas akhir.
- Support Vector Machines (SVM): Algoritma ini mencari *hyperplane* optimal yang memaksimalkan margin antara kelas-kelas yang ada. SVM dapat digunakan untuk klasifikasi biner dan multi-kelas. SVM memiliki beberapa kernel yang bisa digunakan seperti linear, radian basis function, sigmoid, polynomial, dan lain lain.
- Linear Regresion : Algoritma ini akan menemukan nilai koefisien regresi yang paling cocok untuk data yang ada dengan cara mengurangi selisih nilai

prediksi dengan nilai sebenarnya. Hasilnya adalah garis linear yang menggambarkan koefisien regresi datasetnya.

- Neural Networks: Jaringan saraf tiruan adalah model yang terdiri dari neuron buatan yang saling terhubung. Lapisan-lapisan dari neuron ini dapat belajar dari data dan melakukan pengklasifikasian.

2.2.12 Bidirectional LSTM [29]

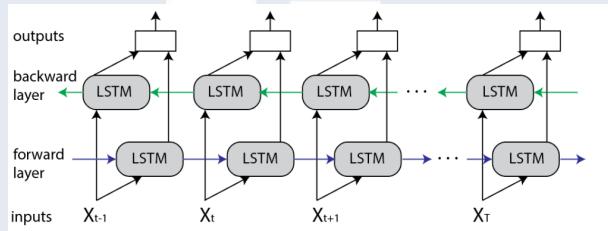
Bidirectional LSTM atau *Bidirectional Long Short-Term Memory* adalah salah satu algoritma dari *deep learning*. *Deep learning* sendiri merupakan cabang dari *machine learning* yang menggunakan saraf tiruan sebagai algoritma pembelajarannya. LSTM menggunakan jaringan saraf berulang atau yang disebut dengan RNN. RNN sendiri teknik untuk memakai output dari sel sebelumnya untuk *input* sel setelahnya. Bedanya LSTM dilakukan modifikasi dengan menambahkan sel khusus yang disebut “sel memori” untuk mengatasi *vanishing and gradient problem* atau hilangnya *memory* jangka panjang akibat melalui banyak langkah dalam waktu. Dalam arsitektur LSTM terdapat tiga gerbang untuk menentukan apakah suatu *memory* akan disimpan atau digantikan yaitu *input gate*, *forget gate*, dan *output gate*. Dengan demikian, LSTM dapat mengatasi masalah long-term *dependencies* atau ketergantungan jarak jauh dalam dataset. Ilustrasi model LSTM dapat dilihat pada gambar 2.24.



Gambar 2.24 Struktur LSTM [29]

Perbedaan utama antara LSTM biasa dengan *bidirectional LSTM* adalah bahwa *bidirectional LSTM* memproses data baik dari awal ke akhir maupun dari akhir ke awal secara bersamaan dan paralel. Hal ini dibutuhkan 2 modul LSTM namun keduanya memiliki arah yang berlawanan. Dengan kata lain, pada setiap

langkah waktu, *bidirectional LSTM* memiliki dua modul LSTM, satu mengambil urutan data dari awal ke akhir dan yang lainnya dari akhir ke awal. Hal ini memungkinkan model untuk memperoleh informasi lebih dari kedua arah, yang dapat meningkatkan kemampuan prediksi dan akurasi data. Namun dengan menambahkan kompleksitas layer *deep learning* yaitu dengan menambahkan satu lagi modul LSTM maka waktu prediksi akan lebih lama. Struktur *bidirectional LSTM* dapat dilihat pada gambar 2.25.



Gambar 2.25 Struktur *Bidirectional LSTM* [29]

Penulis akan menggunakan *Bidirectional LSTM* untuk mendapatkan akurasi yang maksimal untuk deteksi serangan DDoS, karena penulis akan menggunakan *time window* yang cukup besar dari model LSTM ini karena ada korelasi antara *time window* yang besar maka akurasi yang didapatkan akan lebih besar juga.

2.2.13 *Imbalance dataset*

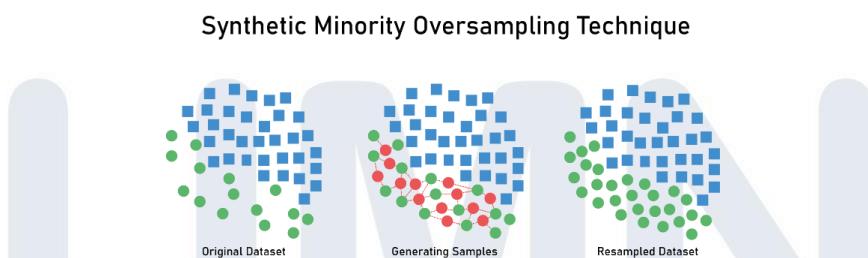
Ketika mengumpulkan dataset jaringan komputer pasti akan ditemukan ketidakseimbangan antara dataset normal dengan dataset serangan DDoS. Hal ini disebabkan karena beberapa faktor, yang pertama karena ketika serangan DDoS akan menghasilkan data yang lebih banyak karena data yang terkirim jauh lebih banyak daripada kondisi normalnya, hal ini sesuai dengan karakteristik serangan DDoS. Yang kedua ketika skenario antara kondisi normal dan DDoS lebih banyak ke salah satu kondisi saja, sehingga menyebabkan ketidakseimbangan dataset.

Kondisi ini disebut dengan *imbalance dataset*. Masalah dengan *imbalance dataset* adalah bahwa model yang dilatih pada dataset ini cenderung memiliki kinerja yang buruk dalam memprediksi kelas yang lebih kecil. Hal ini disebabkan oleh perbedaan jumlah sampel yang signifikan antara kelas mayoritas dan minoritas. Namun sebenarnya jika kondisi tidak terlalu timpang, tidak cukup

menjadi masalah untuk model *machine learning* berlatih. Terdapat beberapa cara untuk mengatasi masalah imbalance dataset atau dataset yang tidak seimbang, beberapa pilihan adalah Over-sampling dan Under-sampling.

2.2.13.1 Smote [30]

SMOTE atau *Synthetic Minority Over-sampling Technique* adalah salah satu teknik *oversampling* yang digunakan untuk mengatasi ketidakseimbangan dataset. Teknik ini dirancang untuk menangani masalah *imbalance* dataset dengan meningkatkan jumlah sampel dalam kelas minoritas. SMOTE bekerja dengan cara menciptakan sampel sintetis baru untuk kelas minoritas dengan menggabungkan dan menginterpolasi fitur dari sampel yang sudah ada. Dengan ini akan membantu mengurangi ketidakseimbangan dataset dan meningkatkan kinerja model *machine learning* dalam memprediksi kelas minoritas. SMOTE telah terbukti efektif dalam berbagai kasus ketidakseimbangan kelas dan merupakan salah satu teknik *oversampling* yang populer dalam penanganan dataset yang tidak seimbang. Oleh karena itu penulis akan menggunakan teknik *oversampling* SMOTE untuk mengatasi masalah *imbalance dataset*. Gambar 2.26 merupakan ilustrasi dari teknik SMOTE :



Gambar 2.26 Ilustrasi Teknik SMOTE untuk menangani masalah dataset tidak seimbang [30]

2.2.14 ReactJS

React adalah salah satu library Javascript dan *framework* yang populer untuk membangun *frontend* (antar muka) yang interaktif dan responsif. Dikembangkan oleh Facebook, React memungkinkan pengembang untuk membuat komponen UI yang dapat digunakan kembali dan mengatur keadaan (*state*) aplikasi dengan efisien. Terdapat beberapa fungsi seperti `useEffect`, `useState`, `porps`, dan lain lain

untuk mendukung fungsionalitas frontend. Dengan fleksibilitas, performa yang baik, dan ekosistem yang luas, React memberikan pendekatan yang kuat untuk membangun antarmuka pengguna yang interaktif dan kompleks. Dengan berbagai kelebihan tersebut, penulis akan menggunakan *framework* react untuk membangun frontend admin panel sistem yang akan dibangun.

2.2.15 Express JS

Express.js adalah *framework* aplikasi web untuk membuat sistem backend yang sederhana dan fleksibel untuk berjalan di atas Node.js. Dirancang untuk membangun aplikasi web dan layanan API dengan cepat dan mudah, Express.js memberikan fondasi yang kuat untuk mengelola rute (*routing*), penanganan permintaan dan respons (*request/response*), serta integrasi dengan berbagai modul atau *middleware*.

Keunggulan dari Express JS adalah :

1. Memudahkan *routing* untuk menangani permintaan HTTP yang sesuai
2. Mudah menambahkan *middleware* untuk melakukan validasi dan membuat JWT
3. Modular yang memudahkan untuk mengorganisir folder proyek
4. Berdasarkan survei Express.js popular karena mudah dipelajari, fleksibel, dan ringan

Oleh karena itu penulis akan menggunakan Express.js untuk membangun backend yang berhubungan dengan BigchainDB, React, dan *firewall controller* pada sistem yang akan dibangun.

2.2.16 Hping3

Hping3 adalah software yang digunakan untuk melakukan pengujian dan pemindaian jaringan. Hping3 memungkinkan kita untuk mengirim paket khusus ke tujuan tertentu dalam jaringan, serta melakukan pemantauan dan analisis terhadap respons yang diterima. Ini dapat digunakan untuk berbagai tujuan, termasuk pemecahan masalah jaringan, pemantauan kinerja, pengujian keamanan, dan pemahaman lebih dalam tentang perilaku jaringan. Hping3 dapat bekerja pada 3 protokol yaitu TCP, UDP, dan ICMP. Hping3 dapat menjadi alat untuk

menyimulasikan serangan DDoS karena Hping3 memiliki mode flood untuk mengirim paket sebanyak dan secepat mungkin ke perangkat tujuan. Hping3 juga memiliki konfigurasi yang fleksibel seperti besar paket dan header untuk menyimulasikan serangan DDoS dengan berbagai jenis. Penulis akan menggunakan Hping3 untuk menjalankan scenario serangan DDoS ke sistem yang dibangun.

2.2.17 Jmeter

JMeter adalah software yang digunakan untuk melakukan *load testing* dan *performance testing* terhadap web server. JMeter dikembangkan oleh Apache Software Foundation dan ditulis dalam bahasa pemrograman Java. Dengan JMeter, kita dapat membuat skenario pengujian yang menyimulasikan akses pengguna ke aplikasi web. Skenario dibuat untuk mengirimkan permintaan HTTP ke server, merekam respons server, dan menganalisis kinerja aplikasi dengan mengukur waktu respons. Hasil pengujian dapat ditampilkan dalam bentuk grafik, tabel, atau laporan untuk analisis lebih lanjut. Penulis memilih aplikasi JMeter untuk digunakan mendapatkan metrik pengujian berupa *response time* dari proxy server.

2.2.18 IP Spoofing [31]

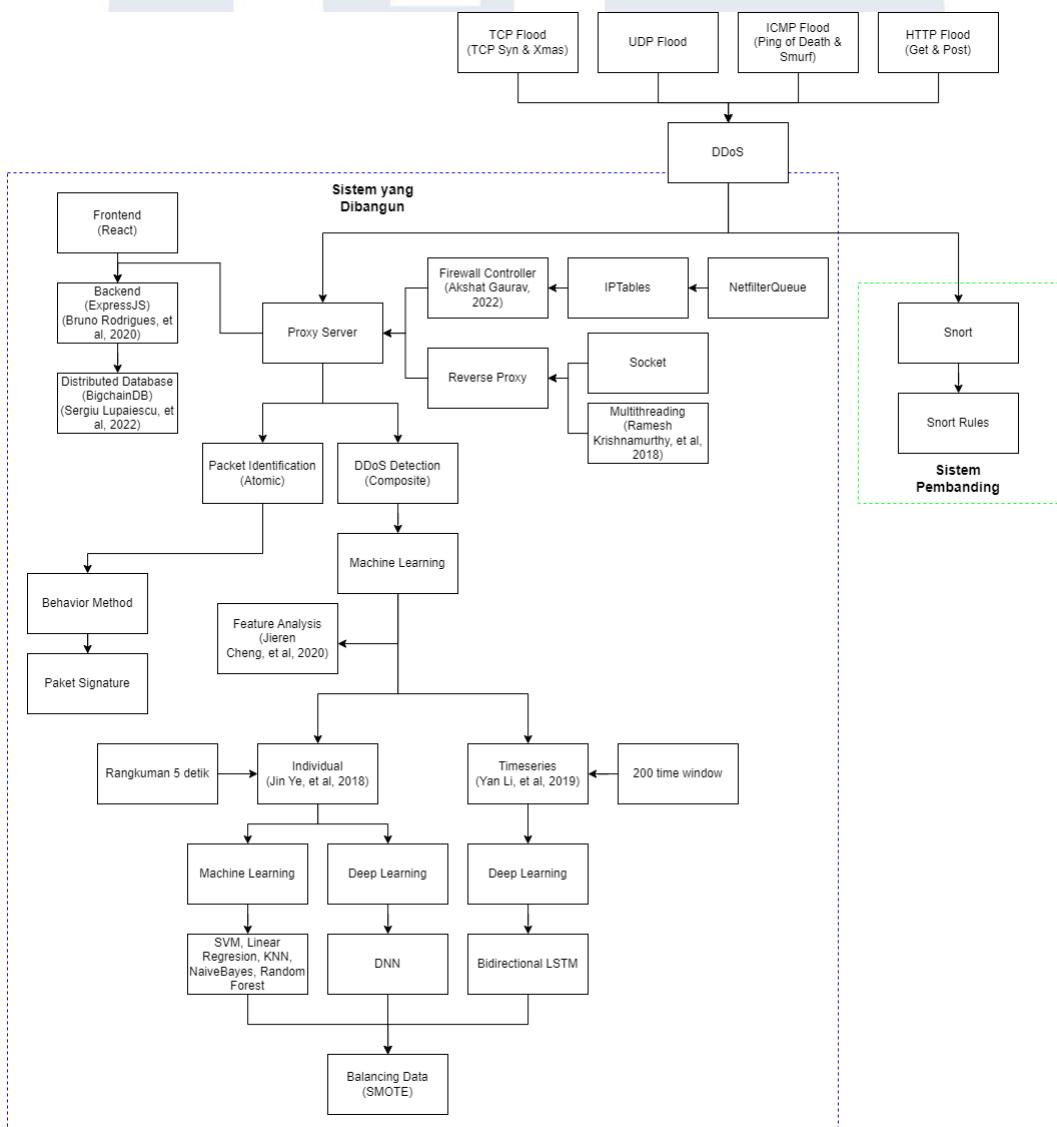
IP spoofing adalah praktik mengirim paket data dengan memalsukan alamat IP pengirimnya. Saat memproses data, komputer atau jaringan sering menggunakan alamat IP pengirim untuk mengidentifikasi asal paket dan memutuskan apakah harus menerima atau menolaknya. Dalam serangan IP spoofing, penyerang mencoba memanipulasi alamat IP pengirim untuk menyembunyikan identitasnya atau untuk memalsukan identitasnya sebagai sumber yang sah. Teknik IP spoofing ada bermacam macam, salah satunya memodifikasi *header* paket yang berisi informasi yang digunakan oleh perangkat yang ditujunya.

2.3 Simpulan

Berdasarkan studi dari referensi penelitian sebelumnya dan juga tinjauan dasar teori, maka penulis memiliki kesimpulan bahwa sistem proxy mitigasi DDoS berbasis *machine learning* dan *blockchain* yang akan dibuat memiliki acuan dan spesifikasi sebagai berikut :

1. Menjadikan rangkuman penelitian sebelumnya menjadi acuan dan referensi penelitian sistem yang akan dibangun.
2. Penulis hanya meneliti serangan DDoS dengan protokol jaringan komputer yaitu TCP, UDP, ICMP, dan HTTP pada jaringan IPV4. Selain protokol yang disebutkan tidak diteliti, karena keempat protokol tersebut merupakan protokol yang sering digunakan.
3. Untuk menyimulasikan keadaan normal dan serangan DDoS akan menggunakan *tools Hping3*. Tipe serangan DDoS yang tidak dapat dilakukan dengan *tools Hping3*, akan dibuat program secara mandiri dengan menyimulasikan karakteristik serangan DDoS tersebut yang didapatkan dari tinjauan teori.
4. Arsitektur *reverse proxy* yang dibuat akan menggunakan *library Socket* dan paradigma *multithreading* untuk mendukung *concurrent connection*. Lalu untuk uji coba dan mendapatkan hasil penelitian akan digunakan *tools JMeter* sebagai *tools* untuk *load testing*.
5. *Firewall controller* akan dibuat dengan *library NetfilterQueue* dan mengonfigurasi IP tables yang ada di OS linux.
6. Frontend sistem akan dibuat menggunakan ReactJS dan Backend sistem akan dibuat menggunakan expressJS yang dilengkapi oleh DemocracyJS dan database BigchainDB.
7. Akan diteliti 2 metode *machine learning* yaitu pendekatan secara “individual” (*probabilistic summary*) dan “timeseries”. Karena keduanya pasti memiliki kelemahan dan kelebihan satu dengan yang lainnya.
8. Temuan bahwa kompleksnya pola dari serangan DDoS dan merupakan pola stokastik. Oleh karena itu akan diteliti beberapa algoritma *machine learning* untuk klasifikasi yaitu SVM, KNN, Naïve Bayes, Random Forest, Linear Regresion, DNN, dan Bidirectional LSTM.
9. Pada dataset yang didapatkan akan digunakan teknik standar deviasi dan juga *balancing* dataset dengan teknik SMOTE.
10. Sistem yang sudah dibuat akan dibandingkan dengan IDS / IPS Snort.

Berdasarkan daftar referensi berupa penelitian terdahulu dan tinjauan dasar teori yang telah penulis bahas pada bab tinjauan teori, penulis menentukan teknologi dan metode yang sesuai pada sistem yang akan penulis rancang dan bangun. Penulis menimbang dan menentukan solusi terbaik yang dipilih berdasarkan batasan penelitian ini. Penulis hanya mempertimbangkan pada keberhasilan hasil penelitian pada jenis serangan DDoS dan protokol yang sudah ditentukan penulis sebagai latar belakang masalah dan batasan penelitian. Oleh karena itu pada gambar 2.27 ditampilkan diagram *state of the art* pada sistem yang akan dibangun oleh penulis :



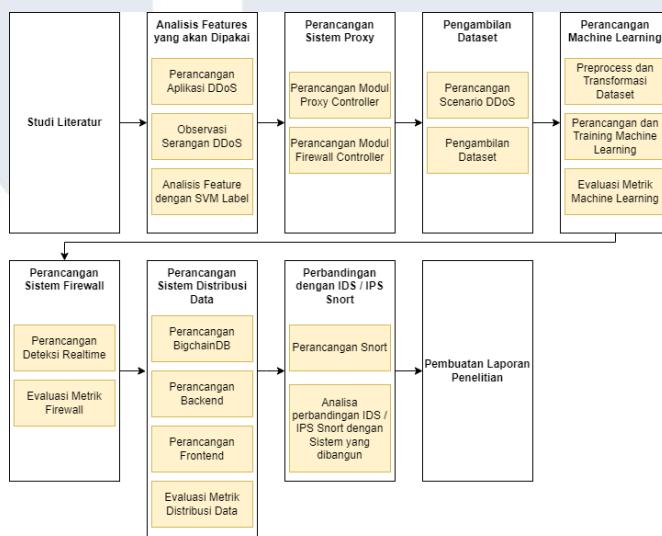
Gambar 2.27 Diagram State of The Art

BAB III

ANALISIS DAN PERANCANGAN SISTEM

3.1 Metode Penelitian

Pada penelitian ini, terdapat 9 tahapan penelitian yang dilakukan oleh penulis untuk mendapatkan hasil penelitian yaitu dengan studi pustaka, analisis features yang akan dipakai, perancangan sistem proxy, pengambilan dataset, perancangan machine learning, perancangan sistem firewall, perancangan sistem distribusi data, perbandingan dengan IDS / IPS Snort, dan pembuatan laporan penelitian. Untuk lebih jelasnya alur metode penelitian dapat dilihat pada gambar 3.1.



Gambar 3.1 Diagram tahapan penelitian

3.2 Studi literatur

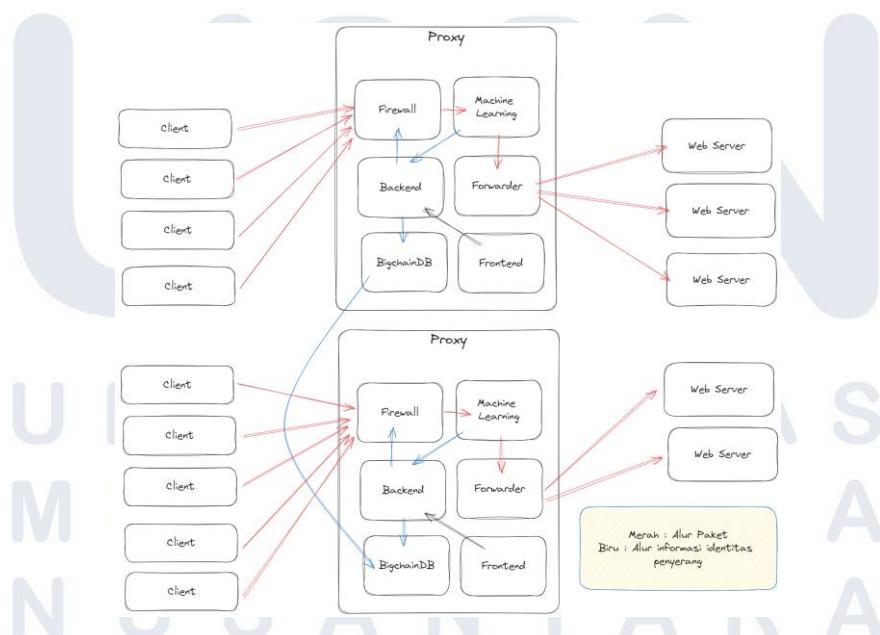
Studi literatur yang dilakukan penulis adalah dengan mencari informasi berupa penelitian terdahulu dan dokumentasi teknologi yang berkaitan dan mendukung penelitian ini. Dalam hal ini adalah penelitian yang terkait dengan proxy, *machine learning* untuk deteksi dan identifikasi serangan DDoS, dan *blockchain*. Setelah menemukan penelitian yang terkait, maka akan dianalisis pada spesifikasi sistem yang dibangun, cara implementasi, dan metode untuk mendapatkan hasil penelitiannya. Penulis juga melakukan pencarian terhadap definisi, karakteristik, dan alat untuk menyimulasikan serangan DDoS guna dilakukan observasi dan uji coba secara langsung. Penulis juga melakukan bimbingan terhadap dosen Program

Studi Teknik Komputer untuk mendapatkan informasi terkait penelitian yang dikerjakan penulis. Tujuan tahap ini adalah untuk memberikan informasi dan referensi yang cukup kepada penulis tentang hal yang akan dirancang dan dibangun.

3.3 Arsitektur General Sistem Proxy Mitigasi DDoS berbasis Machine Learning dan Blockchain

Sebelum melakukan perancangan lebih lanjut, pada gambar 3.2 merupakan gambaran awal dari rancangan sistem proxy mitigasi DDoS berbasis machine learning dan blockchain. Hal ini guna untuk menjadikan acuan dalam pengambilan keputusan untuk perencanaan modul-modul sistem dibawahnya seperti pemilihan teknologi, metode, arsitektur, dan sebagainya. Proxy akan menjadi penengah antara komunikasi *client* dan juga *web server*.

Pada sistem proxy terdapat 6 modul utama yaitu *firewall* sebagai modul pertama yang menerima paket dari *client* dan fungsi memfilter paket yang diterima atau ditolak, *machine learning* sebagai pendekripsi dan pengidentifikasi paket DDoS, *forwarder* sebagai penyalur paket ke tujuan *client*, *backend* sebagai *control plane* sistem distribusi informasi identitas penyerang, *BigchainDB* sebagai *database* untuk menyimpan identitas penyerang, dan *frontend* sebagai alat kontrol admin proxy.



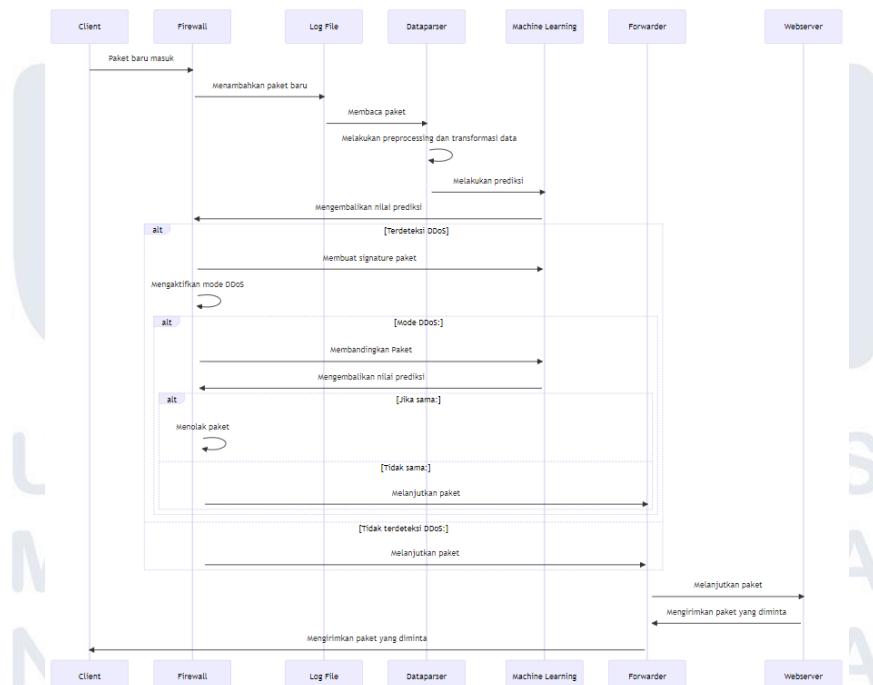
Gambar 3.2 Diagram Arsitektur Sistem secara General

3.4 Alur Sistem Proxy Mitigasi DDoS berbasis Machine Learning dan Blockchain

Berikut merupakan penjabaran ide berbentuk alur kerja dari arsitektur yang sudah dijelaskan pada bagian 3.3. Dibagi menjadi beberapa alur untuk mempermudah pembacaan.

1. Deteksi Paket

Penjelasan dari alur pada *sequence diagaram* untuk deteksi paket pada gambar 3.3 adalah sebagai berikut, *firewall* akan melakukan *logging* terus menerus ketika paket baru datang dan akan dimasukkan ke dalam *log file*. Lalu data parser akan mengambil data baru dari *log file* dan dilakukan *pre-processing* dan transformasi data sebagian *input machine learning*. Data *input* akan diprediksi oleh model *machine learning*, jika terdeteksi sebagai serangan DDoS maka akan dibuat suatu *signature* dari paket-paket yang merupakan serangan DDoS. Jika dan hanya sedang keadaan DDoS, maka setiap paket yang masuk akan diidentifikasi dengan melakukan perbandingan dengan *signature* paket DDoS yang telah dibuat. Jika sama dengan *signature* tersebut maka paket akan ditolak, jika tidak sama maka paket akan diproses ke *forwarder*.



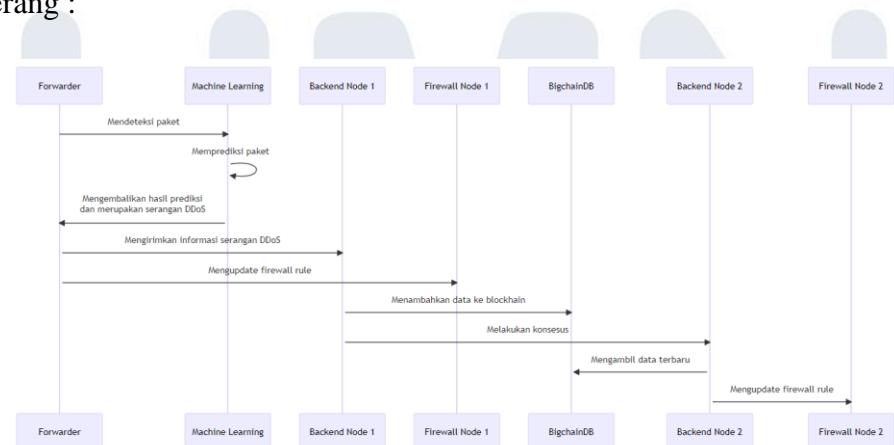
Gambar 3.3 Sequence Diagram untuk Deteksi Paket

59

Alur ini sesuai dengan solusi pada sumber referensi penelitian terdahulu nomor 2.1.1. Khusus untuk deteksi paket protokol HTTP akan dilakukan di *forwarder*, karena sesuai dengan landasan teori bahwa paket HTTP merupakan gabungan dari 1 atau lebih paket TCP maka perlu dikumpulkan terlebih dahulu di modul *forwarder* dan tidak adanya pembuatan signature dari paket HTTP (hal ini akan dijelaskan di bagian berikutnya).

2. Persebaran Data Informasi Identitas Penyerang

Informasi identitas penyerang pada penelitian ini hanya berupa *IP address* dari perangkat penyerang, dan hanya didapatkan dari deteksi protokol paket HTTP. Hal ini karena untuk paket protokol TCP, UDP, dan ICMP bisa dilakukan teknik *IP Spoofing* dengan mengubah informasi *header* paket. Untuk menghindari pemblokiran yang salah karena menggunakan IP palsu, sehingga dilakukan metode yang menggunakan identifikasi paket *signature*. Namun karena untuk melakukan komunikasi HTTP harus melakukan TCP *handshake* dengan baik terlebih dahulu, sehingga kemungkinan untuk memakai IP palsu yang diset dari *header* TCP-nya akan kecil. Jika menggunakan IP palsu pun kemungkinan penyerang menggunakan proxy atau teknologi lain untuk menyerang yang tidak dipertimbangkan pada penelitian ini. Pada gambar 3.4 ditampilkan alur pembagian data informasi identitas penyerang :

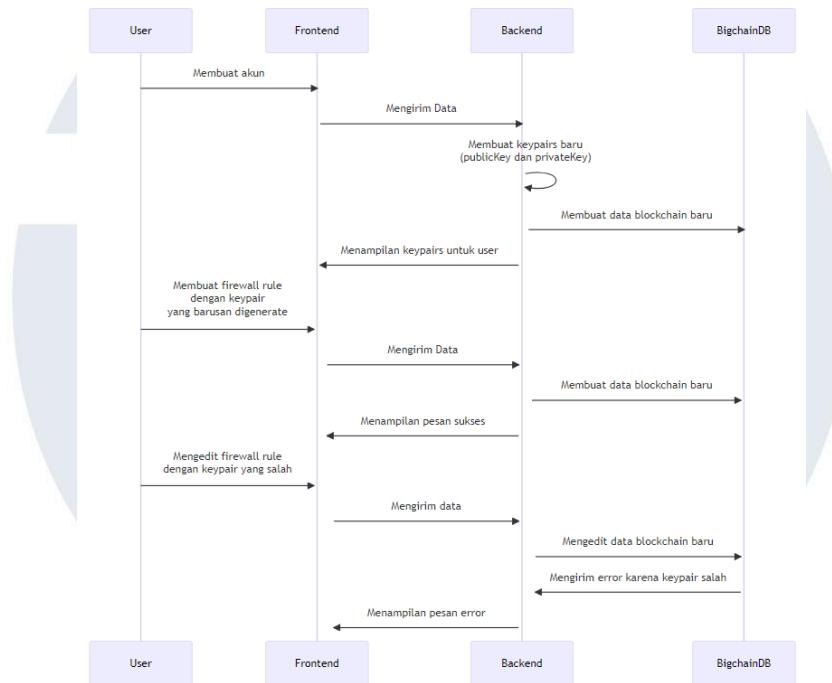


Gambar 3.4 Sequence Diagram untuk Persebaran Data Informasi Penyerang

Forwarder disini yang melakukan prediksi karena merupakan paket protokol HTTP yang sudah dijelaskan alasannya sebelumnya. *Backend* akan sebagai control

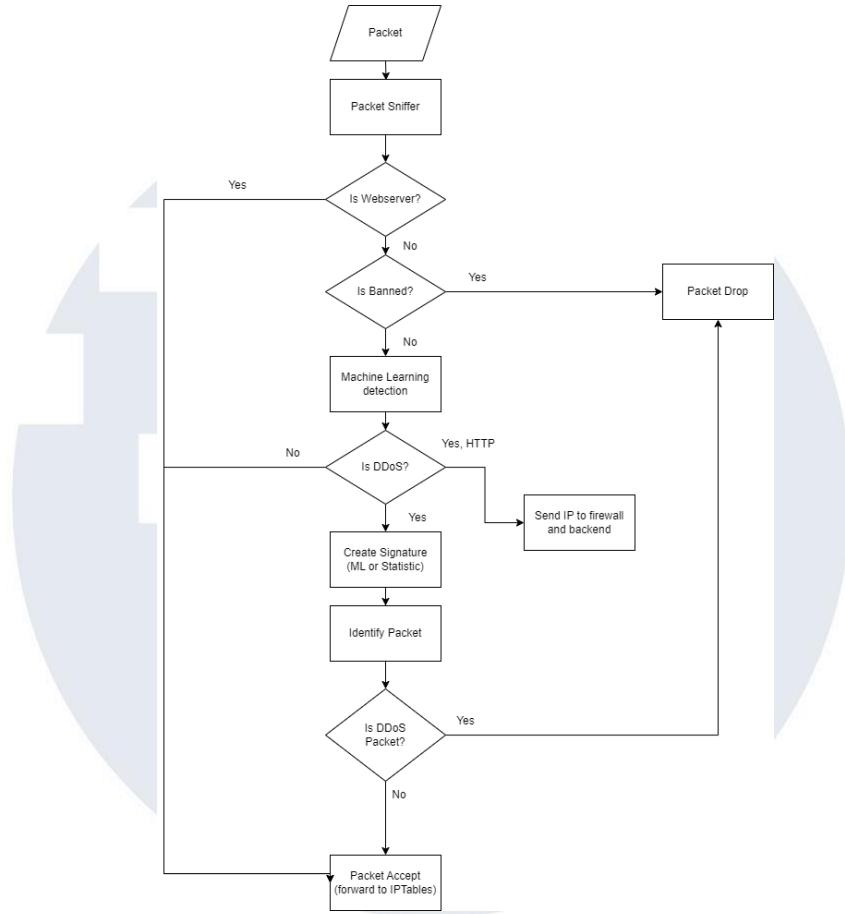
plane antara forwarder, BigchainDB, dan Firewall. Alur ini sesuai dengan solusi pada sumber referensi penelitian terdahulu nomor 2.1.4.

3. Admin Membuat Akun dan mengontrol Firewall Rule



Gambar 3.5 Sequence Diagram untuk Admin Membuat Akun dan Melakukan Operasi Database

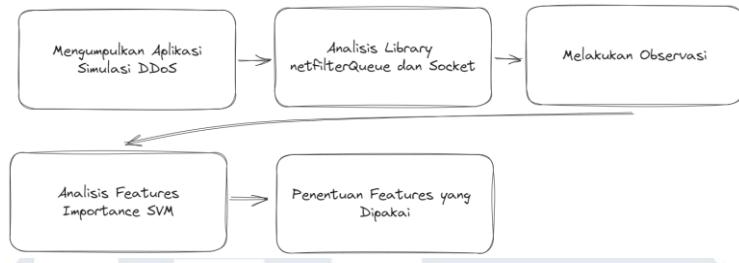
Penjelasan untuk *sequence diagram* pada gambar 3.5 adalah saat admin atau user yang mengoperasikan sistem ingin menambahkan *firewall rule* atau data pada sistem yang dibangun. Pertama kali user akan membuat akun terlebih dahulu. Setiap user yang baru mendaftar, akan mendapatkan *keypair* baru berupa *publicKey* dan *privateKey* yang akan di-*generate* oleh backend dan simpan oleh user untuk melakukan setiap transaksi di *blockchain*. Hal ini karena BigchainDB merupakan *database* yang memiliki fitur *blockchain*, oleh karena itu setiap data atau transaksi yang dibuat atau dioperasikan akan membutuhkan *keypair* yang sesuai dengan pemilik *asset* (data) yang dilakukan operasi tersebut. Jika *keypair* yang digunakan oleh user berbeda dengan *keypair* pemilik *asset*, maka transaksi tidak akan bisa dibuat. Setiap *asset* pada BigchainDB dapat berisi *keypair* yang berbeda beda sesuai dengan pemiliknya, agar user lain dapat mengoperasikannya maka diperlukan transfer asset dari satu user ke user lain.



Gambar 3.6 Alur Kerja Sistem Proxy Mitigasi DDoS berbasis Machine Learning dan DDoS

3.5 Analisis Features yang akan Dipakai

Sebelum melakukan pembuatan proxy dan pengambilan dataset, penulis melakukan analisa dan penentuan *features* yang akan dipakai di penelitian ini terlebih dahulu. Karena sesuai dengan tinjauan teori yang didapatkan, bahwa pemilihan *features* akan sangat krusial pada performa *machine learning* dan proxy secara keseluruhan. Karena pemilihan *features* yang buruk dapat menyebabkan akurasi *machine learning* yang buruk dan waktu proses yang lama. Pada tahap ini peneliti akan mencocokkan juga dengan informasi yang telah didapatkan di bab tinjauan teori dengan data yang bisa didapatkan. Oleh karena itu ada beberapa tahapan dalam analisa dan penentuan *features* di penelitian ini yang dapat dilihat pada diagram gambar 3.7.



Gambar 3.7 Alur Analisis Features yang akan Dipakai

1. Mengumpulkan aplikasi simulasi DDoS

Untuk menguji coba dan melakukan observasi simulasi serangan DDoS dan normal, maka penulis mengumpulkan alat dan tutorial mengenai berbagai *software* yang sering digunakan. Penulis juga akan membuat aplikasi sendiri berdasarkan karakteristik yang didapatkan karena tidak menemukan alat yang sesuai. Pada bahasan ini akan dikategorikan ke protokol yang digunakan dan serangan yang sudah ditetapkan pada bab tinjauan teori. Dan akan dijabarkan perintah yang dijalankan dengan penjelasan argumen yang digunakan.

A. TCP

menggunakan aplikasi Hping3 yang dikonfigurasi dengan mengikuti dokumentasi aplikasi [32].

- a. TCP Flood Syn Single IP : hping3 -S --flood -p {port} {IP address}
- b. TCP Flood Syn Random IP : hping3 -S --flood -p {port} {IP address}--rand-source
- c. TCP Flood XMAS Single IP : hping3 -SARFU --flood -p {port} {IP address}
- d. TCP Flood XMAS Random IP : hping3 -SARFU --flood -p {port} {IP address}--rand-source

-S memiliki arti untuk menggunakan TCP Flags SYN, -SARFU memiliki arti untuk menggunakan TCP Flags SYN ACK RST FIN URG, -flood untuk mengirim *request* secepat mungkin tanpa adanya delay, dan –rand-source untuk menggunakan IP yang berbeda beda pada setiap paketnya.

B. UDP

- a. UDP Flood Single IP : hping3 --flood -2 -p {port} {IP address}
- b. UDP Flood Random IP : hping3 --flood -2 -p {port} {IP address} -rand-source
- c. UDP Flood paket besar : hping3 --flood -2 -p {port} {IP address} -d 65535
 - 2 memiliki arti untuk menggunakan protokol UDP, --flood untuk mengirim *request* secepat mungkin tanpa adanya *delay*, -d adalah untuk menentukan besar paket yang akan dikirim (defaultnya adalah 64 bytes, dengan maksimal 65535 bytes), dan –rand-source untuk menggunakan IP yang berbeda beda pada setiap paketnya.
- d. Skenario normal dengan waktu *delay* yang bervariasi

Dibuat aplikasi dengan mengirim data konstan ke server namun dengan *delay* yang acak yaitu sekitar 0 – 10 detik dan diharuskan menunggu menerima data dari server. Karena pada normalnya aplikasi berkomunikasi 2 arah (mengirim dan menerima). Aplikasi ini menggunakan *library* Socket sebagai pengirim datanya. Cuplikan dari kodennya pada gambar 3.8.

```
while time.time() < t_end:  
    try:  
        msgFromClient = 'HI UDP SERVER'  
        bytesToSend = str.encode(msgFromClient)  
        start_time = time.perf_counter()  
        # Send to server using created UDP socket  
        UDPClientSocket.sendto(bytesToSend, serverAddressPort)  
        msgFromServer = UDPClientSocket.recvfrom(bufferSize)  
        print(f"\nPacket send to {serverAddressPort}, response time:  
        msg = "Message from Server {}".format(msgFromServer[0])  
        print(msg)  
    except Exception as e:  
        print(f"Send packet error, error : {e}")  
  
    # Random sleep time from 0 to 10 seconds  
    time.sleep(random.randint(0, 10))
```

Gambar 3.8 Potongan Kode Program Random Request UDP

- e. Skenario normal dengan *streaming* video dengan transmisi UDP

Dibuat aplikasi yang akan melakukan permintaan data berupa gambar dan dilakukan secara *realtime* dengan koneksi UDP sehingga menjadi video. Untuk potongan kode ditampilkan pada gambar 3.9.

```

while True:
    msg,client_addr = server_socket.recvfrom(BUFF_SIZE)
    print('GOT connection from ',client_addr)
    WIDTH=400
    frame = q.get()
    encoded,buffer = cv2.imencode('.jpeg',frame,[cv2.IMWRITE_JPEG_QUALITY,80])
    message = base64.b64encode(buffer)
    server_socket.sendto(message,client_addr)
    frame = cv2.putText(frame,'FPS: '+str(round(fps,1)),(10,40),cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)
    if cnt == frames_to_count:
        try:
            fps = (frames_to_count/(time.time()-st))
            st=time.time()
            cnt=0
            if fps>FPS:
                TS+=0.001
            elif fps<FPS:
                TS-=0.001
            else:
                pass
        except:
            pass
        cnt+=1
        cv2.imshow(' TRANSMITTING VIDEO', frame)
        key = cv2.waitKey(int(1000*TS)) & 0xFF
        if key == ord('q'):
            os._exit(1)
            TS=False
            break

```

Gambar 3.9 Potongan Kode Program Video Streaming UDP

C. ICMP

- a. ICMP Flood Single IP : hping3 --icmp --flood {IP address}
- b. ICMP Flood Random IP: hping3 --icmp --flood {IP address} --rand-source
- c. ICMP Smurf Flood Single IP: hping3 --icmp -d 1400 --flood {IP address}
- d. ICMP Smurf Flood Random IP : hping3 --icmp -d 1400 --flood {IP address} --rand-source
--ICMP memiliki arti untuk memakai protokol ICMP, --flood untuk mengirim *request* secepat mungkin tanpa adanya *delay*, -d adalah untuk menentukan besar paket yang akan dikirim, dan --rand-source untuk menggunakan IP yang berbeda beda pada setiap paketnya.
- e. Normal : melakukan ping biasa dan membuat *bash script* yang akan memanggil Hping3 dengan besar paket dan *delay* yang acak.
Potongan kode ditampilkan pada gambar 3.10.

```

while true; do
    packet=$(shuf -i 1-20 -n 1)
    bytes=$(shuf -i 64-200 -n 1)
    pause=$(shuf -i 1-5 -n 1)
    sudo hping3 -c $packet -d $bytes --icmp 192.168.29.128
    sleep $pause
done

```

Gambar 3.10 Potongan Kode Bash Script ICMP Normal

D. HTTP

a. HTTP Get Flood

Dibuat aplikasi yang secara terus menerus mengirim request ke suatu server tanpa adanya jeda waktu namun tetap menerima paket yang di-request. Paket yang di-request hanya 1 endpoint saja. Aplikasi ini memakai library request dari Python. Untuk potongan kodennya sebagai ditampilkan pada gambar 3.11.

```
def run():
    while time.time() < t_end:
        try:
            url_request = "http://192.168.29.128:3001/"
            start_time = time.perf_counter()
            response = requests.get(url_request, timeout=2.50)
            print(f"\nPacket send to {url_request}, response ti
            print(response.content)
        except Exception as e:
            print(f"Send packet error, error : {e}")
```

Gambar 3.11 Potongan Kode Program HTTP GET Flood

- b. HTTP Get Flood dengan tidak menerima paket dari server : sama seperti aplikasi pertama namun hanya mengirim paket HTTP dengan library socket tanpa menerima paket. Potongan dari kode ini ditampilkan pada gambar 3.12.

```
packet = str("GET / HTTP/1.1\nHost: "+host+"\n\n User-Agent: "+random.choice(uagent)+"\n"+data)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host,int(port)))
if s.sendto( packet, (host, int(port)) ):
    s.shutdown(1)
```

Gambar 3.12 Potongan Kode Program HTTP GET Flood 2

- c. HTTP Get Flood tanpa ada isi paket : sama seperti aplikasi pertama namun hanya mengirim paket tanpa adanya isi paket. Potongan dari kode ini ditampilkan pada gambar 3.13.

```
while time.time() < t_end:
    try:
        s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        s.connect((host,int(port)))
        print(f"\nPacket send to {ip}:{port} {End}")
```

Gambar 3.13 Potongan Kode Program HTTP GET Flood 3

- d. Normal HTTP : Membuat aplikasi dimana akan memanggil 7 endpoint berbeda secara acak dan juga memiliki delay yang acak dengan multithreading untuk mensimulasikan IP public.

2. Analisis Library NetfilterQueue dan Socket

Karena informasi paket yang dapat diterima oleh sistem bisa saja berbeda dengan referensi tinjauan teori karena perbedaan teknologi yang dipakai, oleh karena itu penulis merancang suatu program sederhana yang memakai *library* netfilterQueue dan Socket untuk mencari tahu data apa saja yang bisa didapatkan. Metode yang dilakukan penulis adalah dengan membuka objek dari netfilterQueue dan socket dan juga membaca dokumentasi *library* yang digunakan. Untuk contoh kodenya ditampilkan pada gambar 3.14

```
if(sca.haslayer(TCP)):
    t = sca.getlayer(TCP)
    if t.dport in listOfWebserverPorts:
        tcp_data = {
            "ip_src": sca.src,
            "port_src": t.sport,
            "port_dest": t.dport,
            "seq": t.seq,
            "ack": t.ack,
            "dataofs": t.dataofs,
            "reserved": t.reserved,
            "flags": str(t.flags),
            "window": t.window,
            "chksum": t.chksum,
            "urgptr": t.urgptr,
            "payload_len": pkt.get_payload_len(),
        }
        print(f"sca TCP, connection : {t}, data : {t.fields}, flags: {t.fields['flags']}, timestamp : ({pkt.get_timestamp()}), len : ({pkt.get_payload_len()})")
```

Gambar 3.14 Potongan Kode *Firewall Sniffer*

Setelah dilakukan pembuatan program sederhana, dirangkumlah data data yang bisa didapatkan menggunakan library netfilterQueue dan Socket berdasarkan protokol yang digunakan.

- A. TCP: IP Source, IP Destination, Port Source, Port Destination, Sequence, Acknowledge, Dataofs, Reserved, Flags, Window, Chksum, Urgpt, Payload length
- B. UDP : IP Source, IP Destination, Port Source, Port Destination, Len (hanya besar dari data saja tidak termasuk header), Chksum, Payload length
- C. ICMP : IP Source, IP Destination, Chksum, Id, Sequence, Payload length

- D. HTTP : Thread number (dalam sistem proxy), Connection time, Payload length, IP Source, IP Destination, Port Source, Port Destination, URL, Status, Socket timeout (akan ada nilai jika koneksi socket sebelumnya ada error)

3. Melakukan observasi

Observasi akan dilakukan dengan beberapa metode yaitu, metode pertama aplikasi sederhana yang sudah dibuat sebelumnya akan diserang menggunakan aplikasi DDoS dan dilakukan observasi menggunakan logger untuk mengekstrak data ke suatu file. Dan juga dilakukan metode observasi pada aplikasi wireshark agar observasi lebih mendetail. Pada tahap ini penulis mendapatkan beberapa kesimpulan yaitu :

1. IP dan Port *Destination* tidak dibutuhkan karena pasti bernilai IP dan port dari sistem proxy tersebut.
2. Pada metode *timeseries* tidak akan digunakan IP *source* karena tidak menjadi pembeda ketika di 2 skenario yaitu single IP dan *multiple* IP.
3. Penulis mendapatkan *features* yang bisa dijadikan komparator untuk identifikasi paket berbasis karakteristik paket terbanyak. Dimana *features-features* ini pada serangan DDoS memiliki nilai yang sama pada setiap paketnya, dan pada normalnya tidak sama pada setiap paketnya. Sehingga bisa dijadikan *signature* dari paket yang merupakan serangan DDoS.
4. Pada beberapa *features* memiliki keunikan seperti pada *sequence* paket ICMP seharusnya memiliki kenaikan nilai yang linear namun pada serangan DDoS nilai ini tidak linear dan bervariasi. Sehingga feature tersebut bisa ditetapkan.

No.	Time	Source	Destination	Protocol	Length	Info
31398	11.3377271	192.168.29.136	192.168.29.1	TCP	68	33288 → 3801 [SYN] Seq=0 ulin512 Len=0
31399	11.3377592	192.168.29.136	192.168.29.1	TCP	68	33289 → 3801 [SYN] Seq=0 ulin512 Len=0
31399	11.3377603	192.168.29.136	192.168.29.1	TCP	68	33289 → 3801 [SYN] Seq=0 ulin512 Len=0
31399	11.3380053	192.168.29.136	192.168.29.1	TCP	68	33290 → 3801 [SYN] Seq=0 ulin512 Len=0
31399	11.3380053	192.168.29.136	192.168.29.1	TCP	68	33290 → 3801 [SYN] Seq=0 ulin512 Len=0
31399	11.3380243	192.168.29.136	192.168.29.1	TCP	68	33292 → 3801 [SYN] Seq=0 ulin512 Len=0
31400	11.338249	192.168.29.136	192.168.29.1	TCP	68	33294 → 3801 [SYN] Seq=0 ulin512 Len=0
31400	11.33845	192.168.29.136	192.168.29.1	TCP	68	33295 → 3801 [SYN] Seq=0 ulin512 Len=0
31400	11.33845	192.168.29.136	192.168.29.1	TCP	68	33295 → 3801 [SYN] Seq=0 ulin512 Len=0
31401	11.338802	192.168.29.136	192.168.29.1	TCP	68	33297 → 3802 [SYN] Seq=0 ulin512 Len=0
31401	11.338802	192.168.29.136	192.168.29.1	TCP	68	33297 → 3802 [SYN] Seq=0 ulin512 Len=0
31401	11.338808	192.168.29.136	192.168.29.1	TCP	68	33299 → 3803 [SYN] Seq=0 ulin512 Len=0
31401	11.339240	192.168.29.136	192.168.29.1	TCP	68	33299 → 3803 [SYN] Seq=0 ulin512 Len=0
31401	11.339252	192.168.29.136	192.168.29.1	TCP	68	33300 → 3803 [SYN] Seq=0 ulin512 Len=0
31401	11.339253	192.168.29.136	192.168.29.1	TCP	68	33300 → 3803 [SYN] Seq=0 ulin512 Len=0
31401	11.339563	192.168.29.136	192.168.29.1	TCP	68	33302 → 3802 [SYN] Seq=0 ulin512 Len=0
31401	11.339980	192.168.29.136	192.168.29.1	TCP	68	33303 → 3803 [SYN] Seq=0 ulin512 Len=0
31410	11.339878	192.168.29.136	192.168.29.1	TCP	68	33304 → 3803 [SYN] Seq=0 ulin512 Len=0
31410	11.339878	192.168.29.136	192.168.29.1	TCP	68	33304 → 3803 [SYN] Seq=0 ulin512 Len=0
31412	11.340179	192.168.29.136	192.168.29.1	TCP	68	33306 → 3801 [SYN] Seq=0 ulin512 Len=0
31413	11.340491	192.168.29.136	192.168.29.1	TCP	68	33307 → 3802 [SYN] Seq=0 ulin512 Len=0
31414	11.340508	192.168.29.136	192.168.29.1	TCP	68	33308 → 3803 [SYN] Seq=0 ulin512 Len=0
31415	11.340797	192.168.29.136	192.168.29.1	TCP	68	33309 → 3802 [SYN] Seq=0 ulin512 Len=0
31416	11.340797	192.168.29.136	192.168.29.1	TCP	68	33309 → 3802 [SYN] Seq=0 ulin512 Len=0

Gambar 3.15 Contoh Observasi Menggunakan Software Wireshark

4. Analisis *features importance* SVM

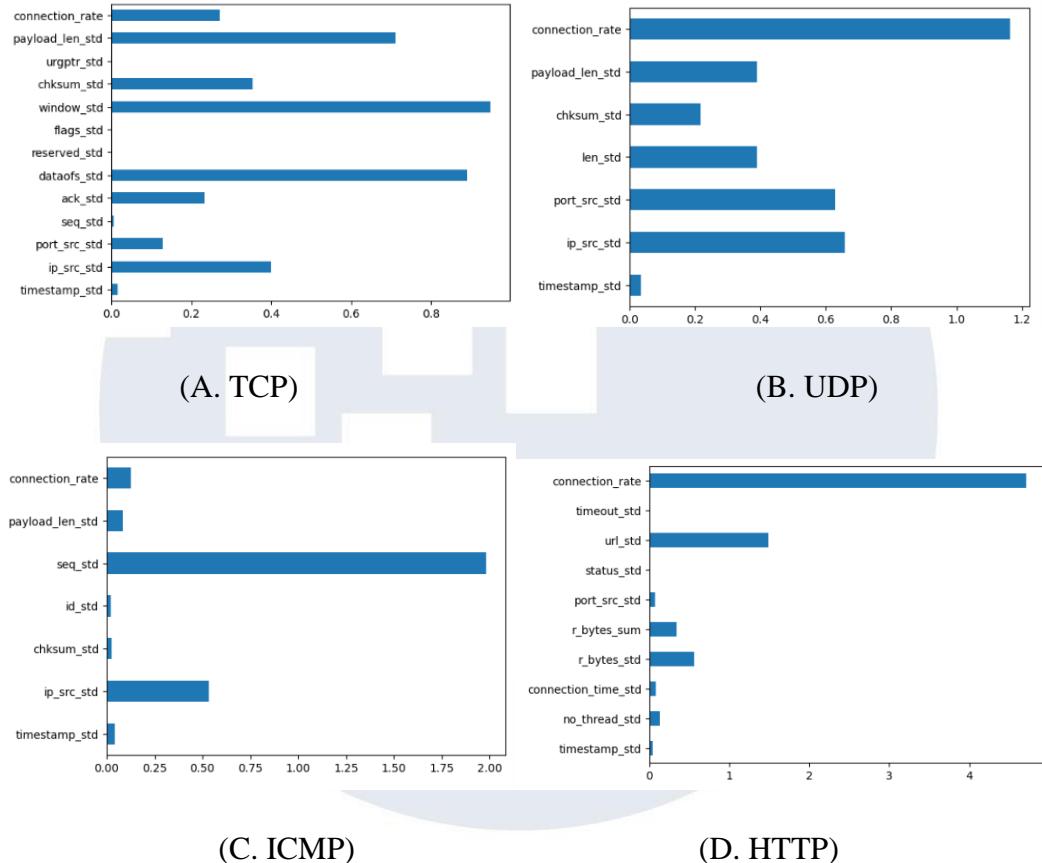
Penulis juga melakukan analisa pada model *machine learning* SVM yaitu *feature importance*. Dimana akan dilakukan *training* data pada algoritma SVM dan akan dilihat data pada model *machine learning* latihnya. Hal ini memiliki tujuan untuk melihat *features* apa yang paling berpengaruh pada saat algoritma SVM berlatih. Dimana jika semakin tinggi pengaruhnya berarti *features* tersebut penting dan memiliki pola khusus untuk proses klasifikasi.

Penulis menggunakan metode analisis *features* berbasis algoritma SVM karena *machine learning* SVM dapat diandalkan untuk menghasilkan akurasi yang tinggi dan baik pada klasifikasi data biner atau kelas yang sedikit. SVM menggunakan konfigurasi *kernel* yang akan menghasilkan *hyperplane* yang optimal untuk membagi antar kelas walaupun dengan *input* data dimensi yang tinggi [33].

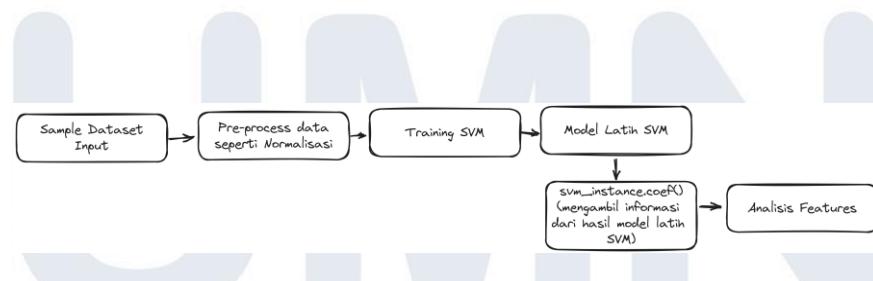
Pada gambar 3.16 dan 3.17 ditampilkan cuplikan kode fungsi yang digunakan untuk analisis *features* pada SVM dan hasil dari plot diagramnya. Hal ini digunakan untuk menyingkirkan *features* yang tidak berpengaruh yang menyebabkan waktu prediksi semakin lama. Namun tidak semua *features* yang tidak berpengaruh dihilangkan karena disesuaikan juga dengan karakteristik dari serangan DDoS.

```
features_names = ['timestamp_std', 'ip_src_std', 'port_src_std', 'len_std', 'chksum_std', 'payload_len_std', 'connection_rate']
pd.Series(abs(svm_inst.coef_[0]), index=features_names).plot(kind='barh')
```

Gambar 3.16 Kode SVM features Importance



Gambar 3.17 Hasil Plot SVM *features Importance*



Gambar 3.18 Alur Kerja Analisis *features importance* SVM

Gambar 3.18 menjelaskan alur kerja dari analisis *features importance* berbasis *machine learning* SVM, dimana *sample dataset* didapatkan dan dinormalisasi terlebih dahulu lalu digunakan untuk melatih model SVM. Setelah model SVM dihasilkan maka dapat diambil informasi koefisien dari setiap *features* data yang digunakan. Setelah memodifikasi pada *features* data, akan dilatih ulang untuk mendapatkan akurasi yang tertinggi dari model SVM.

5. Penentuan *features* yang akan dipakai

Setelah dilakukan analisis dan observasi pada serangan DDoS dan juga atas referensi dari tinjauan teori, maka penulis menetapkan *features* yang akan dipakai. Setiap protokol memiliki *features* yang berbeda beda sesuai data yang bisa didapatkan dari sistem yang dibangun, karakteristik, dan acuan dari penelitian yang sebelumnya. Beberapa informasinya diambil dari *header* paket protokol. *Features* akan dibagi menjadi 3 karena untuk metode *machine learning* ada 2 yaitu individual dan *timeseries* dan *comparator* untuk proses identifikasi setiap paket :

A. Individual

Jika *features* ada tulisan _std berarti merupakan standar deviasi dari rangkuman beberapa paket pada *dataset* tersebut. Standar deviasi digunakan untuk mencari persebaran data sehingga akan memperlihatkan karakteristik serangan DDoS sesuai dengan tinjauan teori. Rumus untuk kalkulasi standar deviasi ada pada bab tinjauan teori.

- a. TCP : timestamp_std, ip_src_std, port_src_std, seq_std, ack_std, dataofs_std, reserved_std, flags_std, window_std, checksum_std, urgptr_std, payload_len_std, connection_rate
- b. UDP : timestamp_std, ip_src_std, port_src_std, len_std, checksum_std, payload_len_std, connection_rate
- c. ICMP : timestamp_std, ip_src_std, checksum_std, id_std, seq_std, payload_len_std, connection_rate
- d. HTTP : timestamp_std, no_thread_std, connection_time_std, r_bytes_std, r_bytes_sum, port_src_std, status_std, url_std, timeout_std, connection_rate

B. Timeseries

- a. TCP: Port Source, Sequence, Acknowledge, Dataofs, Reserved, Flags, Window, Chksum, Urgpt, Payload length
- b. UDP : Port Source, Len (hanya besar dari data saja tidak termasuk header), Chksum, Payload length
- c. ICMP : Chksum, Id, Sequence, Payload length

- d. HTTP : Thread number (dalam sistem proxy), Connection time, Payload length, Port Source, URL, Status, Socket timeout (akan ada nilai jika koneksi socket sebelumnya ada error)

C. Comparator (untuk signature paket)

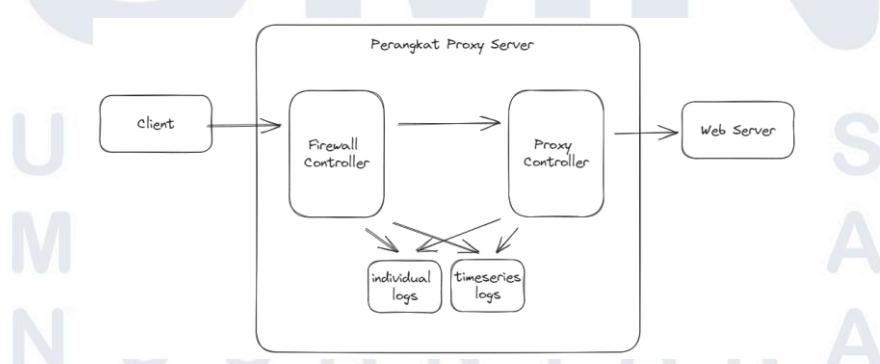
- a. TCP : dataofs, reserved, flags, window, urgptr, dan payload length
- b. UDP : len dan payload length
- c. ICMP : id dan payload length
- d. HTTP : IP source

3.6 Perancangan Sistem Proxy

Pada penelitian ini arsitektur *reverse proxy* yang akan dibangun dibagi menjadi 2 bagian karena untuk meningkatkan modularitas dan menghindari *couple tight* karena jika ada perubahan di salah satu modul tidak terpengaruh pada modul lainnya. yaitu :

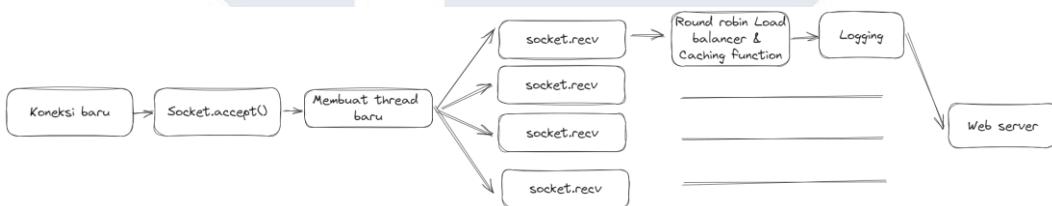
1. *Firewall controller* : Sebagai gerbang awal paket yang diterima, akan dilakukan logging terhadap paket layer 4 (*transport layer*) dengan protokol TCP, UDP, dan ICMP pada modul ini.
2. *Proxy controller* (sebelumnya disebut sebagai *forwarder*) : Sebagai inti dari program *reverse proxy*, dimana difungsikan sebagai penyalur paket layer 7 (*application layer*) dari *client* dan *web server* dan melakukan *logging* terhadap paket dengan protokol HTTP.

Untuk ilustrasi arsitekturnya dapat dilihat pada gambar 3.19.



Gambar 3.19 Arsitektur Sistem Reverse Proxy

1. *Firewall controller* akan menggunakan *library* netfilterQueue untuk mendapatkan paket dari IP tables, dan akan dilakukan *data parsing* dan *logging* pada setiap protokolnya yaitu TCP, UDP, dan ICMP. Dilakukan juga *whitelist* pada *IP address* dan *port web server* yang digunakan. Setelah dilakukan *data parsing* dan *logging* maka paket akan diteruskan ke *proxy controller*.
2. *proxy controller* digunakan library Python Socket server untuk menerima paket dari *client* dan socket client untuk mengirimkan data ke *webserver*. Akan dibuat dengan paradigma *multithreading* agar dapat menangani masalah *concurrent connection*. Akan dilakukan juga *logging* pada protokol HTTP. Pada *proxy controller* ini akan diimplementasikan *load balancing* dengan algoritma *round robin* dan sistem *caching* menggunakan *memory store*. Untuk arsitektur dari penggunaan paradigma *multithreading* pada *library socket* dapat dilihat pada gambar 3.20.



Gambar 3.20 Arsitektur Multithreading Socket

Karena fungsi *logging* dilakukan secara *multithreading* dan menggunakan 1 file log bersama maka agar tidak menyebabkan situasi *race condition* dilakukanlah metode *thread safe logging* dengan menggunakan *library* dari python.

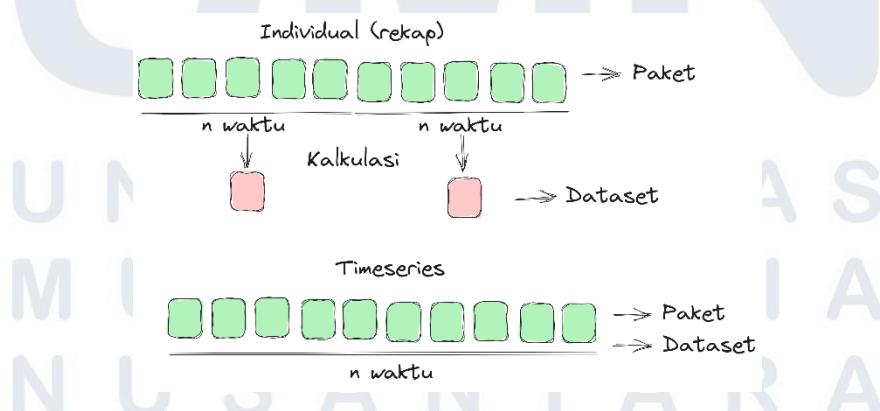
Logging paket akan dimasukkan ke suatu *file .log* dan *logging* akan berjumlah 2 yaitu untuk dataset individual (beberapa paket sampel dengan rentang waktu tertentu sudah dirangkum menjadi 1 *sample* dengan kalkulasi tertentu) dan *timeseries* (semua paket yang berlalu disimpan). Dibuat 2 metode log untuk memastikan dataset sama dengan data *input machine learning* nantinya agar tidak ada bias yang terjadi terutama perihal konfigurasi rentang waktu tertentu. Sementara untuk proses pengambilan dataset *timeseries*, setiap packet datang akan langsung ditambahkan pada dataset *timeseries*.

Pada gambar 3.21 ditampilkan alur dari pemrosesan dataset individual:



Gambar 3.21 Sequence diagram untuk proses pengambilan dataset individual

Ada perbedaan yang mendasar dari kedua metode machine learning yang nanti akan digunakan yaitu individual dan timeseries, dimana individual akan merekap suatu fenomena atau behavior acak dengan kalkulasi tertentu sehingga akan menghasilkan data kalkulasi acak individual dan dataset timeseries akan meneliti setiap paket yang masuk kedalam sistem. Untuk ilustrasi lebih jelasnya dapat dilihat pada gambar 3.22.



Gambar 3.22 Perbedaan Metode Machine Learning Individual dan Timeseries

3.7 Pengambilan Dataset

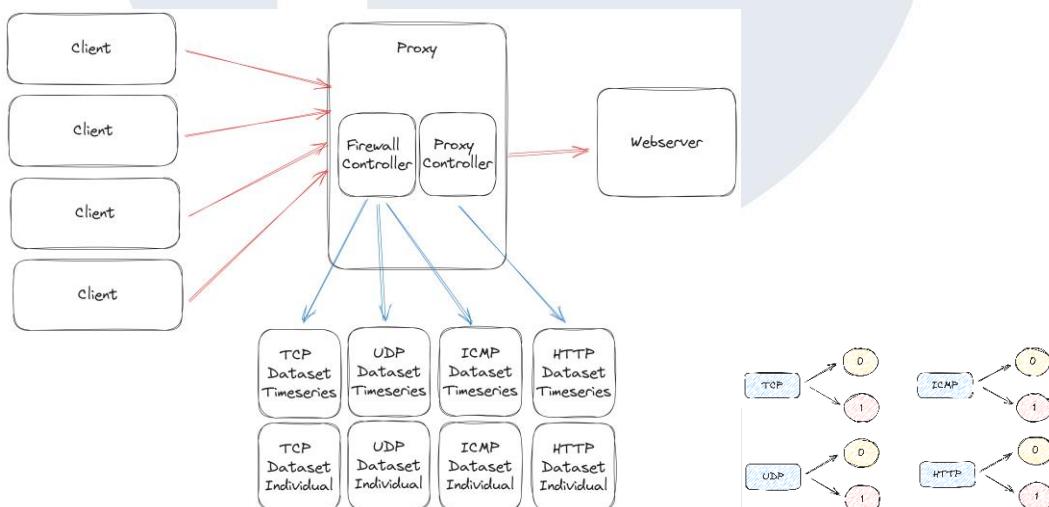
Karena deteksi *machine learning* akan dilakukan pada sistem yang dibangun sendiri, oleh karena itu untuk menghindari faktor-faktor pembeda situasi seperti perbedaan arsitektur data *logger*, infrastruktur dan lain-lain, maka dihindari menggunakan dataset publik. Sehingga dilakukan pengambilan dataset dengan sistem yang dibangun. Akan disiapkan beberapa skenario DDoS dan normal menggunakan tools Hping3 dan program yang dibuat sendiri oleh penulis berdasarkan karakteristik serangan yang didapatkan dari studi literatur. Karena penulis merencanakan untuk membuat model *machine learning* yang berbeda pada setiap protokolnya sehingga dikategorikan skenario berdasarkan protokol yang diteliti.

Scenario yang akan dijalankan adalah:

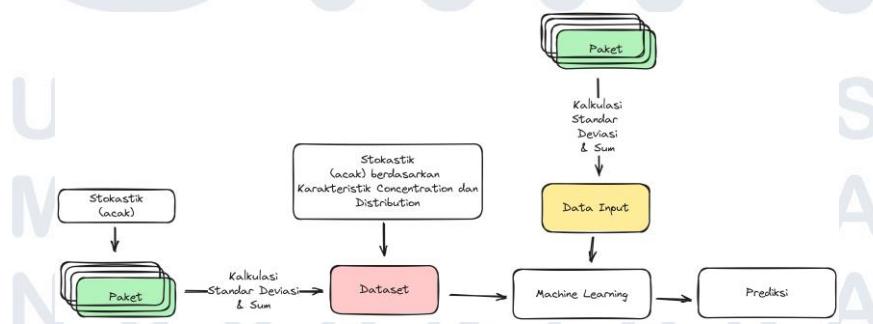
1. TCP
 - a. TCP Flood SYN Single IP : Selama 4 menit
 - b. TCP Flood SYN Random IP : Selama 4 menit
 - c. TCP Flood XMAS Single IP : Selama 4 menit
 - d. TCP Flood XMAS Random IP : Selama 4 menit
 - e. Normal : Semua skenario untuk protokol HTTP. Karena serangan HTTP akan dideteksi menggunakan model *machine learning* lain.
2. HTTP
 - a. HTTP Get Flood : Selama 4 menit
 - b. HTTP Get Flood dengan tidak menerima paket dari server : Selama 4 menit
 - c. HTTP Get Flood tanpa ada isi paket : Selama 4 menit
 - d. Normal : Selama 10 menit
3. UDP
 - a. UDP Flood Single IP : Selama 4 menit
 - b. UDP Flood Random IP : Selama 4 menit
 - c. Normal random delay : Selama 4 menit
 - d. Normal Video streaming : Selama 4 menit
4. ICMP

- a. ICMP Flood Single IP : Selama 4 menit
- b. ICMP Flood Random IP : Selama 4 menit
- c. ICMP Smurf Single IP (payload besar) : Selama 4 menit
- d. ICMP Smurf Random IP : Selama 4 menit
- e. Ping biasa : Selama 16 menit
- f. Normal ICMP Script: Selama 4 menit

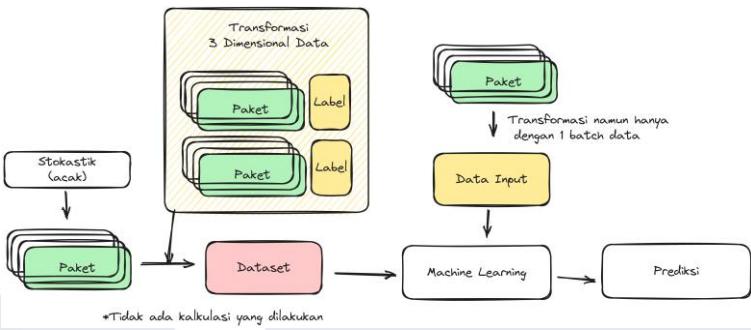
Semua skenario akan dijalankan dengan 4 *client* dan 1 *webserver*. Dataset akan berformat CSV yang nantinya akan digunakan untuk proses *training machine learning*. Dan untuk label penulis menentukan bahwa “0” adalah untuk label normal dan “1” adalah untuk label serang. Untuk arsitektur sistem yang akan dipakai diilustrasikan pada gambar 3.23.



Gambar 3.23 Arsitektur Pengambilan Dataset



Gambar 3.24 Cara Kerja Dataset Individu



Gambar 3.25 Cara Kerja Dataset Timeseries

Pada gambar 3.24 dan 3.25 dapat dilihat alur dan cara kerja *dataset* untuk metode *machine learning* individual dan *timeseries*. Dimana untuk *dataset* individual mendapatkan input data stokastik / acak berupa paket paket dalam interval 5 detik, setiap intervalnya akan dirangkum menjadi 1 informasi dengan kalkulasi standar deviasi dan sum. Tujuan dilakukan kalkulasi standar deviasi dan sum adalah untuk mewakili karakteristik dari serangan DDoS yaitu *distribution* dan *concentration*. Hasil kalkulasi ini masih bersifat stokastik atau acak secara nilai. Setelah itu akan didapatkan *features* atau keunikan tersendiri dari dataset melalui proses latih *machine learning*. Untuk dataset *timeseries* tidak akan ada kalkulasi data, input *machine learning* adalah input asli dari urutan paket paket yang masuk ke sistem. Hal ini dilakukan dengan harapan agar *deep learning* bidirectional LSTM dapat mencari sendiri *features* atau keunikan tersendiri dari dataset melalui proses latihnya yang sesuai dengan karakteristik serangan DDoS.

3.8 Perancangan Machine Learning

Pada tahap penelitian ini terdapat beberapa langkah yang dilakukan oleh penulis yaitu *pre-process* dan transformasi dataset, perancangan dan *training* model *machine learning*, dan evaluasi metrik *machine learning*.

1. Pre-process dan transformasi dataset

Pada tahapan ini dilakukan beberapa tahap yaitu *balancing* data menggunakan algoritma SMOTE, normalisasi data menggunakan StandardScaler, transformasi dataset, dan pembagian dataset *training* dan *testing*

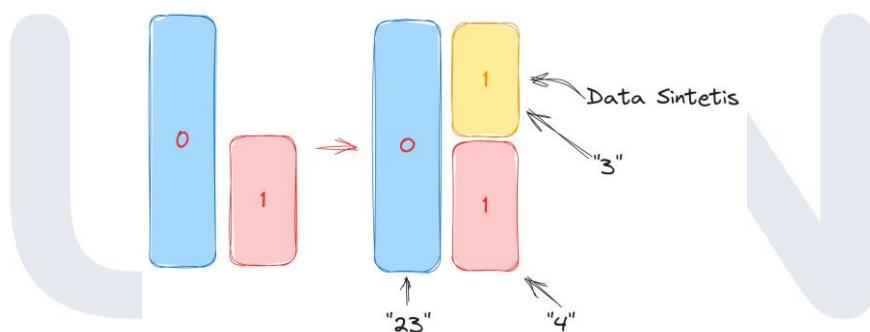
A. Balancing dataset

Jika dilihat pada skenario pengambilan dataset pada bagian 3.7, maka akan sulit jika mendapatkan dataset yang seimbang. Jika didapatkan ketidakseimbangan data, dan akan berpengaruh buruk pada model klasifikasi yang akan dibuat. Oleh karena itu diterapkanlah teknik SMOTE. Menurut lembar penelitian asli dari SMOTE, mereka menyarankan untuk melakukan *oversampling* dengan SMOTE lalu menerapkan juga *random undersample*. Hal ini karena jika data sintesis yang dibuat terlalu banyak juga memiliki pengaruh yang buruk, sehingga harus menurunkan data *majority* [30]. Oleh karena itu bisa dilakukan secara berurutan *oversampling* dan *undersampling*. Untuk potongan kode dari fungsi SMOTE dapat dilihat pada gambar 3.26.

```
over = SMOTE()
under = RandomUnderSampler()
steps = [('o', over), ('u', under)]
pipeline = Pipeline(steps=steps)
X, Y = pipeline.fit_resample(X, Y)
```

Gambar 3.26 Potongan Kode SMOTE untuk *Balancing Dataset*

Untuk ilustrasi dari proses SMOTE dapat dilihat pada ilustrasi gambar 3.27.



Gambar 3.27 Ilustrasi Proses Balancing SMOTE

Penjelasan pada gambar 3.27 adalah teknik SMOTE akan menghasilkan data sintetis (data buatan) berdasarkan label minoritas. Data sintetis tersebut merupakan nilai acak yang mendekati nilai label tersebut, hal ini untuk menghindari perubahan *features* (informasi penting) yang seharusnya dimiliki dataset aslinya. Proses

balancing akan dilakukan setelah membagi dataset *training* dan *testing*, hal ini dilakukan karena data sintetis yang dihasilkan tidak ingin berada di dataset testing yang menyebabkan validasi menjadi bias dan memiliki kesamaan karakteristik dengan data *training* yang akan menjadi salah satu faktor penyebab *overfitting*. Dan akan tidak akan dilakukan pada dataset *timeseries*.

B. Normalisasi dataset

Untuk mengoptimalkan rentang nilai antara 1 *feature* dengan *features* lain sehingga tidak ada *features* yang berat sebelah maka akan diimplementasikan normalisasi data atau sering disebut *features scaling*. Dimana akan menyamakan rentang nilai antar *features* dataset. Untuk melakukan data normalisasi dapat menggunakan library dari SKlearn yaitu StandarScaller. Setelah melakukan normalisasi penulis juga akan mengekspor *instance* dari StandarScaller agar dapat digunakan saat prediksi nanti. Untuk ilustrasi normalisasi dataset dapat dilihat pada tabel 3.1.

Tabel 3.1 Ilustrasi Normalisasi Dataset

	Feature 1	Feature 2	Feature 3
Dataset Asli	10	4324	10323213
Dataset Normalisasi	1.70623	0.0	29.142651

C. Transformasi dataset

Pada tahapan ini khusus dilakukan pada model *machine learning* dengan metode *timeseries*. Hal ini dilakukan karena model Bidirectional LSTM memerlukan 3-dimentional *matrix* yaitu *batch size*, *time step*, dan *input dimension*. Untuk rencana implementasinya akan sesuai dengan referensi penelitian terdahulu nomor 2.1.3. Dimana menggunakan *time window* untuk memilih beberapa paket lalu digabungkan menjadi satu dan menggunakan label paket terakhir.

D. Pembagian dataset

Penulis akan menggunakan perbandingan 80:20 untuk dataset *testing* dan *training* dan pembagian berdasarkan label sehingga perbandingan label tetap terjaga di dataset *training* dan *testing*. Hal ini dilakukan untuk menghindari *overfitting* sehingga model dapat dianalisis dengan baik.

2. Perancangan dan training model machine learning

Pada penelitian ini dibutuhkan *machine learning* untuk deteksi serangan DDoS dan identifikasi paket yang merupakan bagian dari DDoS. Oleh karena itu akan ada 2 pendekatan yaitu *atomic* (paket secara mandiri) untuk identifikasi dan *composite (stream)* untuk deteksi serangan. Karena untuk mendeteksi adanya serangan tidak bisa melihat satu persatu paket namun harus melihat kumpulan paket. Oleh karena itu setidaknya akan ada 3 kategori model *machine learning* yaitu deteksi DDoS secara individual, deteksi DDoS secara *timeseries*, dan identifikasi paket.

1. *Machine learning* deteksi DDOS secara individual

Akan dibandingkan 6 *machine learning* untuk klasifikasi yaitu SVM, linear regression, naïve bayes, KNN, Random Forest, dan Deep Neural Network dengan waktu rangkum 5 detik. Alasan untuk menggunakan ke 6 *machine learning* ini adalah karena merupakan algoritma *machine learning* yang populer dan penelitian bertujuan untuk mencari model *machine learning* yang cocok untuk klasifikasi serangan DDoS secara *realtime*. Penulis memiliki hipotesis bahwa setiap protokol memiliki *features*, karakteristik, dan penggunaan yang berbeda sehingga terjadi perbedaan karakteristik *machine learning* yang dibutuhkan. Penulis ingin membandingkan algoritma machine learning dan deep learning karena penulis ingin tahu apakah untuk menemukan features dari suatu serangan DDoS dibutuhkan neural network yang lebih kompleks namun dapat mengatahi masalah klasifikasi yang tidak bisa diselesaikan oleh pembuatan hyperplane machine learning. Untuk penjabaran spesifikasi model *machine learning* dan *deep learning* sebagai berikut :

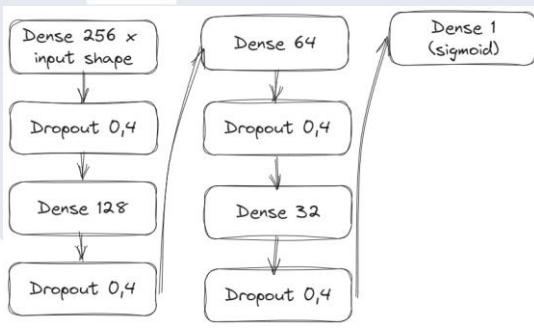
- *Machine learning*

Hyperparameter yang digunakan adalah sebagai berikut :

- A. SVM : menggunakan kernel linear dengan random state 0
- B. Linear regression : menggunakan random state 0
- C. KNN : menggunakan jumlah neighbors 5
- D. Random Forest : n_estimator 10, criterion “entropy”, random state 0

- Deep learning

Model *deep learning* yang digunakan adalah DNN yaitu dengan arsitektur yang diilustrasikan pada gambar 3.28.



Gambar 3.28 Arsitektur Machine Learning DNN

Layer dense *input* dan *hidden layer* memiliki aktivasi ReLU dan setiap *fully connected* layernya terdapat Dropout untuk mencegah terjadinya *overfitting* dengan membuang jumlah neuron secara acak. Dan untuk aktivasi output layer adalah Sigmoid. Sigmoid bagus untuk *binary classification*. Untuk loss function menggunakan binary crossentropy, optimizer adam, dan epoch sebanyak 100. Untuk mencegah terjadinya *overfitting* penulis mengimplementasikan EarlyStopping dengan *patience* 30 epoch (waktu tunggu untuk berhenti ketika tidak ada kenaikan performa) dan monitor *validation loss* dan ModelCheckpoint dengan monitor variabel *validation loss* sebagai *callback* fungsi *training*.

2. Machine learning deteksi DDoS secara timeseries

Untuk bagian ini, penulis menggunakan Bidirectional LSTM sebagai model *machine learning timeseries*. Menggunakan Bidirection LSTM karena penulis akan

mengkonfigurasi *time window* yang lebih besar sehingga diperlukan model LSTM dengan akurasi yang lebih besar. Penulis akan menggunakan *time window* sebesar 200. Untuk arsitektur dari model Bidirectional LSTM dapat dilihat pada gambar 3.29.

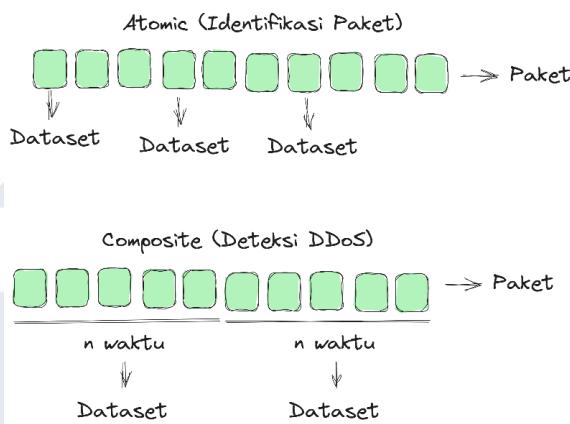


Gambar 3.29 Arsitektur Machine Learning Bidirectional LSTM

Hyperparameter yang diterapkan adalah loss function menggunakan binary crossentropy, optimizer adam, dan epoch sebanyak 30. Untuk mencegah terjadinya *overfitting* penulis mengimplementasikan EarlyStopping dengan *patience* 10 epoch (waktu tunggu untuk berhenti ketika tidak ada kenaikan performa) dan monitor *validation loss* dan ModelCheckpoint dengan monitor variabel *validation loss* sebagai *callback* fungsi *training*.

3. Machine learning identifikasi paket DDoS

Secara dasar, cara kerja *machine learning* metode atomic untuk identifikasi paket DDoS mirip dengan *machine learning* deteksi DDOS secara individual namun bedanya adalah menggunakan dataset dari *timeseries* untuk mengklasifikasikan setiap paket yang masuk apakah termasuk dari rangkaian paket serangan DDoS atau tidak. Yang melabelkan data input *machine learning* ini adalah hasil dari deteksi *machine learning* deteksi serangan DDoS (metode *composite*) yang sudah dibahas sebelumnya. Untuk lebih jelasnya dapat dilihat pada ilustrasi gambar 3.30.

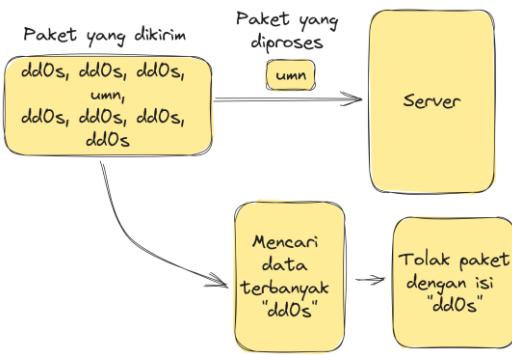


Gambar 3.30 Perbedaan *Atomic* dan *Composite*

Akan dibandingkan 5 model *machine learning*, dimana dataset yang digunakan adalah dataset *timeseries*. Namun model-model ini akan dilatih berulang kali secara *realtime* saat implementasinya karena data yang harus diklasifikasikan akan bersifat acak dan berbeda-beda setiap saat. Akan dicari algoritma *machine learning* yang memiliki keseimbangan antara akurasi yang tinggi dengan waktu latih dan prediksi yang paling kecil.

4. Paket Identifikasi Menggunakan Karakteristik Paket Terbanyak

Untuk membandingkan efektivitas dan efisiensi komputasi dari identifikasi paket yang termasuk serangan DDoS menggunakan *machine learning*, penulis memperkenalkan teknik untuk mendapatkan *signature* dengan algoritma sederhana untuk mengurangi beban komputasi. Yaitu ketika serangan DDoS terjadi, pasti paket mayoritas adalah paket dari kumpulan serangan DDoS itu sendiri dimana sesuai dengan karakteristik *concentration*, oleh karena itu bisa dicari nilai mayoritas dari *features comparator* yang sudah ditetapkan pada tahap sebelumnya. Jeleknya menggunakan metode ini adalah jika ke depannya penyerang menggunakan metode yang berbeda maka bisa saja *signature comparator* tidak berfungsi. Untuk ilustrasi dari metode ini dapat dilihat pada gambar 3.31.



Gambar 3.31 Ilustrasi Metode Identifikasi Karakteristik Paket Terbanyak

3. Evaluasi metrik *machine learning*

a. Akurasi

Akurasi untuk performa *machine learning* dibandingkan dengan dataset test dan validasi. Semakin tinggi akurasi semakin baik namun belum tentu mencerminkan situasi aslinya karena tergantung dengan cara pembuatan dataset test dan validasi.

$$Akurasi = \frac{TP + TN}{TP + FP + FN + TN}$$

b. Confusion matrix

Merupakan metrik klasifikasi yang sangat penting karena dapat melihat kelas atau kategori apa yang tidak seimbang, sehingga dapat diperbaiki untuk datasetnya, memperbaiki proses *pre-processing* data, atau hanya untuk bahan analisis model *machine learning*.

c. AUC – ROC

AUC-ROC hanya digunakan dalam menilai dataset *binary classification* dimana merupakan perbandingan antara *true positive* dengan *false positive*, AUC – ROC merupakan metrik yang baik untuk dataset yang *imbalance* dan *balance* dan mempermudah untuk membandingkan suatu model dengan model lainnya. Kita harus memilih nilai AUC-ROC yang paling besar karena miliki prediksi *true positive* dan *false positive* paling besar.

d. Presisi

Menggambarkan akurasi dengan label data yang diminta dengan hasil prediksi *machine learning*. Menggambarkan seberapa jauh / banyak kesalahan *false positive* yang berdampak pada kondisi normal yang dianggap sebagai serangan DDoS.

$$\text{Presisi} = \frac{TP}{TP + FP}$$

e. Recall

Keberhasilan model untuk menemukan kembali sebuah *features* atau data yang penting. Menggambarkan seberapa banyak data *false negative* yang berdampak pada data yang seharusnya merupakan serangan DDoS namun dikira normal.

$$\text{Recall} = \frac{TP}{TP + FN}$$

f. F-1 Score

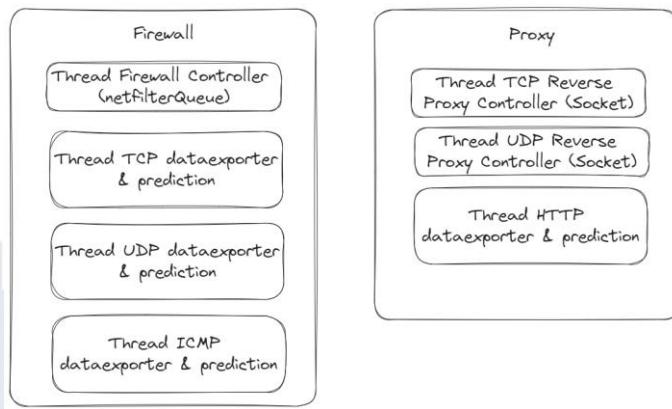
Perbandingan antara rata-rata presisi dengan recall

$$F1\ Score = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}}$$

3.9 Perancangan Sistem Firewall

Untuk melakukan prediksi *machine learning* secara *realtime* tanpa mempengaruhi alur koneksi yang ada maka perlu dibuat program dengan paradigma *multiprocessing*. Arsitektur umum dari sistem firewall dan proxy diilustrasikan pada gambar 3.32.

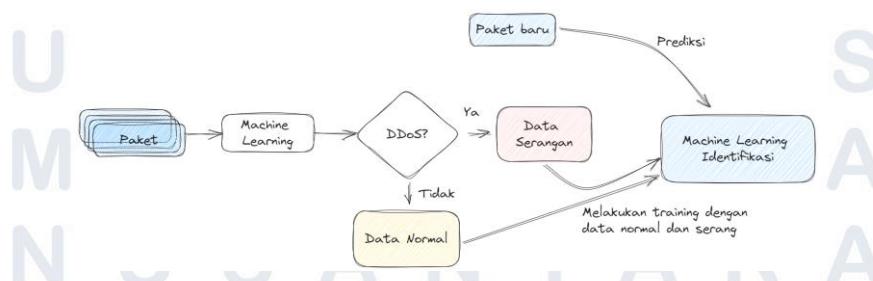




Gambar 3.32 Arsitektur Sistem Firewall dan Proxy

Terdapat *thread-thread* khusus yang difungsikan untuk menyalurkan komunikasi dan menjadi *listener*, namun juga dibuat *thread-thread* khusus yang difungsikan untuk mengolah data dan melakukan prediksi *machine learning*. Hal ini dimaksudkan untuk tidak mengganggu alur koneksi agar tidak menambah *latency* dari sistem proxy. Namun perlu diperhatikan dampak dari *deadlock*, sehingga penulis perlu memperhatikan penggunaan *global variable*, seperti mematikan status DDoS saat dilakukan latih *instance machine learning*.

Untuk deteksi secara keseluruhan terdapat 2 tahap *machine learning* yaitu deteksi serangan DDoS dan *training machine learning* untuk membuat *signature* dari paket serangan DDoS. Untuk membuat *signature* dengan *machine learning supervised* maka diperlukan dataset normal dan serang. Oleh karena itu setiap deteksi awal yang menentukan serangan DDoS atau bukan akan dimasukkan ke dalam kelompok data. *Machine learning* ke 2 akan di-*training* ulang ketika keadaan sebelumnya normal, jadi tidak ada *training* ulang ketika serangan DDoS masih dilancarkan. Ilustrasinya dapat dilihat pada gambar 3.33.



Gambar 3.33 Diagram 2 Fase Machine Learning

Untuk menguji tahapan ini, maka ditetapkan matrik pengujian yaitu lama waktu yang diperlukan untuk suatu konklusi didapatkan, dalam hal ini adalah dari awal waktu prediksi *machine learning* deteksi pertama, *training machine learning* kedua, dan waktu prediksi *machine learning* kedua.

Untuk protokol HTTP akan menggunakan identifikasi *IP Address*, oleh karena itu pada *firewall controller* harus terdapat observer untuk mengecek apakah *file firewall controller* berubah atau tidak. Hal ini dapat dicapai dengan menggunakan *library watchdog* observer pada bahasa pemrograman Python.

3.10 Perancangan Sistem Distribusi Data

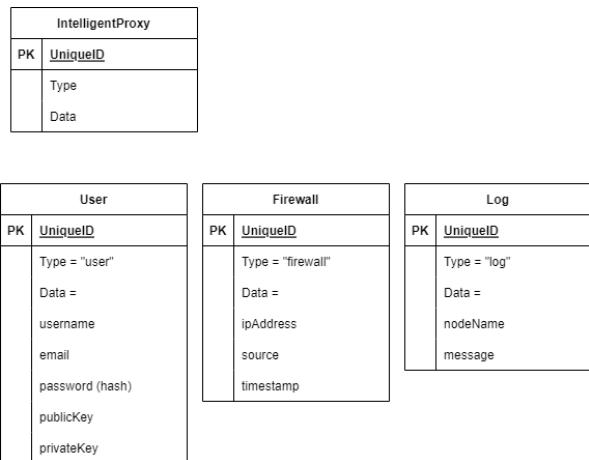
Perancangan sistem distribusi data terdiri dari beberapa proses yaitu perancangan *database* terdistribusi BigchainDB, perancangan sistem backend, dan perancangan sistem frontend, dan evaluasi dengan metrik pengujian sistem distribusi data. Untuk menambahkan informasi identitas penyerang atau dalam hal ini adalah *IP address*, hanya ada 2 alur yaitu ditambahkan oleh *proxy controller / forwarder* ke API backend dan admin melalui frontend.

3.10.1 Perancangan database terdistribusi BigchainDB

Pada dokumentasi resmi dari BigchainDB terdapat 3 cara untuk instalasi BigchainDB pada perangkat pribadi, yaitu dengan Docker namun hanya untuk *single node*, Ansible script, Kubernetes *script*, dan Manual instalasi. Karena penulis ingin membuat beberapa *node* sekaligus maka penulis memutuskan untuk memakai cara manual instalasi. Ada beberapa modul yang harus diinstal yaitu :

1. BigchainDB
2. MongoDB : Dasar database dari BigchainDB
3. Tendermint : Modul konsensus BigchainDB
4. Monit : Untuk sinkronisasi antara BigchainDB dan Tendermint
5. Konfigurasi Tendermint seperti genesis.json dan config.toml untuk membuat BigchainDB Network

Untuk database schema yang akan dibuat dapat dilihat pada gambar 3.34.



Gambar 3.34 Database Schema

IntelligentProxy adalah *assets* utama dari data yang akan disimpan, sebenarnya BigchainDB bisa menyimpan banyak *assets* namun pada versi terbaru BigchainDB terdapat bug yang menyebabkan tidak bisa dibuatnya beberapa *assets*. Oleh karena itu penulis menanggulanginya dengan membuat satu *assets* namun memiliki kolom “Type” sebagai pembeda antara fungsi data. Dengan solusi ini kemampuan untuk *immutability* masih didapatkan.

3.10.2 Perancangan Sistem Backend

Backend akan dibuat dengan *library* expressJS, BigchainDB ORM JS, DemocracyJS, CronjobJS, dan jsonwebtoken.

BigchainDB ORM adalah *library* untuk integrasi penggunaan *database* BigchainDB ke bahasa pemrograman Javascript, sehingga mempermudah untuk operasi CRAB. DemocracyJS untuk membuat sistem *pub/sub message* agar persebaran data semakin cepat sampai di entitas terakhir yaitu *firewall rule*. Cronjob digunakan untuk melakukan penjadwalan terhadap fungsi pembacaan database, sehingga *firewall node* akan selalu diperbarui setiap 1 menit. Untuk *authentikasi* antara *frontend* dan *backend* menggunakan JWT medianya.

Ada beberapa route yang dibuat pada database untuk melakukan operasi CRAB

:

Tabel 3.2 List API Endpoint Backend

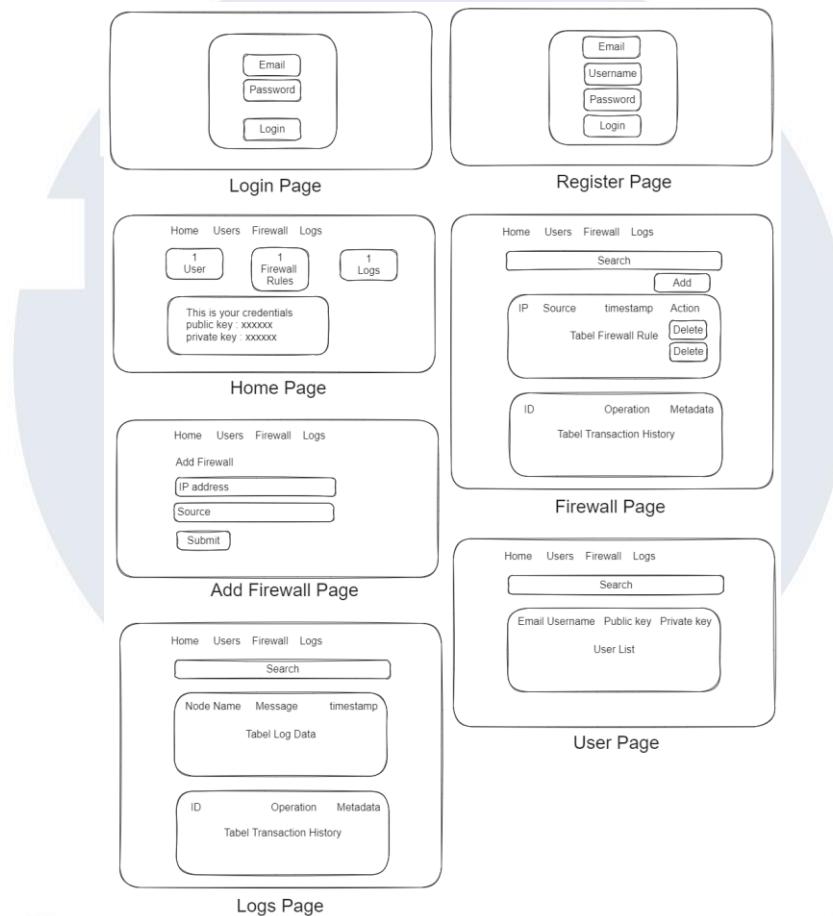
Function	Route	Endpoint	Database Operation	Payload	Pakai JWT ?
User	Membuat user baru (jika blockchain user belum ada maka dibuat)	POST /api/user	Create & Append	1.Keypair {publicKey & privateKey} 2.Username 3.Email 4.Password	Ya
	Menghapus user dengan username	POST /api/user/delete/{username}	Append	1.Keypair {publicKey & privateKey}	Ya
	Mengambil semua user	GET /api/user	Retrieve	-	Ya
	Menghapus blockchain user	Delete /api/user	Burn	1.Keypair {publicKey & privateKey}	Ya
	Login	POST /api/auth/signin	Retrieve	1.Username 2.Email	-
	Signup	POST /api/auth/signup	Create & Append	1.Username 2.Email 3.Password	-
Firewall	Membuat firewall rule baru (jika blockchain	POST /api/firewall	Create & Append	1.Keypair {publicKey & privateKey}	Ya

	firewall belum ada maka dibuat)			2.IP Address 3.Source	
	Menghapus salah satu firewall rule dengan IP Address	POST /api/firewall /delete/{ipAddress}	Append	1.Keypair {publicKey & privateKey}	Ya
	Mengambil semua firewall rule	GET /api/firewall	Retrieve	-	Ya
	Menghapus blockchain firewall	Delete /api/firewall	Burn	1.Keypair {publicKey & privateKey}	Ya
Log	Membuat Log	POST /api/log	Create & Append	1.Keypair {publicKey & privateKey} 2.Node Name 3.Message	Ya
	Mengambil semua log	GET /api/log	Retrieve	-	Ya

3.10.3 Perancangan Sistem Frontend

Sistem frontend akan dibangun menggunakan framework React, bootstrap, dan axios. Frontend akan tetap berbentuk *client side rendering* untuk meringankan beban sistem ketika *website* diakses. Ada 5 halaman dasar yang akan dibuat yaitu

halaman authorisasi, firewall, logs, user, dan home. Dimana untuk rancangan *layout* dari frontend dapat dilihat pada gambar 3.35.

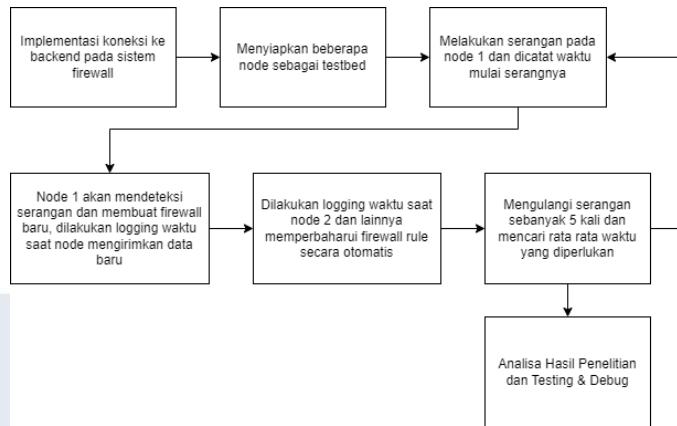


Gambar 3.35 Rancangan Frontend

3.10.4 Analisa Kecepatan Persebaran Data

Penelitian ini akan menganalisis kecepatan rata-rata persebaran data pada sistem distribusi data yang telah dibuat. Hal ini bertujuan untuk mendapatkan evaluasi performa pada sistem yang telah dibuat, seperti memastikan koneksi antar modul *database*, *backend*, *frontend*, dan *firewall* berjalan, efektivitas penggunaan *database* BigchainDB, pengaruh sistem konsensus Democracy JS, dan optimalisasi sistem persebaran data ke depannya.

Penelitian ini memiliki alur penelitian yang ditampilkan pada gambar 3.36.



Gambar 3.36 Alur Evaluasi Sistem Distribusi Data

Metrik pengujian pada penelitian ini adalah :

1. Rata-rata response time BigchainDB

Data penelitian didapatkan dengan melakukan *request API* dengan program Postman.

2. Uji penetrasi BigchainDB

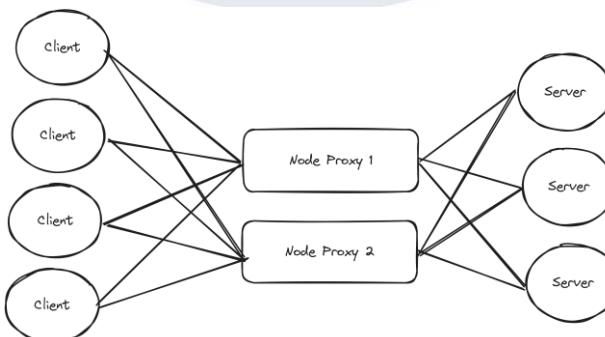
Penelitian ini memiliki tujuan untuk menguji immutabilitas data pada database BigchainDB. Langkah pada penelitian ini adalah melakukan insert data dan update data dengan melakukan *request* ke backend namun dengan *public key* dan *private key* yang salah. Akan diobservasi perilaku dari sistem backend dan *database* BigchainDB menggunakan bantuan dari informasi yang didapatkan dari sistem frontend.

3. Rata rata waktu yang diperlukan untuk informasi data sampai ke semua node

Penelitian dilakukan dengan melakukan *logging* pada sistem yang dibuat, yaitu pada modul *firewall controller* yang terdapat *watchdog observer* sehingga dapat dilihat waktu saat *firewall rule* terganti sejak node lain mengirimkan *firewall rule* terbaru.

3.11 Perancangan Deployment Infratruktur Sistem

Sistem yang dibangun akan mendukung untuk diimplementasikan secara terdistribusi maupun individual (*standalone*) hal ini karena fleksibilitas dari BigchainDB. Dibutuhkan *software* seperti sistem operasi Linux yang mendukung Iptables agar library netfilterQueue dapat bekerja, Python, NodeJS, Tendermint, MongoDB, dan lain lain. Untuk pengembangan lebih lanjut, *software* ini dapat diinstalasi menggunakan Docker, namun untuk penelitian ini penulis hanya melakukan instalasi tanpa menggunakan Docker. Lalu untuk mengurangi *single point of failure* maka diperlukan *mirroring* proxy server, karena sistem yang dibangun akan menggunakan sistem terdistribusi yang fleksibel dan terdapat fitur *load balancing* maka kebutuhan ini dapat terpenuhi. Untuk ilustrasi implementasinya dapat dilihat pada gambar 3.37. Node proxy 1 dan 2 diimplementasikan pada 1 sistem yang sama dan di-*mirror* agar jika ada 1 node proxy yang gagal maka masih ada sistem proxy yang lainnya. Dan node proxy 1 dan 2 masih bisa mendistribusikan informasi identitas penyerang dengan sistem terdistribusinya.



Gambar 3.37 Ilustrasi Deployment Sistem Proxy Mitigasi DDoS

3.12 Perbandingan dengan IDS / IPS Snort

Pada penelitian ini dilakukan beberapa tahapan yaitu instalasi aplikasi IPS / IDS Snort, melakukan konfigurasi dasar pada aplikasi Snort, uji aplikasi Snort dengan skenario yang sama dengan sistem yang dibangun, dan dilakukan analisis terhadap perbedaan cara kerja dari IPS / IDS Snort dengan sistem yang dibuat.

Untuk konfigurasi Snort Rule, penulis menetapkan pengaturan sebagai berikut untuk *testbed* penelitian:

1. TCP

- a. alert tcp any any -> \$HOME_NET any (flags: S; msg:"Possible TCP SYN Flood DDoS"; flow: stateless; detection_filter: track by_src, count 100, seconds 5; sid: 10001)
- b. alert tcp any any -> \$HOME_NET any (flags: S; msg:"Possible TCP SYN Flood DDoS"; flow: stateless; detection_filter: track by_dst, count 100, seconds 5; sid: 10001)

2. UDP

- a. alert udp any any -> \$HOME_NET any (msg:"Possible UDP Flood DDoS"; flow: stateless; detection_filter: track_by_src, count 200, seconds 5; sid: 10003)
- b. alert udp any any -> \$HOME_NET any (msg:"Possible UDP Flood DDoS"; flow: stateless; detection_filter: track_by_dst, count 200, seconds 5; sid: 10003)

3. ICMP

- a. alert icmp any any -> \$HOME_NET any (msg:"Possible ICMP Flood DDoS"; detection_filter: track_by_src, count 200, seconds 5; sid: 10004)
- b. alert icmp any any -> \$HOME_NET any (msg:"Possible ICMP Flood DDoS"; detection_filter: track_by_dst, count 200, seconds 5; sid: 10004)

4. HTTP

- a. alert tcp any any -> \$HOME_NET any (msg:"GET Request flood attempt"; content:"HTTP"; detection_filter:track_by_src, count 200, seconds 10; sid: 10005; rev:005;)

Penjelasan dari Snort rule yang dipakai di atas adalah suatu threshold akan memakai 2 identitas sebagai penghitungnya yaitu track_by_src untuk mengatasi masalah random ip yang memiliki IP sumber yang berbeda beda dan track_by_dst untuk mengatasi serangan DDoS yang hanya pada 1 IP sumber, pada setiap identitasnya hanya diberi kuota 200 paket setiap 5 detiknya.

BAB IV

IMPLEMENTASI DAN PENGUJIAN SISTEM

4.1 Spesifikasi Sistem dan Testbed

Penelitian ini menggunakan laptop penulis untuk melakukan pembuatan *virtual machine*, *pre-process* dan *training machine learning*, dan pengujian sistem yang dibangun. Spesifikasi perangkat laptop yang digunakan adalah :

- CPU : Intel Core I5-12500H @ 2.50Ghz (16 Thread (4P core & 8E core))
- GPU : Nvidia GeForce RTX 3050 Laptop GPU 4GB VRAM
- RAM : 32GB DDR4 3200Mhz
- Storage : SSD 512GB PCIe NVME Gen 4
- OS : Windows 11 Home
- Software yang digunakan :
 - VMWare Workstation Pro 11
 - Tensorflow & Keras dengan GPU *driver* (*CUDA core support*)
 - Python 3.10 dan PIP
 - NodeJS 18 dan NPM 9

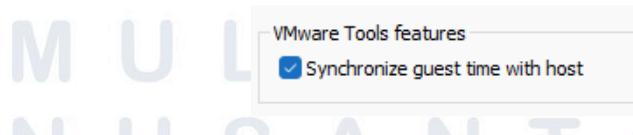
Sedangkan untuk *testbed* menggunakan *virtual machine* yang diinstall pada perangkat laptop penulis, dengan spesifikasi sebagai berikut :

1. *Virtual Machine* Sistem Proxy Mitigasi DDoS berbasis *machine learning* dan blockchain dan IDS Snort
 - a. CPU : 8 core
 - b. RAM : 8GB
 - c. Storage : 40 GB
 - d. OS : Ubuntu Desktop 22.04.2 LTS
 - e. Software yang digunakan
 - i. Python 3.10 dan PIP (Python 3.6 untuk BigchainDB)
 - ii. NetfilterQueue
 - iii. Scikit Learn
 - iv. Tensorflow CPU

- v. BigChainDB 2.2.2
 - vi. MongoDB 4.4
 - vii. Tendermint v0.31.5
 - viii. NodeJS 20 dan NPM 9
 - ix. BigChainDB ORM JS
2. *Virtual Machine webserver* dan *client* untuk simulasi serangan DDoS
- a. CPU : 2 Core
 - b. RAM : 2 GB
 - c. Storage : 20 GB
 - d. OS : Debian server 11
 - e. Software :
 - i. Python 3.10 dan PIP
 - ii. Node JS 20 dan NPM 9
 - iii. Flask
 - iv. Hping3

Sistem *virtual machine* yang digunakan pada penelitian ini menjadi suatu batasan penelitian, karena *machine learning* yang menggunakan framework Tensorflow / Keras secara *native* komputasi menggunakan set instruksi AVX2 dan dapat menggunakan akselerasi GPU. Namun untuk *virtual machine* tidak dapat menggunakan teknologi tersebut, sehingga dapat menimbulkan perbedaan hasil penelitian.

Untuk setiap *virtual machine*, dikonfigurasikan menggunakan jaringan NAT dengan subnet 24. Dan diaktifkan fitur dari VMware yaitu sinkronisasi waktu dari guest OS dengan host OS, hal ini diperlukan karena ada beberapa uji penelitian yang membandingkan waktu di antara beberapa *virtual machine* sekaligus. Oleh karena itu diaktifkan fitur ini, ditampilkan opsi dari fitur VMware pada gambar 4.1.



Gambar 4.1 VMware Synchronize time

Webserver yang dipakai sebagai objek pengujian dibuat menggunakan *framework* Flask dengan bahasa pemrograman Python. *Webserver* berisi 7 *endpoint* dengan HTML page yang berbeda beda. Hal ini ditujukan untuk menyimulasikan aktivitas akses internet yang dinamis. Web server ini akan berjalan pada port 5000 secara *default* namun dapat menggantinya dengan menambahkan argumen saat menjalankan *webserver*-nya. Pada gambar 4.2 merupakan potongan kode dari *webserver* yang dipakai sebagai objek penelitian :

```
from flask import Flask, render_template, jsonify, request
import argparse

app = Flask(__name__)

@app.route('/')
def hello_world():
    return render_template('index.html')

@app.route('/login')
def login_page():
    return render_template('page3.html')

@app.route('/home')
def landing_page():
    return render_template('page2.html')

@app.route('/register')
def register_page():
    return render_template('page1.html')

@app.route('/documentation')
def documentation_page():
    return render_template('page5.html')

@app.route('/singlepageapplication')
def single_page():
    return render_template('page4.html')

@app.route('/message', methods=["GET"])
def message():
    posted_data = request.get_json()
    name = posted_data['name']
    return jsonify("Hope you are having a good time " + name + "!!!")

@app.route('/api')
def apitest():
    return 'API Success'

@app.route('/name', methods=["POST"])
def setName():
    if request.method=='POST':
```

Gambar 4.2 Potongan Kode Webserver

Sebagai lingkungan uji coba, penulis menyiapkan 4 *virtual machine* client, 1 *virtual machine webserver*, dan 2 *virtual machine* proxy.

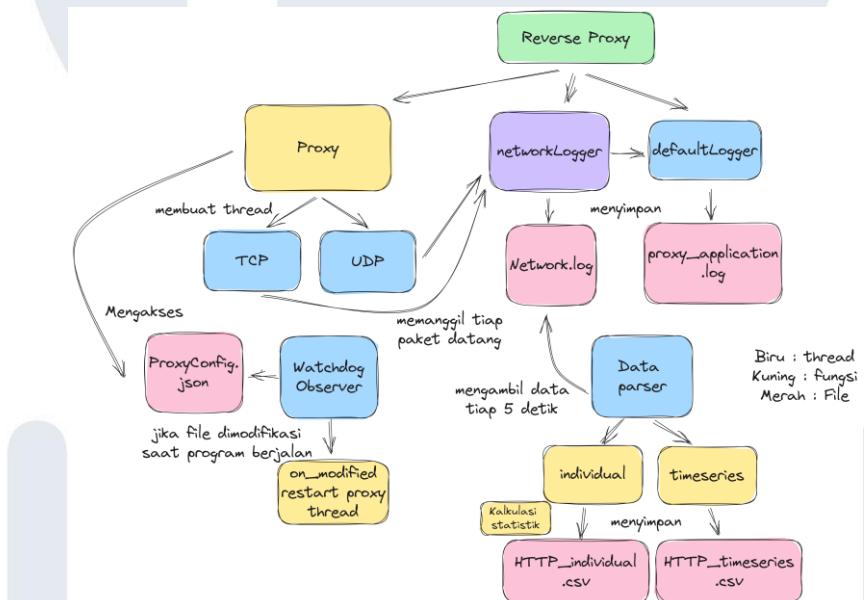
4.2 Implementasi Sistem

Implementasi sistem terdiri dari beberapa langkah sesuai dengan metode penelitian yaitu pembangunan sistem proxy, *machine learning*, sistem *firewall*, sistem distribusi data, dan IPS / IDS Snort sebagian sistem banding.

4.2.1 Pembangunan Sistem Proxy

1. Multithreaded Reverse Proxy

Untuk membangun *Multithreaded* server diperlukan beberapa *library* yaitu Socket untuk membuat koneksi server dan client, threading untuk membuat *multithread* sebuah *process*, Json untuk melakukan operasi dengan JSON, Watchdog Observer untuk membuat suatu *instance observer* yang berfungsi mendeteksi ketika suatu file dimodifikasi, logging untuk membuat *thread safe logging*, dan *library* umum seperti os, sys, numpy, pandas, dan lain lain. Pada proxy server ini, penulis juga mengimplementasikan *load balancing* dengan round robin dan juga sistem *caching* dengan memory store. Untuk arsitekturnya dapat dilihat pada gambar 4.3.



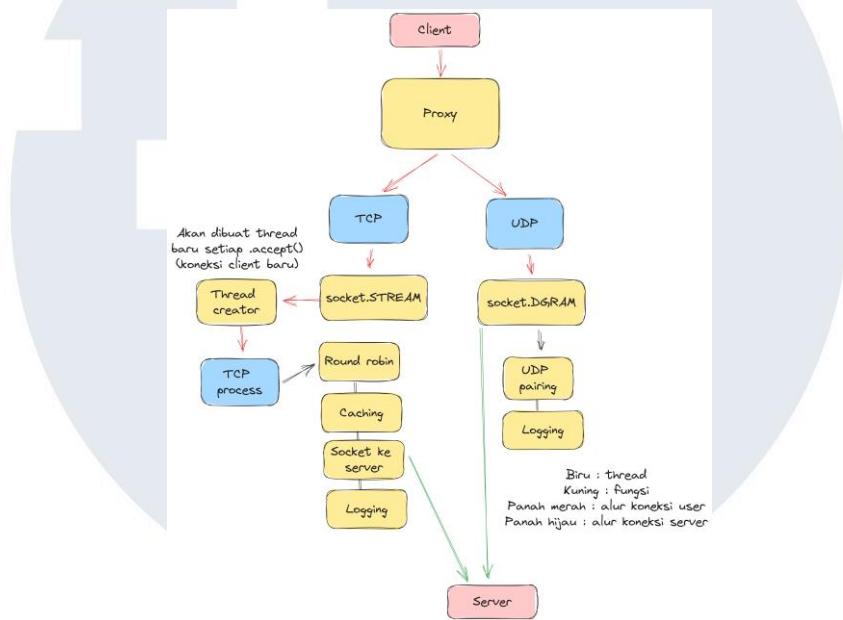
Gambar 4.3 Arsitektur Reverse Proxy

Sistem proxy memiliki 5 *thread* utama yaitu proxy TCP dan UDP, watchdog *observer*, *logger*, dan data parser.

a. Proxy

Modul ini adalah inti dari reverse proxy, dimana bertugas untuk menerima paket dari client dan menerukannya ke server dan sebaliknya. Reverse proxy TCP dapat membuat thread baru secara otomatis saat beberapa client mengakses proxy secara bersamaan. Untuk reverse proxy UDP hanya dibuat 1 thread karena UDP

merupakan koneksi stateless, sehingga bisa saja paket berikutnya yang datang bukan sumber yang sama dengan paket sebelumnya. Sehingga untuk reverse proxy UDP diperlukan suatu fungsi UDP pairing untuk pencatatan alamat sumber sesuai dengan permintaannya. Untuk Ilustrasi arsitektur modul proxy dapat dilihat pada gambar 4.4.



Gambar 4.4 Arsitektur Modul Proxy

Ada beberapa tipe socket yaitu socket.SOCK_STREAM dan socket.SOCK_DGRAM. Keduanya adalah variabel yang digunakan dalam *library* Socket Python untuk menentukan jenis protokol komunikasi yang akan digunakan untuk komunikasi jaringan komputer. Variabel SOCK_STREAM digunakan untuk membuat socket berbasis TCP. Di sisi lain variabel SOCK_DGRAM digunakan untuk membuat socket berbasis UDP. Lalu terdapat variabel AF_INET digunakan untuk menentukan bahwa soket akan menggunakan protokol IPv4 untuk berkomunikasi melalui jaringan. Ketika kita bekerja dengan *library* ini kita akan menggunakan paradigma *object oriented*, karena Socket dan paket yang diterima akan berbentuk *object*.

Kita harus menentukan nama dari *thread* yang akan dibuat, nama *thread* diharuskan unik sehingga penulis menggunakan iterasi dari jumlah koneksi saat itu. Jadi jika *thread* baru terbuat akan ditambahkan 1 dan jika *thread* sudah selesai maka

akan dikurangi 1. Dalam pembuatan *thread* kita juga harus menentukan fungsi yang akan dijalankan dan juga variabel yang akan dikirim seperti objek dari socket itu sendiri dan *client address*.

```
def run(self):
    # Waiting for TCP connection
    while 1:
        # Accept incoming connection from iptables
        (clientSocket, client_address) = self.serverSocket.accept()

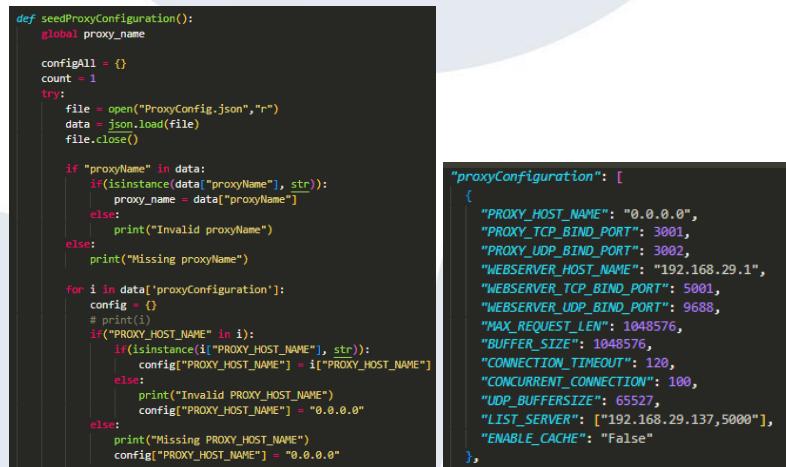
        # Handling new client connection with new thread
        th = threading.Thread(
            name=self._getClientName(),
            target=self.proxy_tcp_thread,
            args=(clientSocket, client_address, self.config),
        )

        # Start TCP thread
        th.start()

    # Close socket if program close
    self.serverSocket.close()
```

Gambar 4.5 Potongan Pembuatan Thread TCP

Untuk mengatur semua konfigurasi pada proxy server akan digunakan *file JSON*. Akan disimpan seperti IP address dan *port reverse proxy*, tujuan web server, dan lain lain.



```
def seedProxyConfiguration():
    global proxy_name

    configAll = {}
    count = 1
    try:
        file = open("ProxyConfig.json", "r")
        data = json.load(file)
        file.close()

        if "proxyName" in data:
            if(isinstance(data["proxyName"], str)):
                proxy_name = data["proxyName"]
            else:
                print("Invalid proxyName")
        else:
            print("Missing proxyName")

        for i in data["proxyConfiguration"]:
            config = {}
            # print(i)
            if("PROXY_HOST_NAME" in i):
                if(isinstance(i["PROXY_HOST_NAME"], str)):
                    config["PROXY_HOST_NAME"] = i["PROXY_HOST_NAME"]
                else:
                    print("Invalid PROXY_HOST_NAME")
                    config["PROXY_HOST_NAME"] = "0.0.0.0"
            else:
                print("Missing PROXY_HOST_NAME")
                config["PROXY_HOST_NAME"] = "0.0.0.0"

            configAll[count] = config
            count += 1
    except:
        print("Error reading file")
```

```
"proxyConfiguration": [
    {
        "PROXY_HOST_NAME": "0.0.0.0",
        "PROXY_TCP_BIND_PORT": 3001,
        "PROXY_UDP_BIND_PORT": 3002,
        "WEBSERVER_HOST_NAME": "192.168.29.1",
        "WEBSERVER_TCP_BIND_PORT": 5001,
        "WEBSERVER_UDP_BIND_PORT": 9688,
        "MAX_REQUEST_LEN": 1048576,
        "BUFFER_SIZE": 1048576,
        "CONNECTION_TIMEOUT": 120,
        "CONCURRENT_CONNECTION": 100,
        "UDP_BUFFERSIZE": 65527,
        "LIST_SERVER": ["192.168.29.137,5000"],
        "ENABLE_CACHE": "False"
    },
    {
        "PROXY_HOST_NAME": "0.0.0.0",
        "PROXY_TCP_BIND_PORT": 3003,
        "PROXY_UDP_BIND_PORT": 3004,
        "WEBSERVER_HOST_NAME": "192.168.29.2",
        "WEBSERVER_TCP_BIND_PORT": 5002,
        "WEBSERVER_UDP_BIND_PORT": 9689,
        "MAX_REQUEST_LEN": 1048576,
        "BUFFER_SIZE": 1048576,
        "CONNECTION_TIMEOUT": 120,
        "CONCURRENT_CONNECTION": 100,
        "UDP_BUFFERSIZE": 65527,
        "LIST_SERVER": ["192.168.29.137,5000"],
        "ENABLE_CACHE": "False"
    }
],
```

Gambar 4.6 Potongan Kode dan Contoh File Konfigurasi Proxy Server

Dengan menggunakan *library logging* dari Python sudah *support* untuk melakukan *thread safe logging*. Dilakukan konfigurasi seperti destinasi *file log* dan format yang akan dihasilkan oleh *logger*. Akan dibuat *logger* khusus untuk menampung informasi paket yang datang. Potongan kode saat pembuatan *thread safe logging* ditampilkan pada gambar 4.7.

```

# Create a logger
networklogger = logging.getLogger('networklogger')
networklogger.setLevel(logging.INFO)
networkloggerhandler = logging.FileHandler('network.log')
networkloggerformatter = logging.Formatter('%(created)f,%(thread)d,%(message)s')
networkloggerhandler.setFormatter(networkloggerformatter)
# Set filter to log only INFO lines
networkloggerhandler.addFilter(LoggerFilter(logging.INFO))
networklogger.addHandler(networkloggerhandler)

# Initialize logging
logging.basicConfig(filename='proxy_application.log', encoding='utf-8', level=logging.DEBUG,
                    format='%(created)f,%(thread)d,%(message)s')

# Data logging
if(clientSocket.gettimeout() != None):
    socket_timeout = clientSocket.gettimeout()
else:
    socket_timeout = 0
networklogger.info(F'{"TCP"},{str((tcp_end_connection - tcp_start_connection)*1000)},
```

Gambar 4.7 Potongan Kode Thread Safe Logging

Thread data parser digunakan untuk melakukan rekap data dan akan tidur selama 5 detik. Potongan dari kode yang dibuat ditampilkan pada gambar 4.8. Untuk data individual akan mengambil data dari file network.log yang berasal dari fungsi *logger* dan untuk *timeseries* akan mengambil data dari *array* data sementara. Setiap setelah mengambil data dari network.log, *file* tersebut akan dikosongkan untuk iterasi berikutnya (*flush logfile* dan *flush data*). Khusus untuk protokol HTTP sudah dikategorikan berdasarkan IP Address setiap datasetnya, hal ini agar hasilnya adalah IP address tersebut merupakan penyerang atau tidak.

```

class DataParser(Thread):
    def __init__(self):
        super(DataParser, self).__init__()

        # Data parser configuration
        self.sleep_time = 5
        self.file_name = "network.log"
        self.data_count = 10

    def run(self):
        global http_model
        global http_scaler

        http_timeseries = {}

        while True:
            # Array for save raw data from file
            raw_datas = []

            # Read logfile
            network_log_file = open(self.file_name, 'r')
            network_lines = network_log_file.readlines()
            if(len(network_lines) != 0): ...

            # Flush logfile
            with open(self.file_name, 'w'):
                pass

            # Flush data
            raw_datas = []

            # Sleep n second
            time.sleep(self.sleep_time)
```

Gambar 4.8 Potongan Kode Data Parser

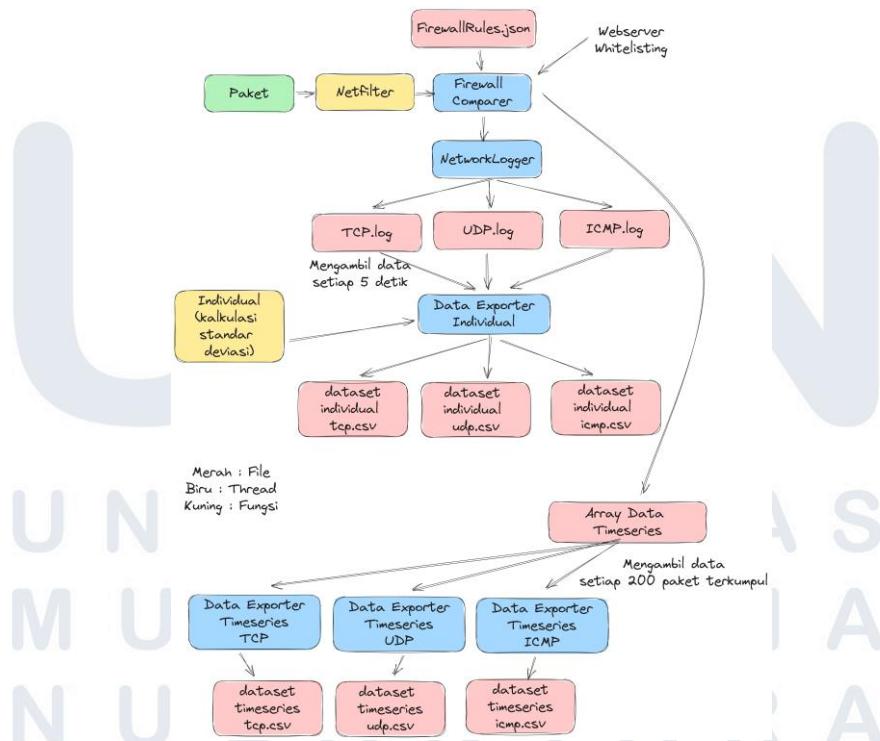
Pada gambar 4.9 merupakan tampilan dari aplikasi proxy ketika berjalan :

```
Node1-proxy is ready, creating proxy thread
creating proxy: proxy1
  type id  privPolyId listenport      ip_addr
  0   1   0   RoundRobin      5000  192.168.29.137
{'RoundRobin': <itertools.cycle object at 0x7fac37a6fbc0>}
creating proxy: proxy2
  type id  privPolyId listenport      ip_addr
  0   1   0   RoundRobin      5000  192.168.29.138
```

Gambar 4.9 Tampilan Proxy Server

2. Firewall Server

Untuk membuat sistem *firewall* server digunakan netfilterQueue sebagai *library* utama. Namun sebelum dipakai, penulis harus menambahkan konfigurasi ke IP tables agar menggunakan netfilterQueue sebagai *callback*-nya yaitu dengan menjalankan perintah ini “`sudo iptables -I INPUT -d 192.168.29.0/24 -j NFQUEUE --queue-num 1`”. 192.168.29.0/24 dapat diganti menjadi *subnet* lain pada jaringan lain dan INPUT berarti hanya mengantrekan paket dari luar ke dalam, namun paket dari dalam ke luar tidak diantrekan. Untuk membuat *firewall* server diperlukan netfilterQueue, Json untuk menyimpan *firewall rule*, watchdog *observer*, dan *logging*. Pada gambar 4.10 dapat dilihat ilustrasi dari arsitektur firewall server.



Gambar 4.10 Arsitektur Firewall Server

Ketika ada paket masuk netfilterQueue, maka akan mengirimkan suatu objek, pada object tersebut ada banyak informasi yang didapatkan misalnya saja IP address source. Dengan informasi tersebut kita bisa mencocokkan ke suatu list *IP Address*, untuk cuplikan kode dari fungsi filter ini dapat ditampilkan pada gambar 4.11.

```
def firewall(pkt):
    # parse packet data from incoming connection
    sca = IP(pkt.get_payload())

    # Webserver whitelist
    if sca.src in ListOfWebserverIpAddress:
        pkt.accept()
        return

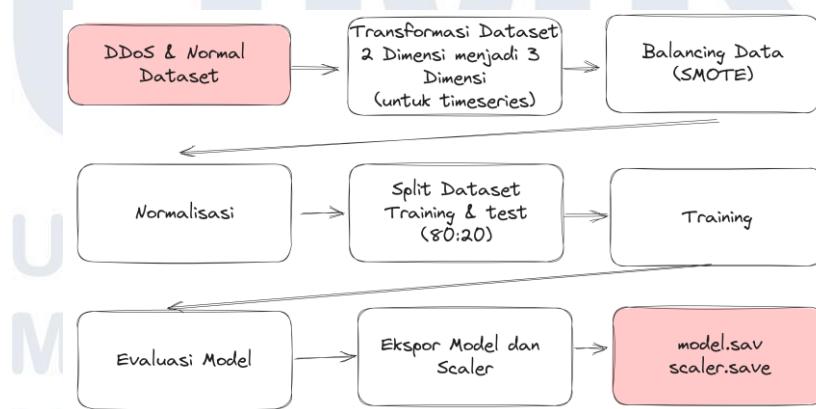
    # IP address firewall
    if sca.src in ListOfBannedIpAddress:
        print(sca.src, "is a incoming IP address that is banned by the firewall.")
        logging.info(f'{sca.src} is a incoming IP address that is banned by the firewall.')
        pkt.drop()
        return
```

Gambar 4.11 Potongan Kode fungsi Filter Firewall

Pada *firewall* terdapat beberapa *thread* yaitu data eksporter individual dan data eksporter *timeseries*. Seperti pada proxy server data individual akan disimpan sementara pada network.log dan untuk data *timeseries* akan disimpan pada struktur data *array* sementara.

4.2.2 Pembangunan Machine Learning

Pada tahap pembangunan model machine learning terdapat beberapa tahapan yang dilakukan oleh penulis yang dapat dilihat pada gambar 4.12.



Gambar 4.12 Alur Kerja Pembangunan Model Machine Learning

A. Pengambilan Dataset

Setelah penulis melakukan skenario percobaan yang dijelaskan pada bab perencanaan dan analisa, dataset yang terkumpul dapat dilihat pada tabel 4.1, dan beberapa bentuk dataset dapat dilihat pada lembar lampiran. Untuk data serang dilabel dengan “1” dan data normal dilabel dengan “0”.

Tabel 4.1. Hasil Dataset yang Terkumpul

Protokol	Individual			Timeseries		
	Serang	Normal	Total	Serang	Normal	Total
TCP	318	615	933	90.273	96.512	186.785
UDP	307	490	797	113.620	31.688	145.308
ICMP	521	733	1.254	62.676	8.299	70.975
HTTP	348	567	915	9.370	1.094	10.464

Jika dilihat pada tabel 4.1, terdapat ketidakseimbangan dataset antara label serang dan normal. Hasil ini didapatkan karena perbedaan jumlah skenario saat pengambilan dataset, dan juga pada dataset *timeseries* mencerminkan jika serangan DDoS akan mengirim data lebih banyak dari pada keadaan normalnya karena karakteristik dasarnya. Jika dibiarkan hal ini akan mempengaruhi kualitas hasil model latih *machine learning* nantinya, oleh karena itu diperlukan teknik *balancing* data untuk menyeimbangkan label serang dan normal.

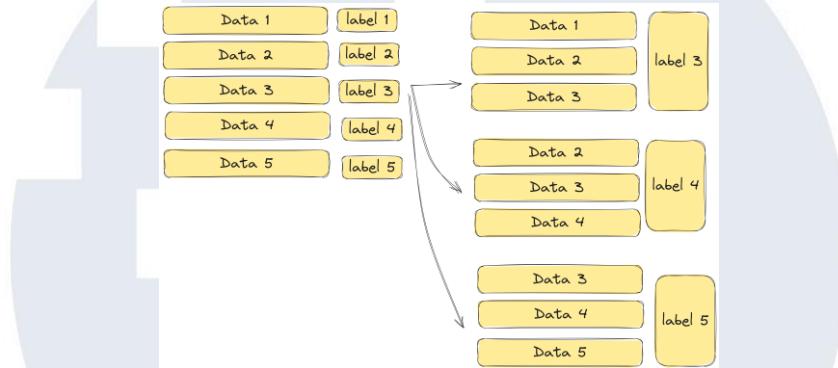
B. Proses transformasi data

Untuk TCP perlu dilakukan konversi data *features flags* yang berisi *string* alfabet menjadi nilai numerik. Penulis melakukan transformasi dari *string* menjadi nilai *bytes* atau nilai *ascii* dari alfabet tersebut. Untuk contoh kode dalam melakukan transformasi data dapat dilihat pada gambar 4.13.

```
def StringToBytes(data):
    sum = 0
    arrbytes = bytes(data, 'utf-8')
    for i in arrbytes:
        sum = sum + i
    return(sum)
```

Gambar 4.13 Potongan Kode Perubah String menjadi Numerik Bytes

Khusus untuk machine learning Bidirectional LSTM perlu dilakukan transformasi data menjadi 3 dimensional matriks yaitu *batch size*, *timestep*, dan *input dimension*. Time window yang dipakai ada 200 paket. Sehingga total dari dataset akan menjadi ($n - 200$). Untuk ilustrasinya dapat dilihat pada gambar 4.14.



Gambar 4.14 Ilustrasi Proses Transformasi Dataset B-LSTM

Kode yang digunakan untuk melakukan transformasi dapat dilihat pada gambar 4.15. Terlihat bahwa data sebelumnya berbentuk 2 dimensi (125352, 4) menjadi 3 dimensi (125152, 200, 4).

```
print(np.shape(X))
print(np.shape(Y))

(125352, 4)
(125352,)

features = len(X[0])
samples = X.shape[0]
train_len = 200
input_len = samples - train_len
I = np.zeros((samples - train_len, train_len, features))

for i in range(input_len):
    temp = np.zeros((train_len, features))
    for j in range(i, i + train_len - 1):
        temp[j-i] = X[j]
    I[i] = temp

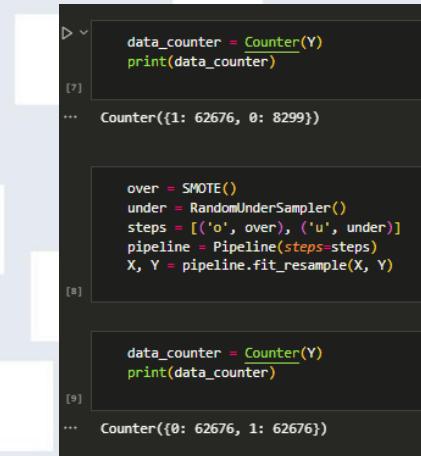
print(I.shape)

(125152, 200, 4)
```

Gambar 4.15 Potongan Kode Transformasi Dataset menjadi 3 Dimensional Matriks

C. Proses *balancing* Data

Proses *balancing* data dilakukan dengan menggunakan teknik SMOTE, untuk potongan kode dapat dilihat pada gambar 4.16.



```
data_counter = Counter(Y)
print(data_counter)
...
Counter({1: 62676, 0: 8299})

over = SMOTE()
under = RandomUnderSampler()
steps = [('o', over), ('u', under)]
pipeline = Pipeline(steps=steps)
X, Y = pipeline.fit_resample(X, Y)

data_counter = Counter(Y)
print(data_counter)
...
Counter({0: 62676, 1: 62676})
```

Gambar 4.16 Potongan Kode Teknik SMOTE dan Perbedaan Jumlah Dataset

Ditampilkan hasil dari proses *balancing* data pada dataset training dapat ditampilkan pada tabel 4.2.

Tabel 4.2 Perbandingan Dataset Sebelum dan Sesudah Balancing

Keterangan	Sebelum <i>balancing</i>	Sesudah <i>balancing</i>
Label 0	615	615
Label 1	315	615
Contoh data pada label 1	[0.8755519298449245, 0.0, 28.0, 654477784.9398003, 607147907.414416, 0.0, 0.0, 0.0, 0.0, 19799.543905240946, 0.0, 0.0, 97.0]	[1.690139684842896, 56.20092472141174, 20390.816939508757, 641892603.0009766, 636921352.6576308, 0.0, 0.0, 0.0, 0.0, 19730.3108700549, 0.0, 0.0, 179.67494782338693]
Contoh validasi dengan menghitung standar deviasi <i>feature 1</i> dan <i>2</i> dari seluruh dataset	0.4910641044212609 & 20.396029661233744	0.443567747452046 & 22.131299068528147

Menggunakan teknik oversampling dan undersampling secara pipeline. Namun karena dibutuhkan dataset yang banyak untuk mencerminkan pola stokastik (data acak), oleh karena itu saat pertama kali oversampling sudah dibuat seimbang. jadi tidak dilakukan undersampling.

Dapat dilihat pada tabel 4.2, bahwa dengan dilakukan teknik SMOTE pada label “1” yang sebelumnya berjumlah 315 ditambahkan data sintetis menjadi total 615. Data sintetis memiliki nilai yang berbeda dengan data aslinya namun tetap menjaga jarak jadi nilai yang sudah ada, hal ini dapat divalidasi dengan menghitung standar deviasi dari beberapa fitur. Standar deviasi dataset sebelum dan sesudah dilakukan teknik SMOTE memiliki nilai yang hampir mirip, ini menandakan data sintetis yang dihasilkan tetap menjaga *features* (informasi penting) yang dihasilkan.

Seperti pada bahasan di bab perencanaan, bahwa penulis hanya mengimplementasikan teknik balancing data setelah dilakukan pembagian dataset dan pada dataset training saja, dan tidak diimplementasikan pada dataset timeseries.

D. Proses normalisasi data

Agar persebaran data semakin terlihat dengan *machine learning* maka dilakukan proses normalisasi data yaitu dengan menggunakan library SKLearn StandarScaller yaitu untuk cuplikan dari kode yang dipakai dapat dilihat pada gambar 4.17.

```
scalar = StandardScaler(copy=True, with_mean=True, with_std=True)
scalar.fit(X)
X = scalar.transform(X)
```

Gambar 4.17 Potongan Kode Proses Normalisasi Dataset

Penulis tidak lupa *instance* dari *scaler* akan disimpan untuk proses identifikasi *realtime* nantinya menggunakan *library* joblib yang kodennya dapat dilihat pada gambar 4.18.

```
joblib.dump(scalar, 'scaler_200.save')
```

Gambar 4.18 Potongan Kode Eksport Instance

Pada tabel 4.3 merupakan contoh perbandingan dataset sebelum normalisasi dan setelah normalisasi.

Tabel 4.3 Perbandingan Dataset Normalisasi

	Features 1	Features 2	Features 3
Sebelum Normalisasi	1.7067831430888252	0.0	138.14265180230296
Sesudah Normalisasi	0.25228296534028055	0.0	11.120501587408526

E. Proses pembagian data *training* dan *testing*

Untuk kode dari pembagian data *training* dan *testing* dapat dilihat pada gambar 4.19.

```
X_train, X_test, Y_train, Y_test = train_test_split(I, Y[200:], test_size = 0.2, stratify=Y[200:], random_state = 0)
```

Gambar 4.19 Potongan Kode Pembagian Dataset *Training* dan *Testing*

Digunakan *argument stratify* untuk menjaga perbandingan label data pada dataset yang akan dibagi, jadi pada dataset *training* dan *testing* perbandingan besar label tetap sama.

F. Proses pembuatan model *machine learning*

Untuk potongan kode pembuatan model dapat dilihat pada gambar 4.20.

```
def create_baseline():
    model = Sequential()
    model.add(Bidirectional(LSTM(64, activation='tanh', kernel_regularizer='l2')))
    model.add(Dense(128, activation = 'relu', kernel_regularizer='l2'))
    model.add(Dense(1, activation = 'sigmoid', kernel_regularizer='l2'))
    model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
    return model
```

Gambar 4.20 Potongan Kode Pembuatan Model *Machine Learning*

G. Proses latih model *machine learning*

Pada saat melatih model *machine learning* digunakan *callbacks* ModelCheckpoint dan EarlyStopping sesuai pada rancangan modul *machine learning*.

```

history = model.fit(X_train, Y_train, epochs = 30, validation_split=0.2, verbose = 1, callbacks=[checkpoint,early])

Epoch 1/30
370/371 [=====>.] - ETA: 0s - loss: 0.2823 - accuracy: 0.9940
Epoch 00001: val_loss improved from inf to 0.06291, saving model to LSTMModel.h5
371/371 [=====] - 12s 26ms/step - loss: 0.2819 - accuracy: 0.9939 - val_loss: 0.0629 - val_accuracy: 0.9879
Epoch 2/30
370/371 [=====>.] - ETA: 0s - loss: 0.1219 - accuracy: 0.9760
Epoch 00002: val_loss did not improve from 0.06291
371/371 [=====] - 9s 25ms/step - loss: 0.1218 - accuracy: 0.9761 - val_loss: 0.0825 - val_accuracy: 0.9879
Epoch 3/30
369/371 [=====>.] - ETA: 0s - loss: 0.2546 - accuracy: 0.9184
Epoch 00003: val_loss did not improve from 0.06291
371/371 [=====] - 9s 25ms/step - loss: 0.2543 - accuracy: 0.9186 - val_loss: 0.1733 - val_accuracy: 0.9656
Epoch 4/30
371/371 [=====] - ETA: 0s - loss: 0.1706 - accuracy: 0.9611
Epoch 00004: val_loss did not improve from 0.06291
371/371 [=====] - 9s 25ms/step - loss: 0.1706 - accuracy: 0.9611 - val_loss: 0.0687 - val_accuracy: 0.9963
Epoch 5/30

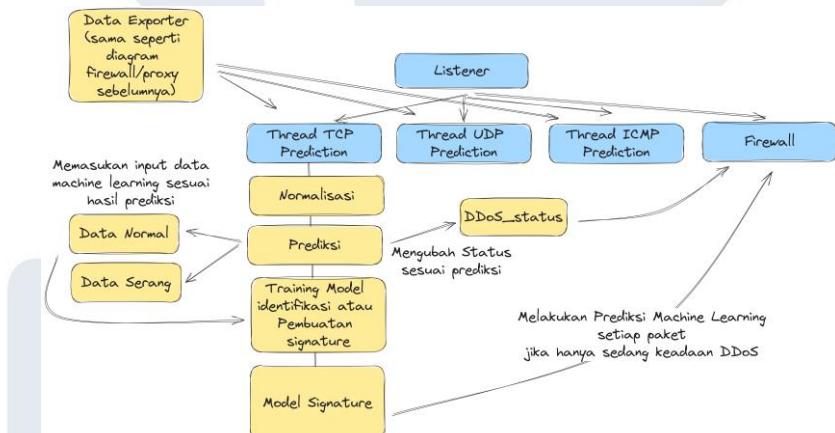
```

Gambar 4.21 Potongan Kode Training Machine Learning

Waktu yang diperlukan untuk melatih model *machine learning* dan *deep learning* adalah untuk algoritma *machine learning* dibutuhkan waktu hanya sekitar 1 detik saja, namun untuk LSTM memiliki rata rata waktu latih 30 menit dengan 10 – 20 epoch setiap protokolnya dan DNN memiliki rata rata waktu latih 5 menit dengan 100 epoch.

4.2.3 Pembangunan Sistem Firewall

Pada gambar 4.22 ditampilkan arsitektur dari sistem firewall



Gambar 4.22 Arsitektur Sistem Firewall

Untuk mengimplementasikan algoritma *machine learning* ke sistem *firewall* digunakan kode yang sama untuk menghasilkan dataset, namun tidak menyimpan ke suatu *file* tapi data dijadikan *input machine learning*. Untuk potongan kodenya dapat dilihat pada gambar 4.23. Dilakukan normalisasi data terlebih dahulu, lalu dilakukan prediksi data

```

icmp_timeseries_data_temp = icmp_lstm_scaler.transform(icmp_timeseries_data_temp)

features = len(icmp_timeseries_data_temp[0])
samples = icmp_timeseries_data_temp.shape[0]
train_len = icmp_timeseries_len
input_len = samples - train_len
I = np.zeros((samples - train_len, train_len, features))

for i in range(input_len):
    temp = np.zeros((train_len, features))
    for j in range(i, i + train_len - 1):
        temp[j-i] = icmp_timeseries_data_temp[j]
    I[i] = temp

icmp_predict = icmp_lstm_model.predict(I[:icmp_timeseries_len], verbose=1)
result = icmp_predict[0][0].round()
print(f'ICMP timeseries result : {result}, prediction time : {time.perf_counter() - icmp_prediction_time_start}')

```

Gambar 4.23 Potongan Kode Implementasi *Machine Learning* pada *Firewall Controller*

Untuk *signature* dari metode karakteristik paket terbanyak akan disimpan sebagai *object dictionary* yang dapat dilihat pada gambar 4.24.

<pre> # Traffic Signature tcp_signature = { "dataofs": 0, "reserved": 0, "flags": "S", "window": 0, "urgptr": 0, "payload_len": 0 } </pre>	<pre> def tcp_comparator(pkt, t): global tcp_signature if(tcp_signature['dataofs'] == int(t.dataofs) and tcp_signature['reserved'] == int(t.reserved) and tcp_signature['flags'] == str(t.flags) and tcp_signature['window'] == int(t.window) and tcp_signature['urgptr'] == int(t.urgptr) and tcp_signature['payload_len'] == int(pkt.get_payload_len())): return True else: return False </pre>
--	--

Gambar 4.24 Potongan Kode Objek Signature

```

if(sca.haslayer(TCP)):
    t = sca.getlayer(TCP)
    if t.dport in listOfWebserverPorts:
        tcp_networkLogger.info(f'{sca.src},{str(t.sport)},{str(t.seq)},'
                               f'{str(t.ack)},{str(t.window)}')
        tcp_identifier = [str(t.sport), str(t.seq), str(t.ack), str(t.window)]
        if(tcp_ddos):
            if(use_machine_learning_identifier):
                tcp_prediction_time_start = time.perf_counter()
                tcp_identifier[5] = StringToBytes(tcp_identifier[5])
                tcp_data = tcp_scaler.transform([tcp_identifier])
                tcp_result = tcp_svm_instance.predict([tcp_data[0]])[0]
                if(tcp_result == "1"):
                    pkt.drop()
                    print(f'TCP packet dropped, ip src : {sca.src}, pred : {tcp_result}')
                    return
            else:
                if(tcp_comparator(pkt, t)):
                    print(f'TCP packet blocked, ip src : {sca.src}')
                    pkt.drop()
                    return

```

Gambar 4.25 Potongan Kode Implementasi Firewall

Dapat dilihat pada gambar 4.25 bahwa ketika paket adalah DDoS atau bukan DDoS akan selalu dilakukan *logging* untuk mendeteksi keadaan secara *realtime*. Dan jika prediksi menyatakan merupakan paket dari serangan DDoS, maka paket akan didrop. Pada gambar 4.26 ditampilkan ketika program *firewall* sedang bekerja.

```

1/1 [=====] - 0s 488ms/step
ICMP timeseries result : 1.0, prediction time : 3.5406370729997434
ICMP Creating machine learning identifier
Size of feature dataset : 402
Size of feature dataset : 402
confussion matrix : [[41  0]
 [ 0 40]]
      precision    recall   f1-score   support
0         1.00     1.00     1.00      41
1         1.00     1.00     1.00      40

   accuracy          1.00      81
  macro avg       1.00     1.00     1.00      81
weighted avg       1.00     1.00     1.00      81

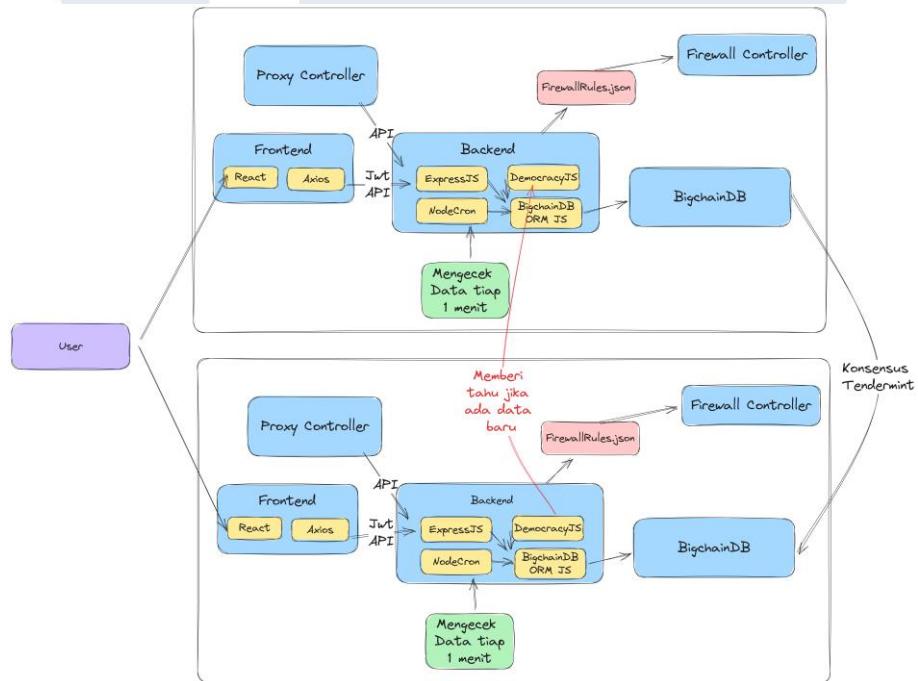
ICMP Finish creating machine learning identifier, creation time : 1.19229130200074
ICMP packet dropped, ip src : 192.168.29.136, prediction time : 0.0005046959995524958
ICMP packet dropped, ip src : 192.168.29.136, prediction time : 0.0008654419998492813
ICMP packet dropped, ip src : 192.168.29.136, prediction time : 0.02276863899987802

```

Gambar 4.26 Tampilan Program *Firewall* saat Bekerja

4.2.4 Pembangunan Sistem Distribusi Data

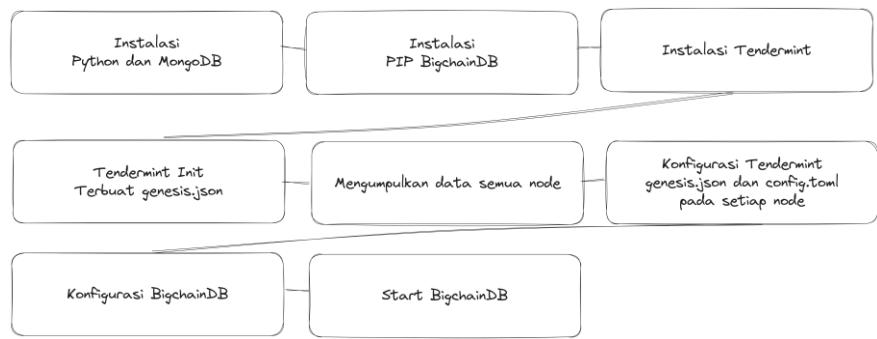
Pada gambar 4.27 ditampilkan arsitektur umum dari sistem distribusi data.



Gambar 4.27 Arsitektur Sistem Distribusi Data

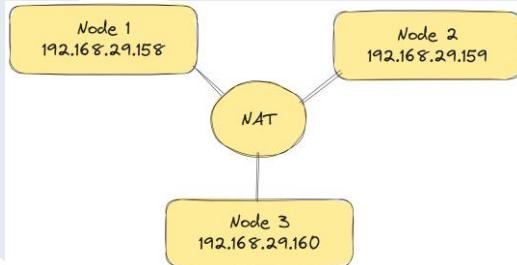
1. Instalasi BigchainDB

Ada beberapa alur yang perlu dilakukan oleh penulis saat melakukan instalasi *database* terdistribusi BigchainDB.



Gambar 4.28 Alur Instalasi BigchainDB Network

Sebagai gambar jumlah node dan ip address dari sistem yang ditampilkan dapat dilihat pada gambar 4.29.



Gambar 4.29 Ilustrasi Infrastruktur BigchainDB Network

Sistem distribusi data akan menggunakan BigchainDB sebagai *database* terdistribusinya. Oleh karena itu perlu dibangun beberapa *node* BigchainDB sebagai objek penelitian. Untuk menginstall BigchainDB dapat dilakukan dengan instalasi manual. Persyaratan yang harus diikuti adalah : *operating sistem* Linux, sudah terinstall Python versi 3, dan MongoDB.

Untuk instalasi BigchainDB dapat dilakukan melalui pip. Pip merupakan *package manager* dari python yang dapat mengunduh *dependency* secara otomatis. Untuk menginstal BigchainDB dijalankan perintah “`pip install bigchaindb`”, lalu bigchaindb dan dependency-nya akan terinstall.

Lalu BigchainDB memerlukan Tendermint sebagai sistem blockchain dan konsensusnya. Oleh karena itu perlu diinstall Tendermint dengan melakukan unduh data dari repositori resminya. Setelah terdownload, memindahkan data Tendermint ke `/usr/local/bin` agar terdaftar sebagai aplikasi sistem. Lalu jalankan “`tendermint`

init” untuk membuat genesis.json dan ID dari node tersebut. Seperti tangkapan yang ditampilkan pada gambar 4.30.

```
root@server:/home/server# tendermint init
I[2023-06-14|00:04:23.557] Found private validator
tor_key.json stateFile=/root/.tendermint/data/priv_validator_state.json
I[2023-06-14|00:04:23.557] Found node key
I[2023-06-14|00:04:23.557] Found genesis file
I[2023-06-14|00:04:23.557] module=main keyFile=/root/.tendermint/config/priv_validator.json
I[2023-06-14|00:04:23.557] module=main path=/root/.tendermint/config/node_key.json
I[2023-06-14|00:04:23.557] module=main path=/root/.tendermint/config/genesis.json
```

Gambar 4.30 Tampilan Tendermint Init

Genesis.json adalah suatu konfigurasi *file* yang digunakan oleh *blockchain* untuk menginisialisasi block pertama kali. Lalu penulis akan mengonfigurasikan BigchainDB *network* untuk menyatukan setiap *node*. Sebelum mengonfigurasikan Genesis.json, penulis mengumpulkan terlebih dahulu data data yang diperlukan tiap node yaitu hostname seperti IP Address, pub_key.value yang bisa didapatkan dari file “\$HOME/.tendermint/config/priv_validator.json”, dan node_id dengan menjalankan perintah “tendermint show_node_id”. Untuk data pub_key.value dan node_id yang dapat dilihat pada gambar 4.31.

```
[{"address": "291CC27F208DA59E84B0BA2534B840C22A7590A4",
"pub_key": {
    "type": "tendermint/PubKeyEd25519",
    "value": "Bkwtc+HxIBEMQ91tdsdNPe5Q4bDlvEGSNk6eDGBbL08="
},
"priv_key": {
    "type": "tendermint/PrivKeyEd25519",
    "value": "JUN4XJ/T1Pra1pX5nnclfSZSpomBYsej5zhQUT98qVrMGTc1z4fEgEQxD3wL2x0097lDhs0W8QZI2Tp4MYFsvTw=="
}]
```

```
root@server:~/.tendermint/config# tendermint show_node_id
482fb27994d112a4e22da0e5d48d227b0bd34347
```

Gambar 4.31 Data public key node dan Node Id

Setelah ketiga data atas semua *node* terkumpul, penulis melakukan perubahan pada file Genesis.json dengan berisi data data yang telah dikumpulkan. Data yang diubah dapat dilihat pada gambar 4.32.

UNIVERSITAS
MULTIMEDIA
NUSANTARA

```
GNU nano 5.4
{
  "genesis_time": "2023-06-10T18:29:56.286524561Z",
  "chain_id": "test-chain-yfdWWd",
  "consensus_params": {
    "block": {
      "max_bytes": "22020096",
      "max_gas": "-1",
      "time_iota_ms": "1000"
    },
    "evidence": {
      "max_age": "100000"
    },
    "validator": {
      "pub_key_types": [
        "ed25519"
      ]
    }
  },
  "validators": [
    {
      "address": "291CC27F2D8DA59EB4B0BA2534B840C22A7590A4",
      "pub_key": {
        "type": "tendermint/PubKeyEd25519",
        "value": "Bkwtc+HxIBEMQ91tdsdNPe5Q4bDlvEGSNk6eDGBbL08="
      },
      "power": "10",
      "name": "Intelligent Proxy Node 1"
    },
    {
      "address": "51I07E2I0ZPHHWVTK8W8HJIC7KNMPG3W03GH7YZ",
      "pub_key": {
        "type": "tendermint/PubKeyEd25519",
        "value": "84BoR0G+ZJ03zG8I0bg1wKzdm/i0sWn6odz+ew4nxgs="
      },
      "power": "10",
      "name": "Intelligent Proxy Node 2"
    },
    {
      "address": "SE60E00YEMDAS1VQRD2NVVYZ4ZNVTBRYXXGI6N9Q",
      "pub_key": {
        "type": "tendermint/PubKeyEd25519",
        "value": "iVJWULGVVCqGQwkyvu3fZb1FN9euqz3USBrv758vGw=="
      },
      "power": "10",
      "name": "Intelligent Proxy Node 3"
    }
  ],
  "app_hash": ""
}
```

Gambar 4.32 Isi File genesis.json Tendermint

Setiap *node* harus memiliki *file* genesis.json yang sama semua, untuk mendeklarasikan genesis_time, chain_id, dan list dari validators. Data data yang sebelumnya dikumpulkan dimasukkan ke dalam *list validators*. Untuk *power mining* diset sama semua walaupun pada Tendermint tidak ada serangan 51% attack. Ketika blockchain sudah berjalan dan ada *node* yang akan bergabung, blockchain tidak perlu dibuat ulang namun dengan menambahkan identitas *node* baru saja ke genesis.json setiap *node*-nya.

Lalu penulis juga melakukan konfigurasi pada config.toml. Pada config.toml ini ada beberapa konfigurasi bawaan Tendermint yang harus diubah dan ditambahkan agar memenuhi kebutuhan BigchainDB dalam melakukan konsensus. Untuk data yang diubah dapat dilihat pada gambar 4.33.

```

GNU nano 5.4

##### main base config options #####
# TCP or UNIX socket address of the ABCI application,
# or the name of an ABCI application compiled in with the Tendermint binary
proxy_app = "tcp://127.0.0.1:26658"

# A custom human readable name for this node
moniker = "Intelligent Proxy Node 1"

# EmptyBlocks mode and possible interval between empty blocks
create_empty_blocks = false
create_empty_blocks_interval = "0s"

# Output level for logging, including package level options
log_level = "main:info,state:info,*:error"
|
# Output format: 'plain' (colored text) or 'json'
log_format = "plain"

# Rate at which packets can be sent, in bytes/second
send_rate = 102400000

# Rate at which packets can be received, in bytes/second
recv_rate = 102400000

##### mempool configuration options #####
[mempool]

recheck = false
broadcast = true
wal_dir = ""

# Comma separated list of nodes to keep persistent connections to
persistent_peers = "482fb27884d112a0e22da0e5d48d227b0bd3434f@192.168.29.158:26656, \
nbg3esunbalvlpb9x6wonmmdu2n6njfnu4337yjr@192.168.29.159:26656, \
wnvd4zo5loz1lerj5zpm7zj8njabo79aheixmbg@192.168.29.160:26656,"
```

Gambar 4.33 Konfigurasi Tendermint Config.toml

Yang terpenting adalah menambahkan *hostname* dan identitas *node* yang akan terhubung ke *persistent_peers*. Penulis mengulangi tahapan pemodifikasinya genesis.json dan config.toml pada setiap *node* yang dipakai.

Setelah Tendermint telah siap untuk BigchainDB network, penulis perlu mengonfigurasi BigchainDB. Pengaturan yang perlu dikonfigurasi adalah untuk *API server bind* bisa dikonfigurasikan dengan 0.0.0.0:9984 untuk dapat diakses dari mana saja dan localhost:9984 untuk hanya bisa diakses dari node itu saja. Lalu untuk Tendermint host perlu dikonfigurasikan menjadi 0.0.0.0 agar tiap node dapat berkomunikasi satu dengan lainnya. Untuk konfigurasinya dapat dilihat pada gambar 4.34.

```
root@server:/home/server# bigchaindb configure
INFO:bigchaindb.config_utils:Configuration loaded from '/root/.bigchaindb'
Config file '/root/.bigchaindb' exists, do you want to override it? (cannot be undone) [y/N]: y
Generating default configuration for backend localmongodb
API Server bind? (default '0.0.0.0:9984'): localhost:9984
WebSocket Server scheme? (default 'ws'):
WebSocket Server host? (default 'localhost'):
WebSocket Server port? (default '9985'):
Database host? (default 'localhost'):
Database port? (default '27017'):
Database name? (default 'bigchain'):
Tendermint host? (default 'localhost')0.0.0.0
Tendermint port? (default '26657')
Configuration written to /root/.bigchaindb
Ready to go!
root@server:/home/server#
```

Gambar 4.34 Konfigurasi BigchainDB

Setelah Tendermint dan BigchainDB selesai, sebelum BigchainDB dijalankan jika sistem menggunakan *firewall* maka perlu dilakukan perubahan *firewall rule* untuk *whitelist port* yang digunakan oleh Tendermint yaitu 26657.

```
root@server:/home/server# sudo ufw allow 26657
Rules updated
Rules updated (v6)
```

Gambar 4.35 Konfigurasi Firewall untuk Tendermint

Setelah itu kita bisa menjalankan BigchainDB, untuk pertama kali kita jalankan “bigchaindb init” terlebih dahulu untuk membuat tabel di MongoDB, dan setelah itu bisa jalankan “bigchaindb start”. Untuk tampilan BigchainDB setelah berjalan dapat dilihat pada gambar 4.36.

```
[2023-06-14 17:27:50] [INFO] (bigchaindb.start)
*****
*                                         BigchainDB 2.2.2
*  codename "jumping sloth"
*  Initialization complete. BigchainDB Server is ready and waiting.
*
*  You can send HTTP requests via the HTTP API documented in the
*  BigchainDB Server docs at:
*  https://bigchaindb.com/http-api
*
*  Listening to client connections on: localhost:9984
*
*****
(MainProcess - pid: 2713)
[2023-06-14 17:27:50 -0700] [2741] [INFO] Starting gunicorn 20.0.4
```

Gambar 4.36 Tampilan BigchainDB setelah Berjalan

2. Pembangunan Backend

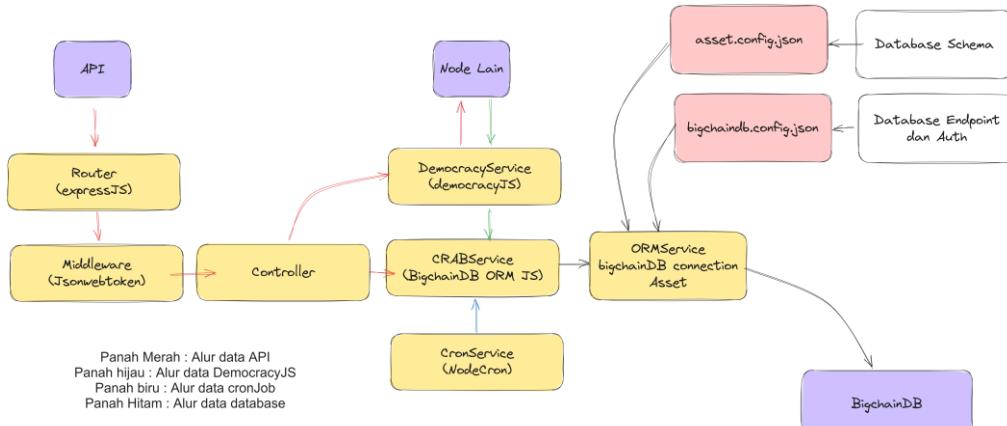
Sistem backend yang dibuat menggunakan library Express JS dimulai dengan menginstall ExpressJS, BigchainDB ORM JS, DemocracyJS, Node-cron, Jsonwebtoken, cors, body-parser, dan bcryptJs dengan NPM. Struktur dari folder backend yang dibuat oleh penulis dapat dilihat pada gambar 4.37.

```

    < backend
      > config
      > configs
        { asset.config.json
        { bigchaindb.config.json
      < controllers
        JS auth.controller.js
        JS firewall.controller.js
        JS log.controller.js
        JS user.controller.js
      < database
        JS connect.bigchaindb.js
      > helper
      < middlewares
        JS authJWT.js
        JS index.js
        JS verifyFirewall.js
        JS verifySignUp.js
      < routes
        JS auth.routes.js
        JS firewall.routes.js
        JS log.routes.js
        JS user.routes.js
      < services
        JS CRABService.js
        JS CronService.js
        JS DemocracyService.js
        JS ORMService.js
        { .gitignore
        { FirewallRulesClone.json
        JS index.js
        { package-lock.json
        { package.json
        { README.md
  
```

Gambar 4.37 Struktur Folder Backend

Terlihat dari struktur folder backend diatas, terdapat 6 folder utama yaitu *configs* untuk menyimpan file konfigurasi backend, *controller* untuk fungsi-fungsi dari *endpoint* yang dipakai, *database* untuk driver ke BigchainDB, middleware untuk melakukan validasi *form* dan JWT, routes untuk konfigurasi *endpoint* dan *controller*, dan *services* untuk modul-modul *service* yang digunakan pada backend. Untuk arsitektur dari modul backend dapat dilihat pada gambar 4.38.



Gambar 4.38 Arsitektur Modul Backend

Penulis akan membahas beberapa pokok dari fungsi backend, yaitu :

N U S A N T A R A

1. Contoh operasi *database*

Untuk membuat data atau melakukan operasi CRAB pada database dapat menggunakan BigchainDB ORM sehingga dengan mudah dilakukan dengan menggunakan fungsi *.createAsset* pada database instance yang sudah dinasalisasi dengan *asset* yang akan dioperasikan dan koneksi ke *database*. Setelah itu perlu dikirim *keypair* dan *metadata*. Contoh salah satu operasi *database* dapat dilihat pada potongan kode di gambar 4.39.

```
crabService.createAsset(userKeypair, metadata).then((value) => {
  // Publish to consensus channel
  dem.publish("firewall-channel", "A new firewall data has been created, id : " + value.id);

  let firewallConfiguration = [];
  value.data.data.map((asset) => {
    firewallConfiguration.push(asset.ipAddress);
  });
  console.log("updating firewall rules to FirewallRules.json");
  firewallRules.ListOfBannedIpAddr = firewallConfiguration;
  fs.writeFile(fileName, JSON.stringify(firewallRules), function writeJSON(err) {
    if (err) return console.log(err);
    // console.log(JSON.stringify(firewallRules));
    // console.log("writing to " + fileName);
  });

  return res.json(value);
});
```

Gambar 4.39 Potongan Kode Database untuk Operasi Data

2. Democracy JS

Untuk melakukan konsensus dengan *democracyJS* kita perlu mengonfigurasikan *peers* mana saja yang akan terhubung dengan *node* itu. Konfigurasi ini dinasalisasi saat pembuatan objek *democracy*. Untuk contohnya dapat dilihat pada gambar 4.40.

```
var dem = new Democracy({
  source: "192.168.29.158:5000",
  peers: ["192.168.29.158:5000", "192.168.29.159:5000", "192.168.29.160:5000"],
});
```

Gambar 4.40 Potongan Kode Konfigurasi DemocracyJS

Lalu untuk melakukan pertukaran data, dilakukan pembuatan fungsi ketika mendapatkan pesan dan melakukan *subscribe* ke suatu *channel pub/sub*. Untuk contohnya dapat dilihat pada potongan kode pada gambar 4.41.

```
dem.on("firewall-channel", (msg) => {
    console.log("New firewall rules from other nodes, firewall id : " + msg);

    crabService.retrieveAllAssets().then((value) => {
        let status = false;

        value.map((asset) => { ... });
    });

    if (status == false) {
        let firewallConfiguration = [];

        console.log("updating firewall rules to FirewallRules.json");
        firewallRules.ListOfBannedIpAddr = firewallConfiguration;
        fs.writeFile(fileName, JSON.stringify(firewallRules), function writeJSON(err) {
            if (err) return console.log(err);
            // console.log(JSON.stringify(firewallRules));
            // console.log("writing to " + fileName);
        });
    }
});

dem.subscribe("firewall-channel");
```

Gambar 4.41 Potongan Kode Pub/Sub democracyJS

3. Cronjob

Untuk memastikan *firewall rule* selalu diperbaharui maka diperlukan cronjob untuk menjalankan suatu fungsi pada jadwalnya. Untuk melakukan cronjob setiap 1 menit dapat menggunakan kode pada gambar 4.42, yaitu “* * * * *” artinya setiap menit, jam, bulan, tahun, dan hari apapun.

```
cron.schedule("* * * * *", () => {
    console.log("Cron for checking firewall rule blockchain");

    crabService.retrieveAllAssets().then((value) => {
        let status = false;

        value.map((asset) => { ... });
    });

    if (status == false) {
        let firewallConfiguration = [];

        console.log("updating firewall rules to FirewallRules.json");
        firewallRules.ListOfBannedIpAddr = firewallConfiguration;
        fs.writeFile(fileName, JSON.stringify(firewallRules), function writeJSON(err) {
            if (err) return console.log(err);
            // console.log(JSON.stringify(firewallRules));
            // console.log("writing to " + fileName);
        });
    }
});
```

Gambar 4.42 Potongan Kode Backend CronJob

Setelah semua modul backend terpasang, tampilan saat aplikasi berjalan seperti gambar 4.43.

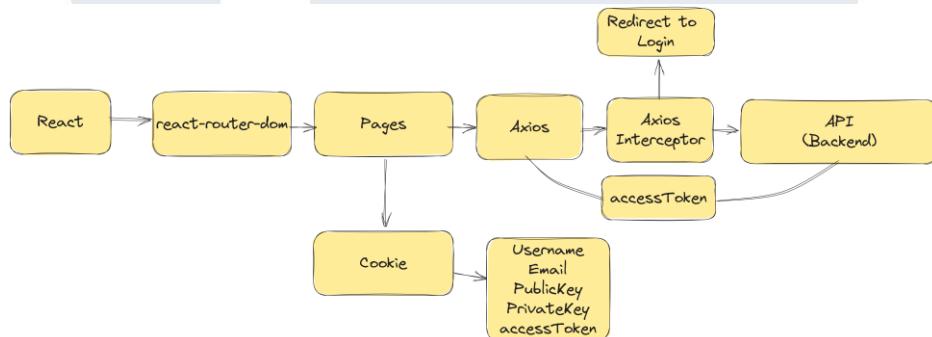
```
(base) brandondani@brandondani-virtual-machine:~/tugasakhir/IntelligentProxy/backend$ npm run dev
> backend-intelligent-proxy@1.0.0 dev
> nodemon index.js

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Server is running on port 5000.
You are elected leader!
Cron for checking firewall rule blockchain
Cron for checking firewall rule blockchain
```

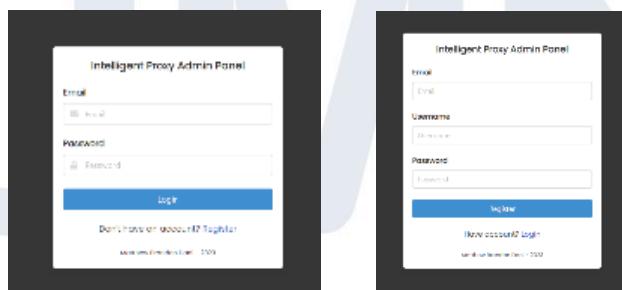
Gambar 4.43 Tampilan Sistem Backend

3. Pembangunan Frontend

Sistem frontend dibuat menggunakan framework ReactJS, dimulai dengan menginstall react, axios, bootstrap 5, dan react-router-dom melalui NPM. Arsitektur dari sistem frontend dapat dilihat pada gambar 4.44 dan hasil tampilan dari frontend yang dibuat pada gambar 4.45

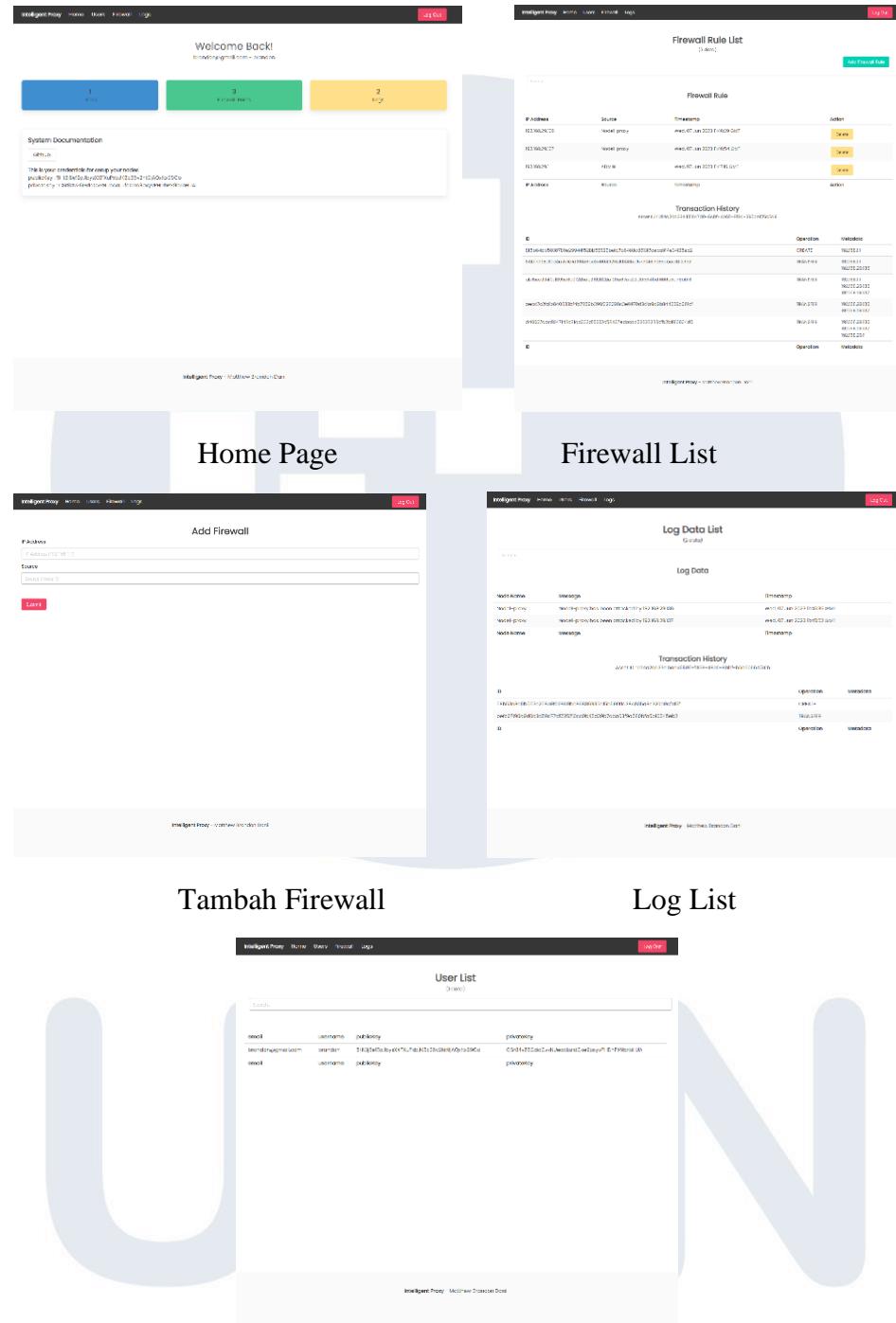


Gambar 4.44 Arsitektur Sistem Backend



Login Page

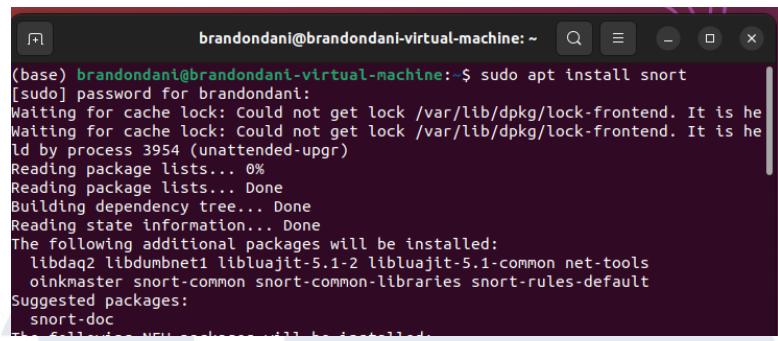
Register Page



Gambar 4.45 Tampilan Sistem Frontend

4.2.5 Pembangunan IPS / IDS Snort

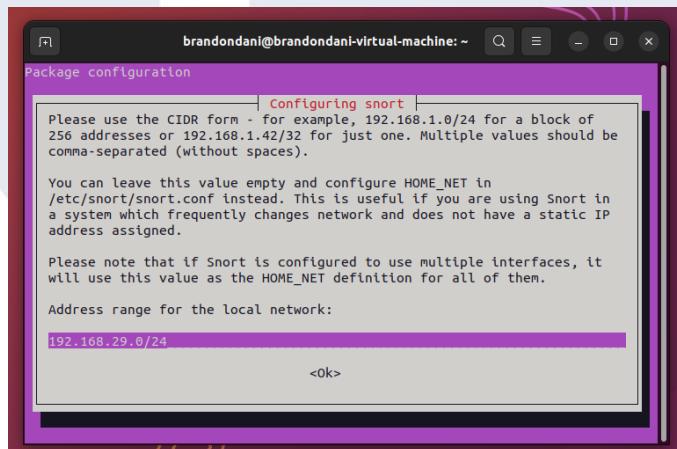
Untuk melakukan instalasi aplikasi Snort dapat dilakukan dengan mudah yaitu dengan menjalankan perintah “sudo apt install snort” seperti pada gambar 4.46.



```
(base) brandondani@brandondani-virtual-machine: ~ $ sudo apt install snort
[sudo] password for brandondani:
Waiting for cache lock: Could not get lock /var/lib/dpkg/lock-frontend. It is held by process 3954 (unattended-upgr)
Reading package lists... 0%
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
libdaq2 libdumbnet1 libluajit-5.1-2 libluajit-5.1-common net-tools
oinkmaster snort-common snort-common-libraries snort-rules-default
Suggested packages:
snort-doc
The following NEW packages will be installed:
libdaq2 libdumbnet1 libluajit-5.1-2 libluajit-5.1-common net-tools
oinkmaster snort-common snort-common-libraries snort-rules-default
```

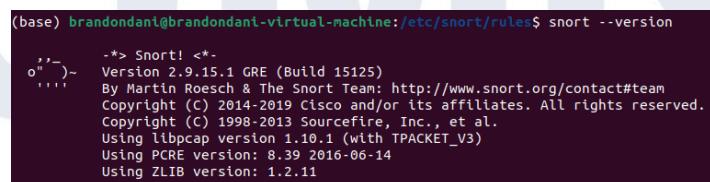
Gambar 4.46 Instalasi Snort

Lalu ketika ada *prompt* untuk menginput *address range* untuk *local network* dapat diinput untuk *subnet NAT* yang dipakai.



Gambar 4.47 Prompt Instalasi Snort

Berikut merupakan tampilan ketika aplikasi Snort berhasil terinstal yang terlihat pada gambar 4.47.

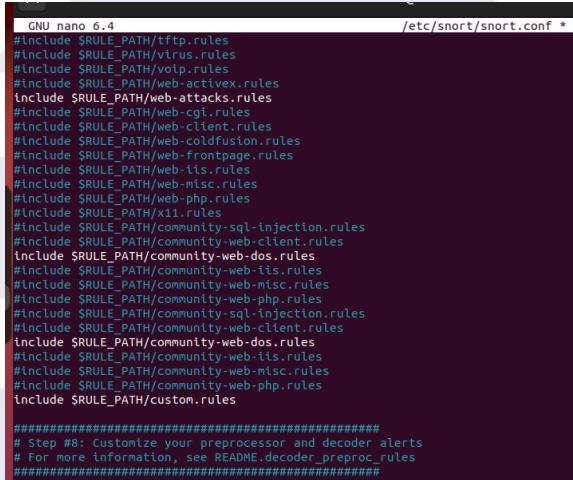


```
(base) brandondani@brandondani-virtual-machine:/etc/snort/rules$ snort --version
--> Snort! <--
o'''~- Version 2.9.15.1 GRE (Build 15125)
By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
Copyright (C) 2014-2019 Cisco and/or its affiliates. All rights reserved.
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using libpcap version 1.10.1 (with TPACKET_V3)
Using PCRE version: 8.39 2016-06-14
Using ZLIB version: 1.2.11
```

Gambar 4.48 Aplikasi Snort

Snort merupakan *command line application* sehingga tidak ada GUI bawaan untuk mengatur konfigurasi dari Snort ini. Untuk mengonfigurasi seperti menambahkan atau menonaktifkan *rules* dan mengubah mode dari Snort itu sendiri.

Konfigurasi perlu ditambahkan di file “/etc/snort/snort.conf”. Untuk isi dari *file* konfigurasi dapat dilihat pada gambar 4.49.

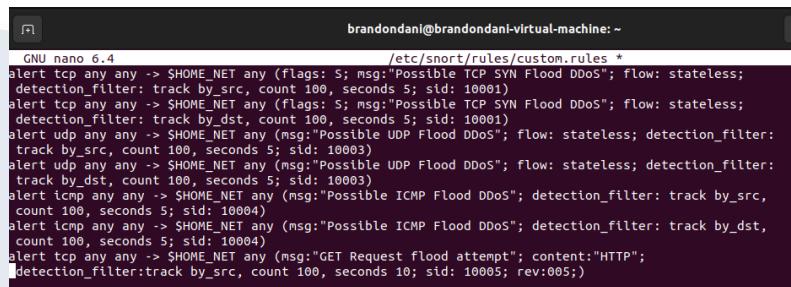


```
GNU nano 6.4                                     /etc/snort/snort.conf *
#include $RULE_PATH/tftp.rules
#include $RULE_PATH/virus.rules
#include $RULE_PATH/voip.rules
#include $RULE_PATH/web-activex.rules
#include $RULE_PATH/web-attacks.rules
#include $RULE_PATH/web-cgi.rules
#include $RULE_PATH/web-client.rules
#include $RULE_PATH/web-coldfusion.rules
#include $RULE_PATH/web-frontpage.rules
#include $RULE_PATH/web-iis.rules
#include $RULE_PATH/web-misc.rules
#include $RULE_PATH/web-php.rules
#include $RULE_PATH/x11.rules
#include $RULE_PATH/community-sql-injection.rules
#include $RULE_PATH/community-web-client.rules
#include $RULE_PATH/community-web-dos.rules
#include $RULE_PATH/community-web-iis.rules
#include $RULE_PATH/community-web-misc.rules
#include $RULE_PATH/community-web-php.rules
#include $RULE_PATH/community-sql-injection.rules
#include $RULE_PATH/community-web-client.rules
#include $RULE_PATH/community-web-dos.rules
#include $RULE_PATH/community-web-iis.rules
#include $RULE_PATH/community-web-misc.rules
#include $RULE_PATH/community-web-php.rules
#include $RULE_PATH/custom.rules

#####
# Step #8: Customize your preprocessor and decoder alerts
# For more information, see README.decoder_preproc_rules
#####
```

Gambar 4.49 Isi dari snort.conf pada Aplikasi Snort

Secara bawaan awal, aplikasi Snort sudah terdapat banyak sekali ruleset dari komunitas dan developer Snort sendiri (Cisco Talos). *Ruleset* bawaan terdapat pada folder /etc/snort/rules/. Kita dapat membuat rule baru di folder tersebut, misal untuk penulis akan menambahkan *ruleset* baru yaitu “custom.rules”. Maka dapat diisi sesuai gambar 4.50.



```
brandondani@brandondani-virtual-machine:~                                     /etc/snort/rules/custom.rules *
GNU nano 6.4
alert tcp any any -> $HOME_NET any (flags: S; msg:"Possible TCP SYN Flood DDoS"; flow: stateless; detection_filter: track_by_src, count 100, seconds 5; std: 10001)
alert tcp any any -> $HOME_NET any (flags: S; msg:"Possible TCP SYN Flood DDoS"; flow: stateless; detection_filter: track_by_dst, count 100, seconds 5; std: 10001)
alert udp any any -> $HOME_NET any (msg:"Possible UDP Flood DDoS"; flow: stateless; detection_filter: track_by_src, count 100, seconds 5; std: 10003)
alert udp any any -> $HOME_NET any (msg:"Possible UDP Flood DDoS"; flow: stateless; detection_filter: track_by_dst, count 100, seconds 5; std: 10003)
alert icmp any any -> $HOME_NET any (msg:"Possible ICMP Flood DDoS"; detection_filter: track_by_src, count 100, seconds 5; std: 10004)
alert icmp any any -> $HOME_NET any (msg:"Possible ICMP Flood DDoS"; detection_filter: track_by_dst, count 100, seconds 5; std: 10004)
alert tcp any any -> $HOME_NET any (msg:"GET Request flood attempt"; content:"HTTP";
[detection_filter:track_by_src, count 100, seconds 10; std: 10005; rev:005;]
```

Gambar 4.50 Custom Ruleset Aplikasi Snort

Ketika kita sudah menambahkan *ruleset* baru dan mengkonfigurasi Snort, dapat dijalankan dengan menggunakan perintah “sudo snort -A console -q -u snort -c /etc/snort/snort.conf -i {id network interface}”. Untuk id dari *network interface* dapat dilihat dengan menggunakan perintah “ip a” pada OS linux. Untuk tampilan ketika terjadi serangan maka tampilan *console* akan seperti pada gambar 4.51.

```
(base) brandondant@brandondant-virtual-machine:/etc/snort/rules$ sudo snort -A console -q -u snort -c /etc/snort/snort.conf -l ens33
06/14-16:30:33.320862 [**] [1:1:0005:5] GET Request flood attempt [**] [Priority: 0] [TCP] 192.168.29.136:47468 -> 192.168.29.158:5000
06/14-16:30:33.323002 [**] [1:1:0005:5] GET Request flood attempt [**] [Priority: 0] [TCP] 192.168.29.136:47490 -> 192.168.29.158:5000
06/14-16:30:33.327789 [**] [1:1:0005:5] GET Request flood attempt [**] [Priority: 0] [TCP] 192.168.29.136:47480 -> 192.168.29.158:5000
06/14-16:30:33.328883 [**] [1:1:0005:5] GET Request flood attempt [**] [Priority: 0] [TCP] 192.168.29.136:47498 -> 192.168.29.158:5000
06/14-16:30:33.335393 [**] [1:1:0005:5] GET Request flood attempt [**] [Priority: 0] [TCP] 192.168.29.136:47510 -> 192.168.29.158:5000
06/14-16:30:33.337359 [**] [1:1:0001:0] Possible TCP SYN Flood DDoS [**] [Priority: 0] [TCP] 192.168.29.136:47518 -> 192.168.29.158:5000
06/14-16:30:33.342156 [**] [1:1:0001:0] Possible TCP SYN Flood DDoS [**] [Priority: 0] [TCP] 192.168.29.136:47534 -> 192.168.29.158:5000
06/14-16:30:33.342795 [**] [1:1:0005:5] GET Request flood attempt [**] [Priority: 0] [TCP] 192.168.29.136:47538 -> 192.168.29.158:5000
06/14-16:30:33.343760 [**] [1:1:0005:5] GET Request flood attempt [**] [Priority: 0] [TCP] 192.168.29.136:47534 -> 192.168.29.158:5000
06/14-16:30:34.049873 [**] [1:1:0001:0] Possible TCP SYN Flood DDoS [**] [Priority: 0] [TCP] 192.168.29.136:47550 -> 192.168.29.158:5000
06/14-16:30:34.050521 [**] [1:1:0005:5] GET Request flood attempt [**] [Priority: 0] [TCP] 192.168.29.136:47550 -> 192.168.29.158:5000
06/14-16:30:34.312345 [**] [1:1:0001:0] Possible TCP SYN Flood DDoS [**] [Priority: 0] [TCP] 192.168.29.136:47556 -> 192.168.29.158:5000
06/14-16:30:34.313168 [**] [1:1:0005:5] GET Request flood attempt [**] [Priority: 0] [TCP] 192.168.29.136:47556 -> 192.168.29.158:5000
06/14-16:30:34.890415 [**] [1:1:0001:0] Possible TCP SYN Flood DDoS [**] [Priority: 0] [TCP] 192.168.29.136:47570 -> 192.168.29.158:5000
06/14-16:30:34.890894 [**] [1:1:0005:5] GET Request flood attempt [**] [Priority: 0] [TCP] 192.168.29.136:47570 -> 192.168.29.158:5000
06/14-16:30:35.074048 [**] [1:1:0001:0] Possible TCP SYN Flood DDoS [**] [Priority: 0] [TCP] 192.168.29.136:47580 -> 192.168.29.158:5000
06/14-16:30:35.074936 [**] [1:1:0005:5] GET Request flood attempt [**] [Priority: 0] [TCP] 192.168.29.136:47580 -> 192.168.29.158:5000
06/14-16:30:36.053413 [**] [1:1:0001:0] Possible TCP SYN Flood DDoS [**] [Priority: 0] [TCP] 192.168.29.136:47586 -> 192.168.29.158:5000
06/14-16:30:36.054186 [**] [1:1:0005:5] GET Request flood attempt [**] [Priority: 0] [TCP] 192.168.29.136:47586 -> 192.168.29.158:5000
06/14-16:30:36.089134 [**] [1:1:0001:0] Possible TCP SYN Flood DDoS [**] [Priority: 0] [TCP] 192.168.29.136:47600 -> 192.168.29.158:5000
06/14-16:30:36.089994 [**] [1:1:0005:5] GET Request flood attempt [**] [Priority: 0] [TCP] 192.168.29.136:47600 -> 192.168.29.158:5000
06/14-16:30:36.156961 [**] [1:1:0001:0] Possible TCP SYN Flood DDoS [**] [Priority: 0] [TCP] 192.168.29.136:47608 -> 192.168.29.158:5000
06/14-16:30:36.157761 [**] [1:1:0005:5] GET Request flood attempt [**] [Priority: 0] [TCP] 192.168.29.136:47608 -> 192.168.29.158:5000
06/14-16:30:36.899812 [**] [1:1:0001:0] Possible TCP SYN Flood DDoS [**] [Priority: 0] [TCP] 192.168.29.136:47614 -> 192.168.29.158:5000
06/14-16:30:36.960315 [**] [1:1:0005:5] GET Request flood attempt [**] [Priority: 0] [TCP] 192.168.29.136:47614 -> 192.168.29.158:5000
06/14-16:30:37.334663 [**] [1:1:0001:0] Possible TCP SYN Flood DDoS [**] [Priority: 0] [TCP] 192.168.29.136:47628 -> 192.168.29.158:5000
06/14-16:30:37.334877 [**] [1:1:0005:5] GET Request flood attempt [**] [Priority: 0] [TCP] 192.168.29.136:47628 -> 192.168.29.158:5000
```

Gambar 4.51 Tampilan Ketika Program Snort Dijalankan

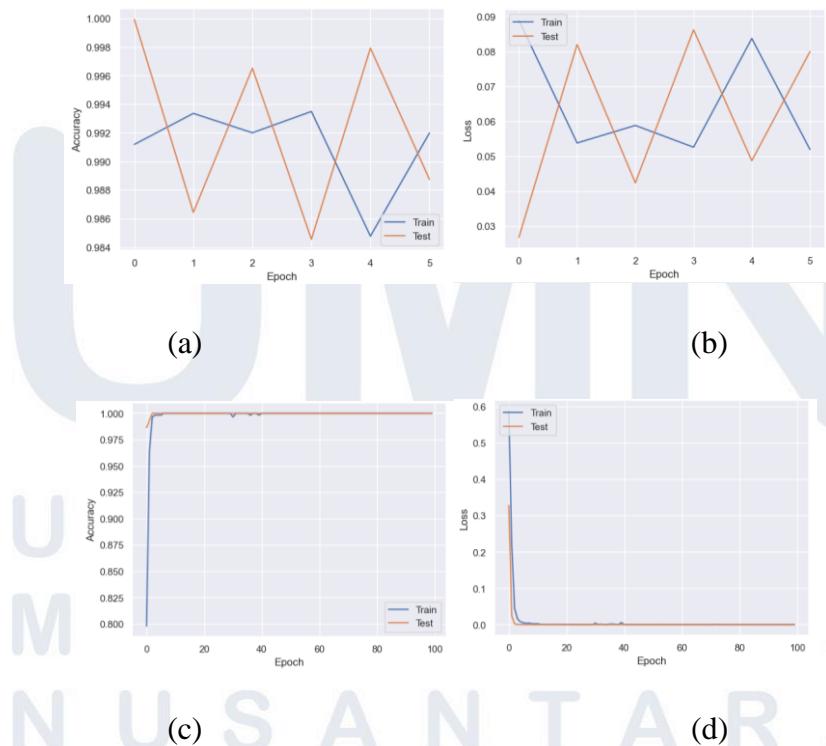
4.3 Hasil dan analisis Pengujian Sistem

Dari sistem yang sudah dibangun, didapatkan hasil evaluasi sesuai metrik penelitian yang sudah ditetapkan.

4.3.1 Analisis Hasil Evaluasi Performa Machine Learning

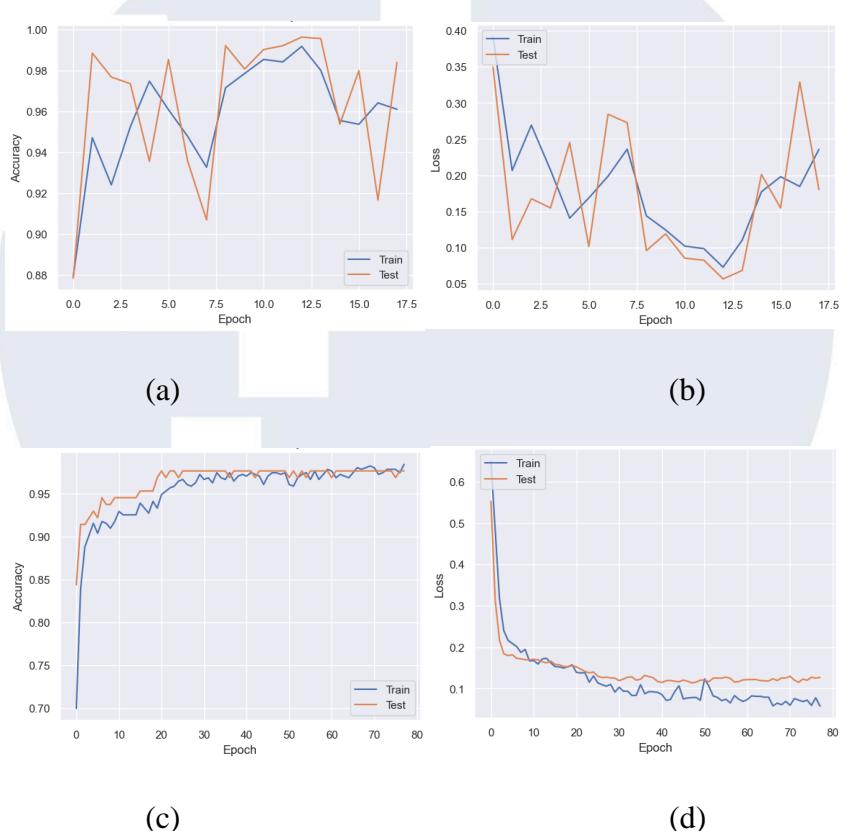
1. Analisis Hasil Latih Bidirectional Lstm dan DNN untuk Validasi Kualitas Dataset

a. TCP



Gambar 4.52 Plot Akurasi dan Loss terhadap Epoch TCP, (a) dan (b)
 Bidirectional LSTM dan (c) dan (d) DNN

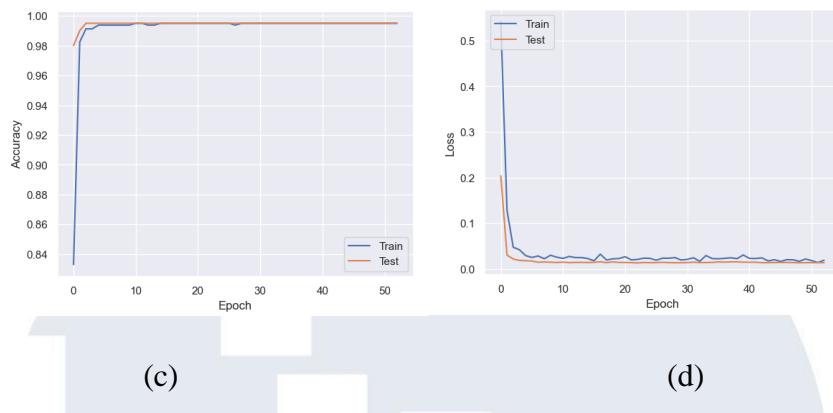
b. UDP



Gambar 4.53 Plot Akurasi dan Loss terhadap Epoch UDP, (a) dan (b)
 Bidirectional LSTM dan (c) dan (d) DNN

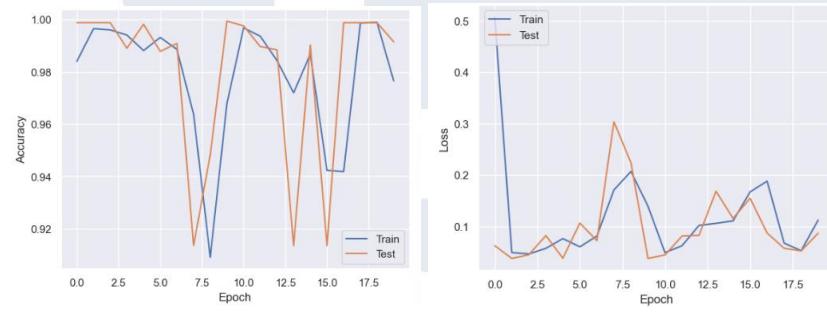
c. ICMP



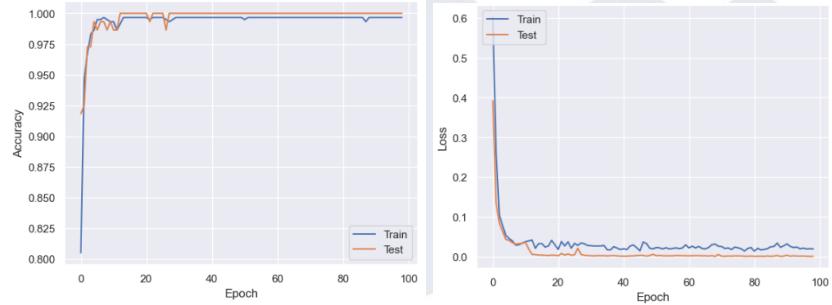


Gambar 4.54 Plot Akurasi dan Loss terhadap Epoch ICMP, (a) dan (b)
Bidirectional LSTM dan (c) dan (d) DNN

d. HTTP



(a) (b)



Gambar 4.55 Plot Akurasi dan Loss terhadap Epoch HTTP, (a) dan (b)
Bidirectional LSTM dan (c) dan (d) DNN

Hasil latih model bidirectional LSTM menunjukkan adanya ketidakstabilan setiap iterasi epochnya terutama pada protokol TCP dan HTTP seperti yang ditampilkan pada gambar 5.52 dan 5.54, hal ini bisa disebabkan oleh beberapa faktor salah satunya adalah kualitas dataset metode *timeseries* yang kurang baik. Menurut penulis penyebab kualitas dataset metode *timeseries* yang kurang baik adalah karena skenario yang dijalankan penulis untuk mendapatkan dataset tidak mencerminkan keadaan normalnya. Yaitu salah satunya, kita tidak akan tahu serangan DDoS akan dilancarkan kapan dan pada target apa, dan terjadi ketika situasi normal sedang terjadi. Seharusnya dilakukan metode pengumpulan dataset yang lebih matang dan memiliki lebih banyak variasi skenario yang mencerminkan situasi normalnya. Dan untuk pelabelan dataset akan menjadi lebih kompleks karena akan ada saat keadaan normal yang tiba tiba terserang DDoS. Selain juga dapat diperbaiki kembali penggunaan *features* untuk berdampak pada proses klasifikasinya menggunakan metode *features* analisis selain SVM yang mewakili analisis pada metode *recurrent*. Dengan perbaikan ini dimungkinkan akan menambah kualitas dataset metode *timeseries*.

Untuk dataset metode individual dari grafik latih algoritma DNN dapat dianalisis bahwa dataset ini memiliki kualitas yang lebih baik dari pada metode *machine learning timeseries* yang diperlihatkan dari kestabilan setiap iterasi epochnya. Namun karena skenario pengambilannya sama dengan metode *timeseries* maka hasil performa klasifikasinya belum tentu lebih baik dan masih bisa ditingkatkan lagi dengan menambah jumlah dataset dengan menambah skenario pengambilan dataset.

2. Analisis performa *machine learning* deteksi serangan DDoS menggunakan *classification report* dan *confussion matrix*. Penulis menandai hasil *machine learning* yang paling bagus dengan warna hijau dan hasil yang paling buruk dengan warna merah dengan membandingkan nilai akurasi dan AUC-ROC.

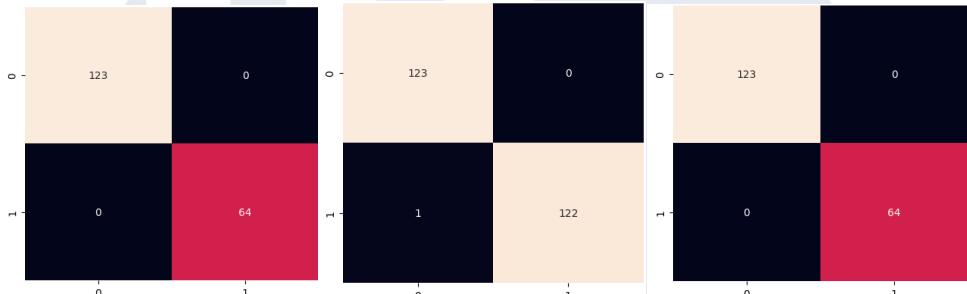
Tabel 4.4 Perbandingan Hasil Performa Machine Learning dengan Classification Report

Protokol	Model Name	Akurasi	precision	recall	F-1	AUC-ROC
TCP	SVM	100%	1.00	1.00	1.00	1.00
	Linear Regresion	100%	1.00	1.00	1.00	1.00
	Naïve Bayes	100%	1.00	1.00	1.00	1.00
	KNN	100%	1.00	1.00	1.00	1.00
	Random Forest	99%	0.99	0.99	1.00	0.996
	DNN	100%	1.00	1.00	1.00	1.00
	B-LSTM	99%	0.98	0.98	0.99	0.989
UDP	SVM	96%	0.94	0.90	0.95	0.952
	Linear Regression	94%	0.93	0.92	0.93	0.939
	Naïve Bayes	68%	0.54	0.47	0.64	0.735
	KNN	98%	0.97	0.98	0.98	0.980
	Random Forest	98%	0.98	0.97	0.98	0.979
	DNN	98%	0.97	0.95	0.98	0.976
	B-LSTM	98%	0.95	0.98	0.96	0.982
ICMP	SVM	99%	0.99	0.99	0.99	0.993
	Linear Regresion	99%	0.99	0.99	1.00	0.995
	Naïve Bayes	99%	0.99	0.99	1.00	0.995
	KNN	99%	0.98	0.99	0.99	0.990
	Random Forest	99%	0.98	0.99	0.99	0.988
	DNN	99%	0.99	0.99	1.00	0.99
	B-LSTM	100%	1.00	1.00	1.00	1.00
HTTP	SVM	99%	0.98	0.97	0.99	0.986
	Linear Regresion	97%	0.96	0.93	0.96	0.960
	Naïve Bayes	99%	0.98	0.98	0.99	0.991
	KNN	98%	0.98	0.97	0.98	0.981
	Random Forest	99%	0.97	0.97	0.99	0.986
	DNN	99%	0.98	0.98	0.99	0.987

	B-LSTM	99%	0.90	0.99	0.95	0.995
--	--------	-----	------	------	------	-------

Berikut merupakan *confusion matrix* dari model latih *machine learning* yang digunakan berdasarkan protokol:

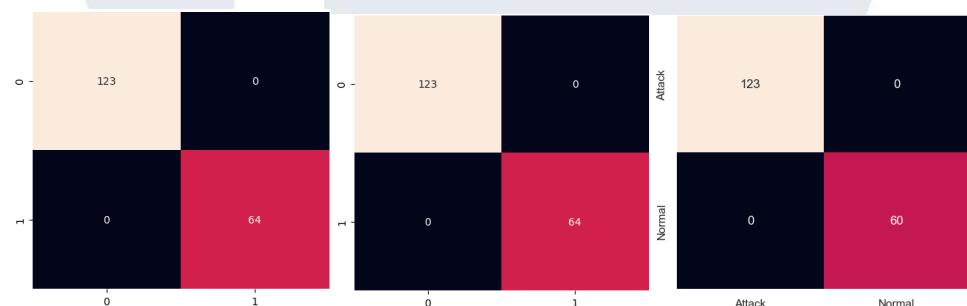
1. TCP



(SVM)

(Random Forest)

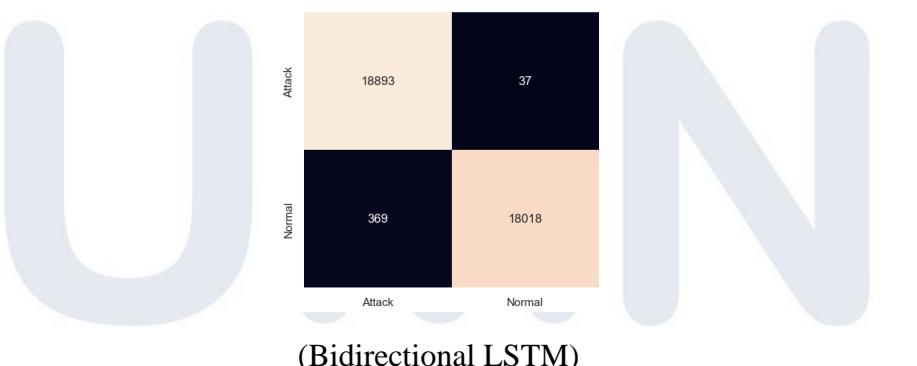
(Naïve Bayes)



(Linear Regresion)

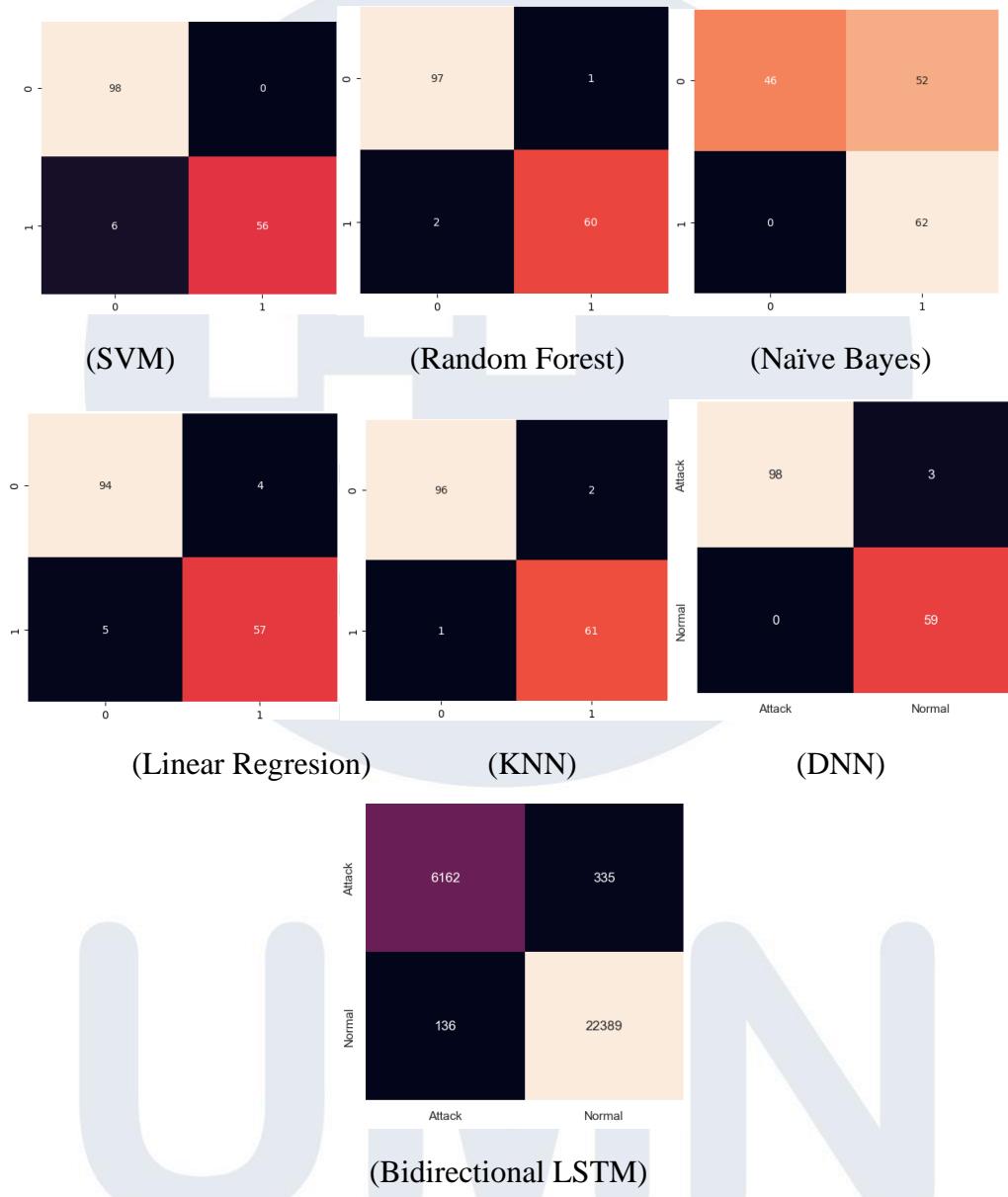
(KNN)

(DNN)



Gambar 4.56 *Confusion Matrix Machine Learning Protokol TCP*

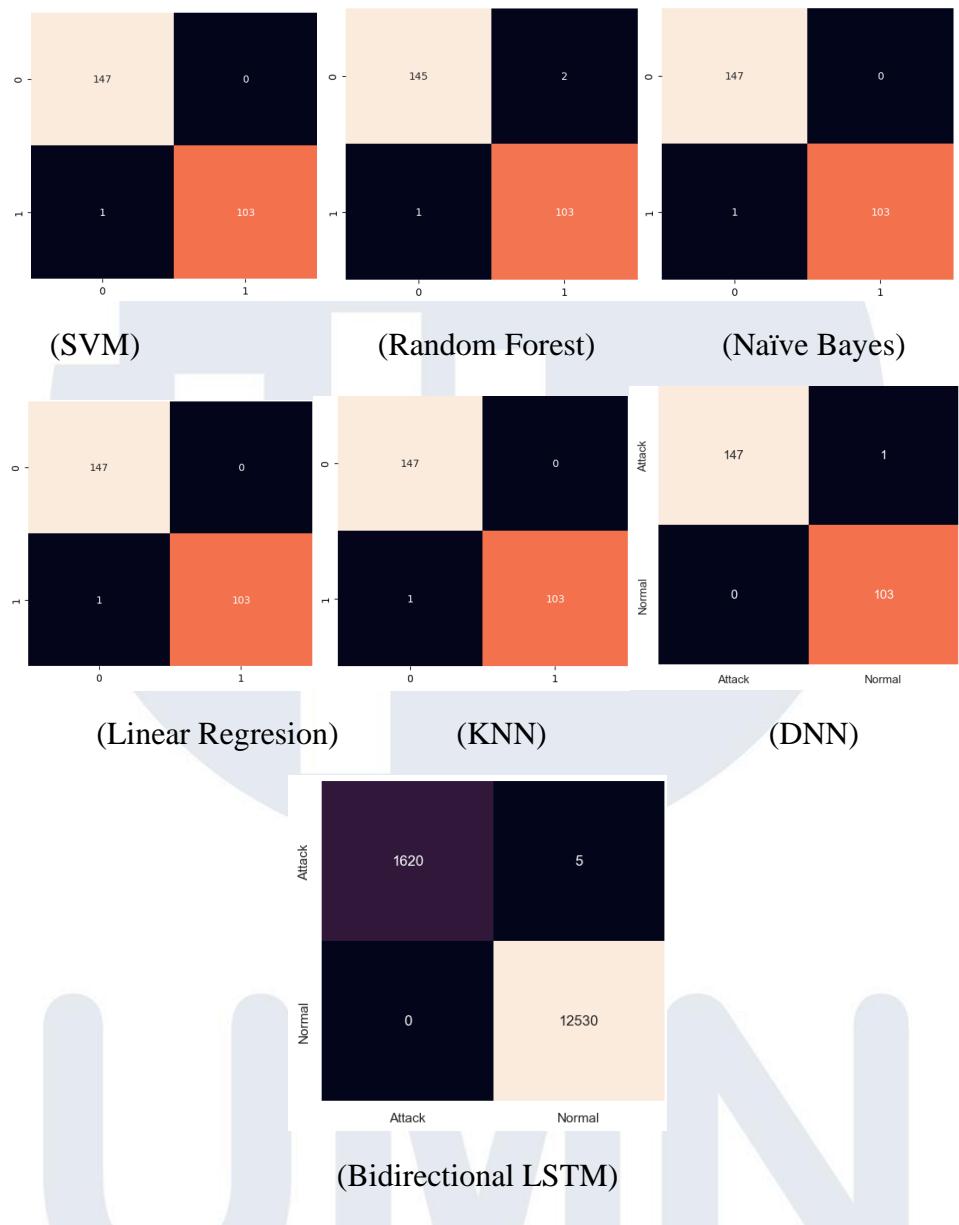
2. UDP



Gambar 4.57 Confusion Matrix Machine Learning Protokol UDP

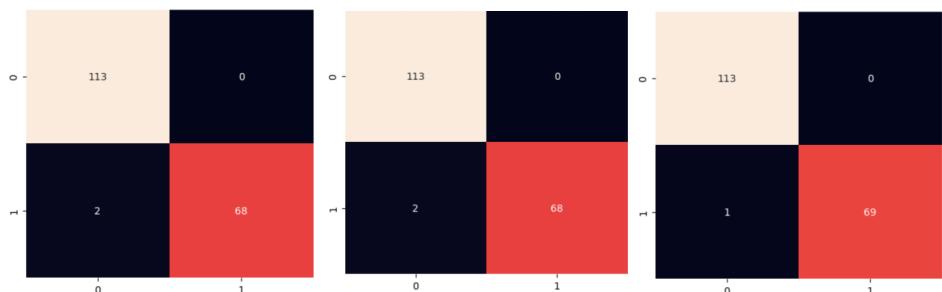
UNIVERSITAS
MULTIMEDIA
NUSANTARA

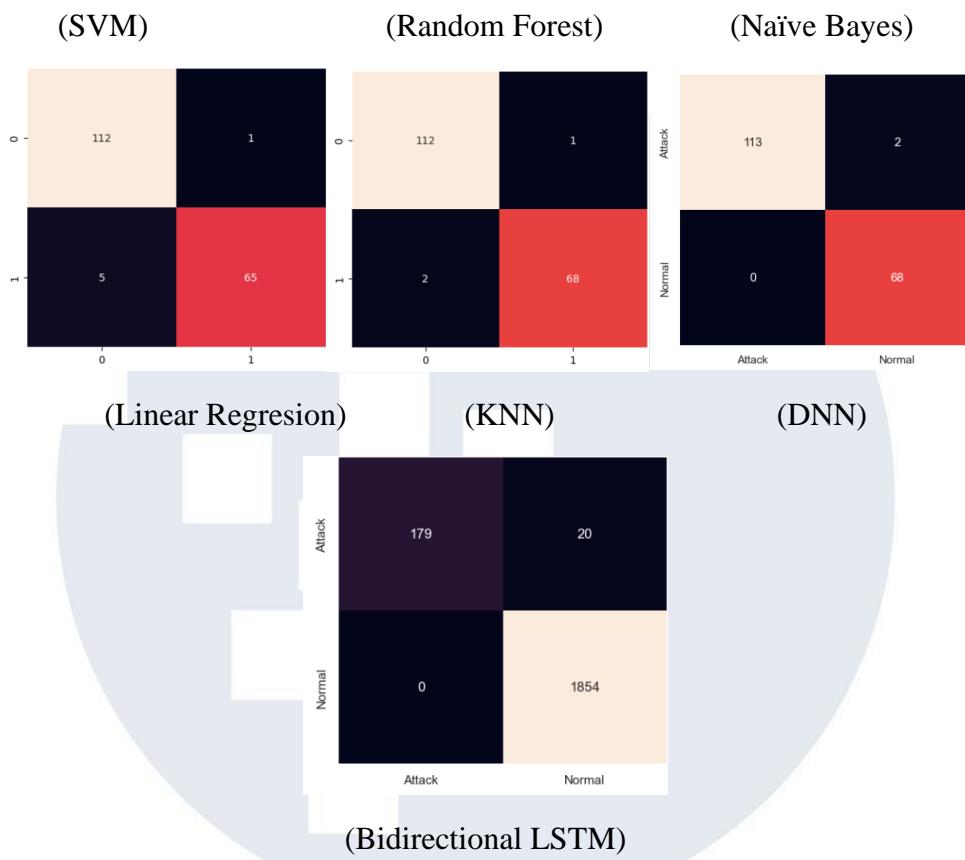
3. ICMP



Gambar 4.58 *Confusion Matrix Machine Learning* Protokol ICMP

4. HTTP





Gambar 4.59 *Confusion Matrix Machine Learning* Protokol HTTP

Jika dilihat pada diagram *confusion matrix* dan hasil akurasi tabel 4.4, dapat dianalisis bahwa untuk protokol TCP menghasilkan akurasi hampir 100% untuk semua algoritma *machine learning*-nya. Hal ini belum tentu menjadi penilaian positif karena bisa saja terjadi *overfitting* atau ketidaksesuaian dengan keadaan aslinya akibat kualitas dataset yang kurang baik sesuai dengan analisis 4.4.1.1. Penulis memiliki analisis yang mengakibatkan *overfitting* adalah bahwa dataset yang dipakai memiliki jumlah yang terlalu sedikit dan memiliki karakteristik yang sama antara semua datanya termasuk data latih dan validasi sehingga tidak mencerminkan pola stokastik yang acak dan tidak statis, sehingga tidak mewakili semua kemungkinan nilai data input secara akurat. Namun faktor lain seperti kemungkinan terdapat *noise* seperti pemilihan *features* yang buruk, data sintetis yang tidak diinginkan masuk ke dalam dataset validasi, dan ketimpangan nilai antar dataset dipastikan oleh penulis tidak terjadi akibat *pre-processing* data yang dilakukan oleh penulis. Namun rata-rata nilai akurasi dari model *machine learning*

ini dapat mencapai 95% ke atas, hal ini menandakan *features* dan data *pre-processing* yang digunakan pada dataset penelitian efektif untuk melakukan klasifikasi pada pola serangan DDoS.

Dari tabel 4.4, dapat dianalisis bahwa setiap model *machine learning* memiliki kelebihan dan kekurangannya sendiri-sendiri sesuai dengan karakteristik model *machine learning* tersebut pada setiap protokol jaringan komputer. Daftar model *machine learning* yang terbaik adalah sebagai berikut:

1. TCP
 - a. Terbaik : SVM, Linear Regresion, Naive Bayes, KNN, DNN
 - b. Buruk : Bidirectional LSTM
2. UDP
 - a. Terbaik : Bidirectional LSTM
 - b. Buruk : Naïve Bayes
3. ICMP
 - a. Terbaik : Bidirectional LSTM dan Linear Regresion
 - b. Buruk : KNN
4. HTTP
 - a. Terbaik : Bidirectional LSTM
 - b. Buruk : Linear Regresion

Jika dilihat pada hasil pada tabel 4.4, bisa dianalisis bahwa untuk mendeteksi serangan DDoS tidak diperlukan *neural network* yang kompleks dan bisa diambil *features* (data penting) melalui kalkulasi matematis, hal ini ditunjukkan dengan tingkat akurasi dari beberapa algoritma *machine learning* sudah memiliki akurasi yang cukup tinggi. Namun analisis ini hanya berlaku pada dataset penelitian ini, karena jika jumlah dataset semakin tinggi atau rendah dan juga terjadi perbedaan karakteristik dataset karena perbedaan skenario pengambilan dataset maka perbandingan hasil akurasi antara algoritma *machine learning* dan *deep learning* bisa saja akan berbeda.

3. Analisis performa *machine learning* berdasarkan waktu prediksi model.

Waktu prediksi model didapatkan dari perangkat laptop yang menjalankan

*library SKLearn dan Tensorflow, akan terjadi perbedaan antara waktu prediksi yang dihasilkan dengan *virtual machine*.*

Tabel 4.5 Perbandingan Hasil Performa Machine Learning dengan Time Prediction

Protokol	Model Name	Time Prediction (dalam Detik)
TCP	SVM	0.0005204000
	Linear Regresion	0.0005877000
	Naïve Bayes	0.0004234999
	KNN	0.0009485000
	Random Forest	0.0013955000
	DNN	0.0303047999
UDP	B-LSTM	0.0005006220
	SVM	0.0004423999
	Linear Regresion	0.0003404000
	Naïve Bayes	0.0005133000
	KNN	0.0011502999
	Random Forest	0.0012358999
ICMP	DNN	0.0481936999
	B-LSTM	0.0019722555
	SVM	0.0006876999
	Linear Regresion	0.0004865999
	Naïve Bayes	0.0006786000
	KNN	0.0012739999
HTTP	Random Forest	0.0010935999
	DNN	0.0413103000
	B-LSTM	0.0004796700
	SVM	0.0003829999
	Linear Regresion	0.0003351000
	Naïve Bayes	0.0003728000
	KNN	0.0009500999
	Random Forest	0.0014533999
	DNN	0.0335450000

	B-LSTM	0.0005001975
--	--------	--------------

Dari tabel 4.5, dapat dianalisis bahwa setiap model *machine learning* memiliki perbedaan kecepatan dalam waktu prediksi. Untuk model *deep learning* memiliki waktu terlama, hal ini karena instruksi *deep learning* lebih kompleks karena terdapat beberapa neural layer di dalamnya. Namun untuk kecepatan prediksi Bidirectional LSTM ternyata menghasilkan waktu yang cukup cepat jika dibandingkan instruksi *machine learning* sederhana. Untuk algoritma Random Forest memiliki waktu yang cukup lama jika dibandingkan algoritma *machine learning* lainnya karena terdapat beberapa iterasi pada setiap pohon keputusannya. Dan algoritma KNN memiliki waktu yang cukup lama untuk komputasi *Euclidean distance*.

- Analisis performa metode identifikasi serangan DDoS menggunakan *classification report*, waktu prediksi, dan response time sistem. Metode karakteristik paket terbanyak didapatkan nilai akurasi dari jumlah data *attack* dibandingkan jumlah data yang sama dengan data *signature*. Untuk dataset disimulasikan menggunakan 200 data normal dan 200 data serang dari *timeseries* dataset. Diambil data 200 karena dalam implementasinya data normal dan data serang minimalnya memiliki jumlah sebanyak 200 paket.

Tabel 4.6 Perbandingan Hasil Performa Machine Learning Identifikasi dengan Classification Matrix

Protokol	Model Name	Akurasi	precision	recall	F-1	AUC-ROC
TCP	SVM	100%	1.00	1.00	1.00	1.00
	Linear Regresion	100%	1.00	1.00	1.00	1.00
	Naïve Bayes	100%	1.00	1.00	1.00	1.00
	KNN	100%	1.00	1.00	1.00	1.00
	Random Forest	100%	1.00	1.00	1.00	1.00

	Metode karakteristik paket terbanyak	100%				
UDP	SVM	100%	1.00	1.00	1.00	1.00
	Linear Regresion	100%	1.00	1.00	1.00	1.00
	Naïve Bayes	100%	1.00	1.00	1.00	1.00
	KNN	100%	1.00	1.00	1.00	1.00
	Random Forest	100%	1.00	1.00	1.00	1.00
	Metode karakteristik paket terbanyak	100%				
ICMP	SVM	99%	0.98	0.97	0.99	0.988
	Linear Regresion	99%	0.98	0.97	0.99	0.988
	Naïve Bayes	100%	1.00	1.00	1.00	1.00
	KNN	99%	0.98	0.97	0.99	0.988
	Random Forest	100%	1.00	1.00	1.00	1.00
	Metode karakteristik paket terbanyak	98,5%				

Dari tabel 4.6, dapat dianalisis bahwa algoritma *machine learning* dapat secara efektif dalam nilai akurasi untuk melakukan identifikasi paket DDoS dan normal dengan *features* yang digunakan dan jumlah dataset yang sedikit (400 data). Untuk metode karakteristik paket terbanyak ternyata untuk protokol ICMP memiliki kekurangan 1 data, yaitu ada 1 data serang yang tidak dapat deteksi menggunakan metode ini. Untuk hasil ini tidak mencerminkan performa klasifikasi pada kondisi nyata, karena data input pasti akan lebih bervariasi dan acak jika dibandingkan dataset yang digunakan oleh penulis untuk melakukan uji coba sehingga didapatkan hasil yang *overfitting*. Hasil ini hanya sebagai referensi penting algoritma *machine learning* yang akan digunakan nantinya.

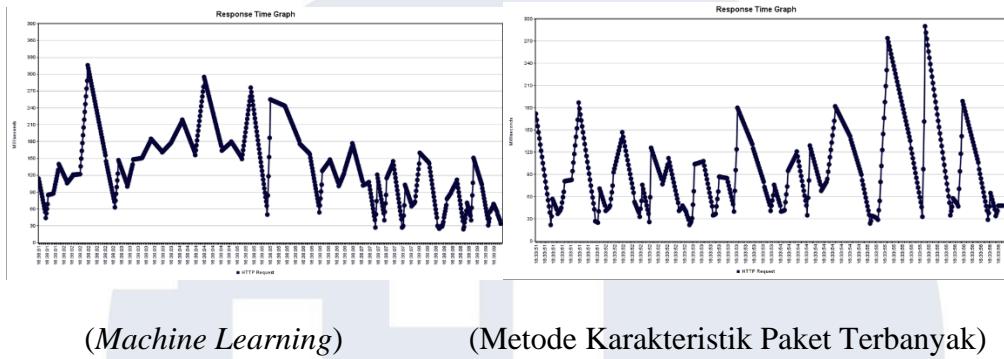
Tabel 4.7 Perbandingan Hasil Performa Machine Learning Identifikasi dengan Waktu Prediksi

Protokol	Model Name	Time Prediction (dalam Detik)	Time Training	Total
TCP	SVM	0.000375000	0.002963999	0,00333900
	Linear Regresion	0.000395299	0.009838900	0,01023420
	Naïve Bayes	0.000472800	0.002376500	0,00284930
	KNN	0.001233900	0.002192699	0,00342660
UDP	Random Forest	0.001451199	0.008197899	0,00964910
	SVM	0.000636899	0.002167899	0,00280480
	Linear Regresion	0.000504100	0.010183000	0,01068710
	Naïve Bayes	0.000379899	0.001750200	0,00213010
ICMP	KNN	0.001021900	0.001416499	0,00243840
	Random Forest	0.001584499	0.008497300	0,01008180
	SVM	0.000357600	0.002288300	0,00264590
	Linear Regression	0.000323400	0.008867799	0,00919120
	Naïve Bayes	0.000529100	0.001932299	0,00246140
	KNN	0.000907000	0.001611299	0,00251830
	Random Forest	0.001178299	0.007569200	0,00874750

Dari tabel 4.7, dapat dianalisis bahwa untuk mendeteksi serangan DDoS setiap algoritma *machine learning* memiliki kelebihan dan kekurangannya masing-masing. Namun karena untuk *machine learning* identifikasi akan dilakukan *training* secara *realtime* dan berulang ulang, oleh karena itu perlu dicari antara waktu prediksi dan waktu *training* paling kecil. Didapatkan model *machine learning* naïve bayes mendapatkan total waktu paling sedikit, hal ini akan berefek pada *time to mitigate* atau waktu paling kecil untuk serangan DDoS mulai dihentikan. Merujuk pada hasil penelitian tabel 4.6, algoritma Naïve bayes juga memiliki performa akurasi yang sempurna pada setiap protokol jaringan.

Berikutnya penulis juga menguji *response time* kedua metode pada saat terjadi serangan DDoS sehingga sistem akan mengidentifikasi satu persatu paket

yang masuk ke dalam sistem. Untuk grafik *response time* dapat dilihat pada gambar 4.60.



Gambar 4.60 Hasil Response Time Metode Identifikasi Paket

Keterangan :

1. *Machine learning* : Max 316 ms, Min 24 ms, *Throughput* 8,8 *request/sec*
2. Metode Karakteristik Paket Terbanyak : Max 290 ms, min 22 ms, *Throughput* 12,7 *request/sec*

Dari gambar hasil penelitian di atas dapat dianalisis bahwa proses komputasi dari metode karakteristik paket terbanyak lebih cepat karena lebih banyak memproses paket dalam satuan waktu walaupun hanya selisih 4 paket per detik. Hal ini menunjukkan bahwa walaupun proses komputasi metode karakteristik paket terbanyak lebih berat dari sisi penggunaan CPU namun lebih cepat memproses setiap paketnya, hal ini berarti dapat melakukan efisiensi dari algoritma *machine learning* agar menggunakan CPU yang lebih maksimal.

5. Analisi perbandingan metode machine learning dengan waktu untuk mengambil kesimpulan dan rata rata penggunaan CPU.

UNIVERSITAS
MULTIMEDIA
NUSANTARA

Tabel 4.8 Perbandingan Hasil Performa Machine Learning dengan Metode yang Berbeda

Protokol	Metode	Waktu untuk mengambil kesimpulan	Rata rata penggunaan CPU
TCP	Deteksi individual, identifikasi machine learning	0.26734	19.35%
	Deteksi individual, identifikasi karakteristik paket terbanyak	0.12103	19.56%
	Deteksi timeseries, identifikasi machine learning	4.57709	27.87
	Deteksi timeseries, identifikasi karakteristik paket terbanyak	3.44679	28.02%
UDP	Deteksi individual, identifikasi machine learning	0.46057	18.23%
	Deteksi individual, identifikasi karakteristik paket terbanyak	0.15687	19.10%
	Deteksi timeseries, identifikasi machine learning	4.15135	25.95%
	Deteksi timeseries, identifikasi karakteristik paket terbanyak	3.97331	26.23%
ICMP	Deteksi individual, identifikasi machine learning	0.38940	20.20%
	Deteksi individual, identifikasi karakteristik paket terbanyak	0.14384	20.94%
	Deteksi timeseries, identifikasi machine learning	4.16244	27.43%
	Deteksi timeseries, identifikasi karakteristik paket terbanyak	3.64641	28.82%
HTTP	Deteksi individual	0.094384	21.03%
	Deteksi timeseries	3.77370	29.12%

Waktu pengambil kesimpulan adalah waktu yang diperlukan dari awal *data parsing* sampai paket *signature* siap untuk dikomparasi dengan paket baru yang masuk. Untuk perbandingan antara metode *machine learning* deteksi serangan DDoS individual dan *timeseries* dalam rata rata penggunaan CPU akan lebih efisien untuk menggunakan metode *machine learning* individual. Hal ini karena jika dalam

kondisi jaringan ramai maka untuk metode individual akan tetap mendeteksi setiap 5 detik, namun untuk metode *timeseries* akan mengikuti jumlah paket yang masuk yaitu seberapa cepat 200 paket terpenuhi.

Untuk perbandingan identifikasi paket DDoS ternyata lebih efisien jika menggunakan *machine learning*. Hal ini ditunjukkan dengan angka rata-rata penggunaan CPU yang lebih rendah di ke 3 protokol. Untuk contoh tampilan saat melakukan pengambilan data rata-rata penggunaan CPU dapat dilihat pada gambar 4.61. Namun dalam sisi waktu pengambilan kesimpulan tercepat didapatkan dengan metode karakteristik paket terbanyak, hal ini karena metode karakteristik paket terbanyak hanya melakukan instruksi untuk mendapatkan nilai terbanyak, namun metode ini akan lebih besar proses komputasinya saat mengkomparasi setiap paketnya jika dilihat pada rata rata penggunaan CPU.

```
(base) brandondani@brandondani-virtual-machine:~/tugasakhir/IntelligentProxy/stuff$ python CPU_MONITOR_MEAN.py
CPU Usage: 23.10% | Memory Usage: 58.50% | Average CPU Usage: 19.35%
```

Gambar 4.61 Tampilan Pengambilan Data Rata Rata Penggunaan CPU

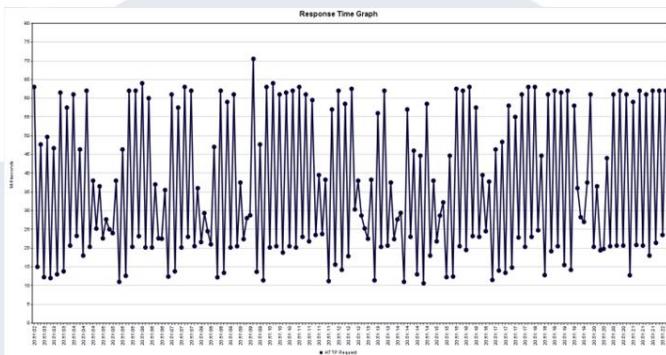
4.3.2 Analisis Hasil Evaluasi Sistem Reverse Proxy

Penulis ingin membandingkan *response time* dari penggunaan Sistem Proxy berbasis *machine learning* untuk deteksi DDoS dengan tidak menggunakan proxy. Hasil penelitian ini didapatkan dari aplikasi Jmeter. Hal ini memiliki tujuan untuk mengetahui apakah dengan menggunakan proxy akan berdampak pada performa *website* dan mempengaruhi kinerja *website*. Akan ada 3 skenario yaitu 700 *request* dengan 1 *concurrent connection*, 700 *request* dengan 10 *concurrent connection*, dan 350 *request* dengan 50 *concurrent connection*. Semua skenario menggunakan 7 *endpoint* berbeda.

UNIVERSITAS
MULTIMEDIA
NUSANTARA

1. 700 request dengan 1 concurrent connection

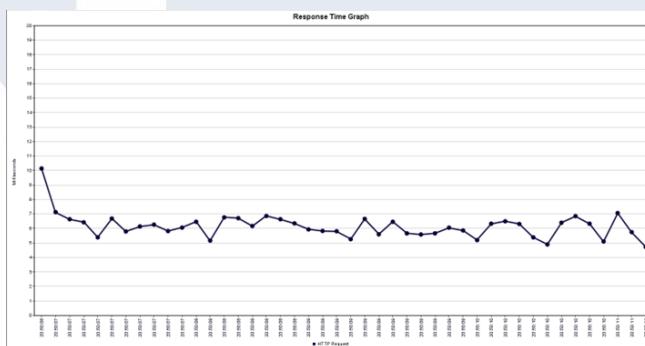
a. Menggunakan proxy



Gambar 4.62 Hasil Penelitian Uji Proxy 1

Keterangan : maximum 71 ms, minimum 10 ms, *Throughput* 35,8 request/detik

b. Tidak menggunakan proxy

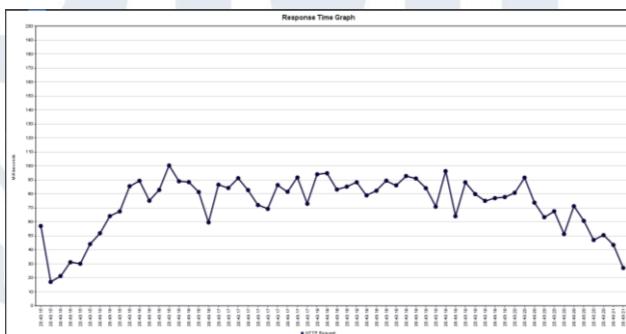


Gambar 4.63 Hasil Penelitian Uji Proxy 2

Keterangan : maksimum 10 ms, minimum 5 ms, throughput 158,1 request/detik

2. 700 request dengan 10 concurrent connection

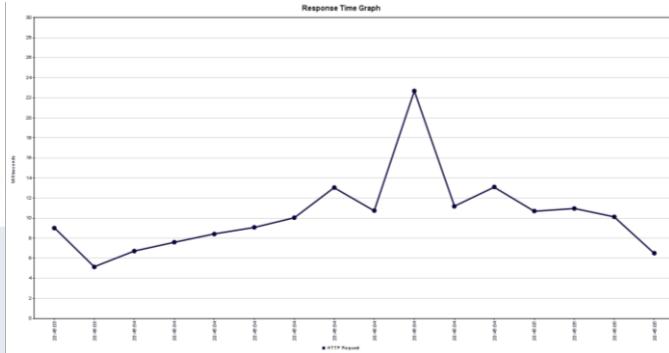
a. Menggunakan proxy



Gambar 4.64 Hasil Penelitian Uji Proxy 3

Keterangan : maksimum 100ms, minimum 15ms, throughput 115,6 request/detik

b. Tidak menggunakan proxy

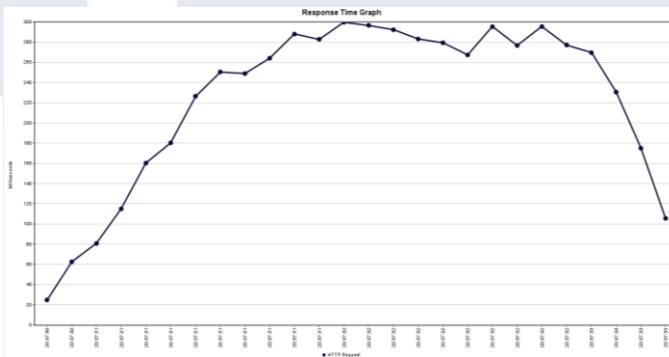


Gambar 4.65 Hasil Penelitian Uji Proxy 4

Keterangan : maksimum 23ms, minimum 5ms, throughput 442,2 request/detik

3. 350 request dengan 50 concurrent connection

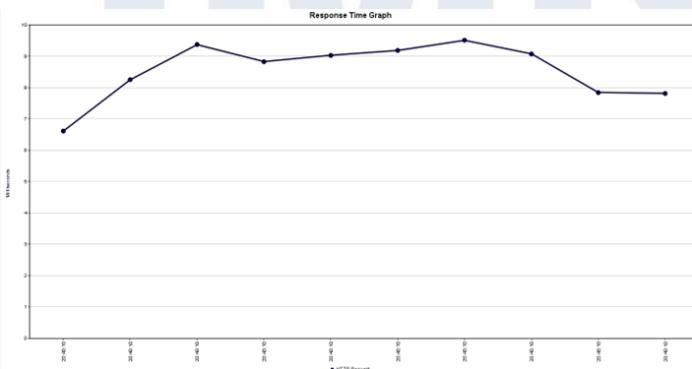
a. Menggunakan proxy



Gambar 4.66 Hasil Penelitian Uji Proxy 5

Keterangan : maksimum 300ms, minimum 21ms, throughput 130,3 request/detik

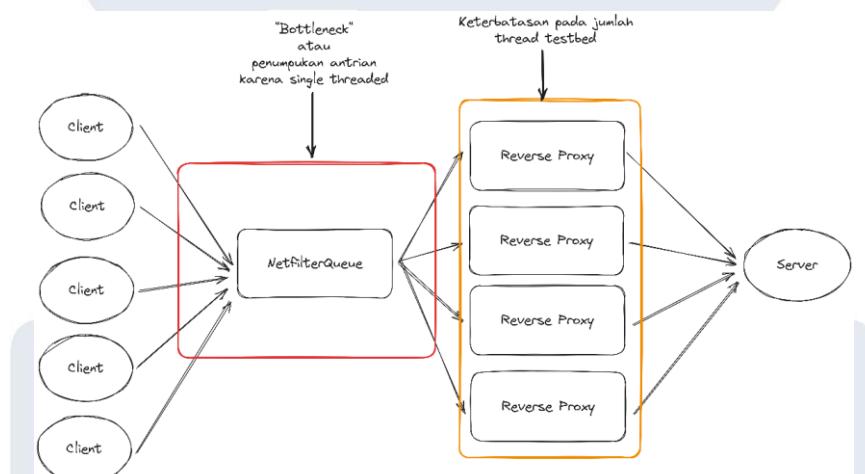
b. Tidak menggunakan proxy



Gambar 4.67 Hasil Penelitian Uji Proxy 6

Keterangan : maksimum 9,5ms, minimum 6,5ms, throughput 336,2 request / detik

Dari hasil penelitian, performa proxy tidak terlalu bagus jika dibandingkan dengan tidak menggunakan proxy. Untuk nilai *throughput*, dengan proxy hanya mendapatkan setengah koneksinya saja. Dan untuk *latency* bisa naik 30 kali lipat jika menggunakan proxy. Hal ini menjadi bahan penelitian berikutnya, karena pengamatan penulis untuk *library netfilterQueue* hanya memakai 1 thread saja sebagai modul terdepan untuk firewall comparator, jadi walaupun *reverse proxy* sudah memakai paradigma *multithreading* namun keterbatasan ada di bagian *firewall controller*. Mungkin bisa diganti library yang mendapatkan dari IP tables namun menggunakan *multithread* agar performa *concurrent user*nya meningkat. Namun karena pada 1 *concurrent connection*-nya juga sudah mendapatkan penurunan performa maka terjadi indikasi bahwa terjadi pemrosesan yang tidak efisien pada sistem *reverse proxy*. Untuk ilustrasinya dapat dilihat pada gambar 4.68.



Gambar 4.68 Ilustrasi Analisis Evaluasi *Reverse Proxy*

4.3.3 Analisis Hasil Evaluasi Sistem Distribusi Data

- Waktu yang diperlukan untuk data tersebar ke seluruh *node*

Uji coba dilakukan dengan mengerang salah satu node dan melakukan *logging* data pada node tersebut yaitu waktu saat mengirim data ke backend dan dilakukan *logging* juga pada *node* ke 2 waktu dari *file firewall rule* berganti.

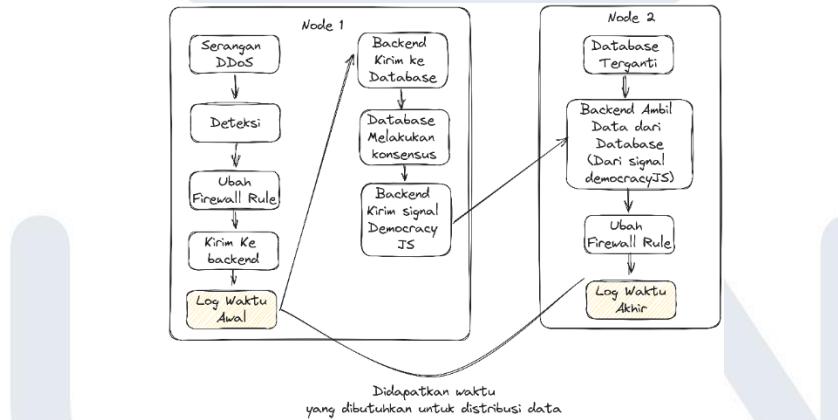
```

brandondani@brandondani-virtual-machine: ~/tugasakhir/int... brandondani@brandondani-virtual-machine: ~/tugasakhir/intelligentProxy/backend$ python3.10 dist-packages/sklearn/base.py:318: UserWarning: Trying to unpickle estimator StandardScaler from version 1.0.2 when using version 1.2. This might lead to breaking code or invalid results. Use at your own risk. For more information, please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
Model-proxy is ready, creating proxy thread
creating proxy: proxy1
  type id  privPolymorphic listenport   ip_addr
0   1   0   RoundRobin    5000  192.168.29.137
('RoundRobin': <ertools.cycle object at 0x7f5e5ed340>)
creating proxy: proxy2
  type id  privPolymorphic listenport   ip_addr
0   0   0   RoundRobin    5000  192.168.29.138
('RoundRobin': <ertools.cycle object at 0x7f5e5edaa0>)
HTTP timeseries result : 1, IP : 192.168.29.136, prediction time : 0.003704470800
00024022
Modify firewall rules to banned ip 192.168.29.136
sending to backend, time : 2023-06-12 00:11:39.872107
brandondani@brandondani-virtual-machine: ~/tugasakhir/intelligentProxy/backend$ sudo iptables -I INPUT -d 192.168.29.0/24 -j NFQUEUE --queue-num 1
(base) brandondani@brandondani-virtual-machine: ~/tugasakhir/intelligentProxy/backend$ cd ..
(base) brandondani@brandondani-virtual-machine: ~/tugasakhir/intelligentProxy$ cd proxy
(base) brandondani@brandondani-virtual-machine: ~/tugasakhir/intelligentProxy/proxy$ sudo python FirewallServer_only.py
(base) brandondani@brandondani-virtual-machine: ~/tugasakhir/intelligentProxy/proxy$ ./FirewallServer_only.py
model-firewall is ready, creating traffic listener
Prefix of 172.253.118.94 is banned by the firewall.
FirewallRules.json has been modified, time : 2023-06-12 00:11:40.540448
FirewallRules.json has been modified, time : 2023-06-12 00:11:40.543198

```

Gambar 4.69 Hasil Penelitian Uji Sistem Distribusi 1 (kiri node 1, kanan node 2)

Terlihat bahwa waktu yang dibutuhkan untuk mengirim data yaitu kurang dari 1 detik atau tepatnya 0,67 detik. Hal ini membuktikan bahwa waktu konsensus bigchainDB sangat cepat ditambah dengan efektivitas DemocracyJS untuk langsung melakukan pergantian data pada *node* lain saat data berhasil dimasukkan ke *database*. Untuk alur pengambilan data hasil penelitian pada tahap ini dapat dilihat pada gambar berikut :



Gambar 4.70 Alur Kerja Evaluasi Sistem Distribusi Data

2. Response time dari backend untuk menambah data ke blockchain

Dilakukan uji coba untuk menambah data ke API backend dan diperhatikan waktu yang diperlukan untuk memproses permintaan data. Didapatkan rata rata *response time* yaitu 432ms setiap transaksinya. Dengan begitu untuk *throughput* dari bigchainDB di sistem yang dibangun yaitu setidaknya 2 transaksi per detik.

```

{
  "keypair": {
    "publicKey": "J3qMKmpasPPgnvZiQV8oXzNNXqtbgVojiKKJ7eHdu1",
    "privateKey": "hfvMY9yDqcKTNQ0ir9kPUF8VmflxCJ7u3quF1yam"
  }
}

```

Status: 200 OK Time: 432 ms Size: 2.23 KB

Gambar 4.71 Hasil Penelitian Uji Sistem Distribusi 2

3. Uji coba menggunakan *public key* yang salah

Penulis menguji coba untuk kemampuan *immutability* dari bigchainDB dengan melakukan pengujian jika user mengirimkan *publickey* yang salah. Hasilnya adalah *database* akan memunculkan *error* dan karena backend buat dengan metode *try and catch* maka backend bisa menangani *error* tanpa aplikasi berhenti. Untuk hasil dari uji coba *publickey* yang salah dapat dilihat pada gambar 4.72. Hal ini baik, karena jika pertahanan pertama (JWT) diambil menggunakan *cookie* atau serangan lainnya, maka penyerang tetap tidak akan bisa untuk mengubah data di *database* dengan *keypair* (*publicKey* dan *privateKey*) yang salah. Dengan catatan *keypair* perlu disimpan oleh user secara terpisah contoh pada suatu file, AWS parameter store, dll.

```

{
  "keypair": {
    "publicKey": "invalidPublicKey",
    "privateKey": "hfvMY9yDqcKTNQ0ir9kPUF8VmflxCJ7u3quF1yam"
  },
  "ipAddress": "192.168.1.14",
  "source": "heho"
}

```

Status: 500 Internal Server Error Time: 104 ms Size: 514 B

```

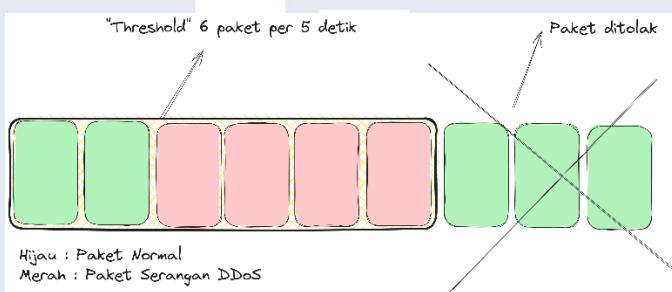
{
  "message": {
    "message": "Cannot read properties of undefined (reading 'data')"
}

```

Gambar 4.72 Hasil Penelitian Uji Sistem Distribusi 3

4.3.4 Analisis Hasil Perbandingan dengan IDS / IPS Snort

Setelah penulis mencoba untuk menggunakan dan membuat rule dari aplikasi Snort, penulis merasa bahwa cara kerja dari sistem yang dibuat oleh penulis dan aplikasi Snort memiliki perbedaan mendasar. Sebabnya untuk aplikasi Snort melakukan deteksi menggunakan sistem “threshold”, hal ini membuat aplikasi Snort seperti *rate limiter* pada suatu sistem. Cara kerja dari aplikasi Snort dapat dilihat pada gambar 4.73.



Gambar 4.73 Hasil Penelitian Uji Sistem Distribusi 3

Rate limiter adalah sebuah mekanisme yang digunakan untuk mengontrol jumlah akses atau *request* yang diberikan dalam periode waktu tertentu. Jika akses sudah melebihi batas yang telah ditentukan, maka akses akan ditolak sampai periodenya berganti tanpa mempertimbangkan faktor lain seperti tidak mengidentifikasi paket dari client normal atau penyerang. Dengan metode ini hanya mengatasi serangan DDoS secara sementara, tidak efektif, tidak keseluruhan, dan tidak akurat sehingga menimbulkan banyak *false positive*. Misalnya adalah ketika penulis menerapkan *rule* untuk HTTP GET hanya boleh 100 kali per 20 detik, namun ketika penulis mencoba melakukan akses HTTP sebanyak 110 kali dalam waktu 15 detik dengan jeda waktu acak, maka akan terdeteksi sebagai serangan DDoS karena jumlah paket HTTP sudah melebihi batasnya padahal yang dilakukan merupakan skenario yang normal. Oleh karena itu akan sulit untuk menentukan nilai “*threshold*” dalam suatu sistem yang diimplementasinya. Hal ini karena Snort hanya dapat diatur untuk memonitor 1 *feature* / data unik saja.

Dibanding dengan sistem yang penulis bangun, untuk mendeteksi suatu serangan memperhatikan beberapa *features* sekaligus. Misal 2 skenario

menghasilkan jumlah paket yang sama dalam 1 periode waktu yaitu 100 paket dalam 1 detik, namun pasti ada perbedaan lain dari 2 skenario tersebut yaitu misalkan variasi dari URL yang diakses. Jika kondisi normal memiliki variansi URL yang besar, namun jika serangan DDoS cenderung memiliki variansi URL yang kecil. Oleh karena itu diperlukan melihat beberapa data atau features sekaligus untuk mendeteksi serangan DDoS. Aplikasi Snort hanya melihat 1 data yang menyebabkan terjadinya false positive.

Pada tabel 4.9 dapat dilihat penulis membandingkan rata-rata penggunaan CPU saat sistem sedang diserang DDoS. Untuk sistem yang dibuat penulis dipakai metode yang paling ringan.

Tabel 4.9 Perbandingan Rata Rata Penggunaan CPU antara Sistem yang Dibuat dengan Aplikasi Snort

Protokol	Sistem yang dibuat penulis	Snort
TCP	19.35%	13.21%
UDP	18.23%	11.67%
ICMP	20.20%	14.24%
HTTP	21.03%	14.56%

Hasilnya adalah sistem Snort memiliki rata rata penggunaan CPU yang lebih kecil sekitar 5 – 10 % dari pada sistem yang penulis buat. Hal ini dikarenakan pada sistem yang penulis buat terdapat 2 model *machine learning* yang akan memprediksi terus menerus. Jika dibandingkan dengan aplikasi Snort yang tidak menggunakan komputasi berat.

Berikutnya penulis melakukan perbandingan nilai TTM (*Time to Mitigate*), penelitian ini dilakukan dengan mencatat waktu pemanggilan program DDoS dengan *script Python* dan mencatat waktu sampai paket pertama DDoS diblokir.

Tabel 4.10 Perbandingan Waktu TTM Sistem yang Dibuat dengan Aplikasi Snort

Protokol	Sistem yang dibuat penulis (dalam detik)	Snort (dalam detik)
TCP	3,3464	0,069117
UDP	3,799	0,082104

ICMP	3,088	0,065102
HTTP	4,243	0,050815

Terlihat bahwa dengan sama sama menggunakan metode menunggu paket Sampai berjumlah 200 paket. Untuk aplikasi Snort jauh lebih cepat karena tidak menggunakan algoritma *machine learning* sehingga hanya menunggu sampai kuota paket terpenuhi. Namun untuk sistem yang dibuat penulis diperlukan waktu untuk prediksi, pembuatan paket *signature* atau model *machine learning* identifikasi dan lain lain. Sebagai catatan pada penelitian ini penulis menggunakan *machine learning* deteksi serangan DDoS metode *timeseries* dengan metode identifikasi paket karakteristik paket terbanyak.

4.4 Kendala dan Solusi penelitian

1. Keterbatasan informasi tentang BigchainDB dan juga aplikasi yang *outdated* pada *repository* aslinya karena untuk *latest releasenya* tahun 2020, sehingga banyak *dependency* yang sudah tidak *support*. Mungkin untuk ke depannya lebih teliti lagi dalam memilih teknologi dan dipastikan masih diperbarui sampai sekarang.
2. Penulis tidak menentukan skop penelitian yang dibatasi dari awal penelitian sehingga penulis kesusahan untuk menentukan matriks pengujian dan mendalami salah satu analisis sistem.

UNIVERSITAS
MULTIMEDIA
NUSANTARA

BAB V

SIMPULAN DAN SARAN

5.1 Simpulan

Dari hasil penelitian yang sudah didapatkan penulis, penulis mendapatkan beberapa kesimpulan yaitu

1. Dengan menggunakan teknologi *machine learning* pada sistem proxy mitigasi DDoS didapatkan hasil akurasi deteksi serangan DDoS dengan nilai rata rata diatas 95% ke 7 model *machine learning* yang berbeda dengan menggunakan data *features* dan metode *pre-processing* seperti normalisasi dan *balancing* SMOTE yang digunakan oleh penulis. Namun masih terjadi *overfitting* pada dataset protokol TCP akibat penggunaan dataset yang kecil sehingga tidak mewakili semua kemungkinan data acak. Semua model *machine learning* didapatkan dengan waktu deteksi dan waktu pengambil kesimpulan yang cukup kecil yaitu sekitar 0.1 – 1 detik pada perangkat laptop penulis dan 3 – 4 detik pada *virtual machine*.
2. Setiap model *machine learning* memiliki perbedaan hasil karakteristik pada setiap protokol jaringan komputer dengan matriks akurasi dan waktu prediksi, sehingga penulis menyarankan untuk memakai model *machine learning* yang berbeda beda tiap protokol untuk mendapatkan performa *machine learning* terbaik.
3. Penulis menyarankan jika sistem ini diimplementasikan pada spesifikasi sistem yang rendah, digunakan metode *machine learning* individual dan identifikasi *machine learning*. Karena secara rata-rata penggunaan CPU merupakan yang paling kecil dan efisien daripada metode yang lainnya. Namun jika diimplementasikan pada sistem yang memiliki spesifikasi lebih tinggi maka penulis menyarankan menggunakan metode *machine learning timeseries* dengan identifikasi *machine learning* karena dengan menggunakan metode individual waktu TTMnya pasti akan sekitar 0 – 5 detik karena adanya waktu tidur pada algoritmanya, dan jika metode *timeseries* akan berkisar 3-4

detik dan bisa lebih kecil jika menggunakan sistem dengan performa CPU yang lebih tinggi.

4. Untuk metode identifikasi paket yang merupakan bagian dari serangan DDoS, penulis mendapatkan kesimpulan bahwa dengan menggunakan metode pendekatan karakteristik paket terbanyak yaitu mencari nilai mayoritas sebagai *signature* paket DDoS bisa mengurangi waktu *time to mitigate* sistem mitigasi dan *throughput* sistem *reverse proxy*, namun proses komputasinya lebih besar dari pada menggunakan metode *machine learning* dan jika ada teknik serangan lainnya bisa saja tidak berfungsi dengan baik seperti terjadi pada protokol ICMP. Disisi lain, algoritma *machine learning* dapat secara efektif dalam nilai akurasi untuk melakukan identifikasi paket yang termasuk serangan DDoS dengan *features* yang digunakan dan jumlah dataset yang sedikit (400 paket data). Namun karena pengujian hanya menggunakan dataset yang kecil, maka terjadi *overfitting* akibat kekurangan semua kemungkinan data acak sehingga hasil tidak mencerminkan performa klasifikasi pada kondisi nyata. Didapatkan *machine learning* algoritma *naïve bayes* mendapatkan waktu total latih dan prediksi paling kecil dari pada algoritma lainnya. Hasil ini hanya sebagai referensi penentu algoritma machine learning yang akan digunakan nantinya.
5. Penggunaan teknologi *database* terdistribusi berbasis *blockchain* dapat meningkatkan efektivitas sistem IPS dari sisi *time to mitigate*, dengan waktu sebar dan konsensus yaitu 0,67 detik, hal ini jauh lebih cepat daripada menggunakan *blockchain* Etherium sesuai referensi penelitian terdahulu 2.1.4 yaitu 96.95 detik.
6. Pengujian kemampuan *immutability* data ketika menggunakan keypair yang salah mendapatkan hasil bahwa data yang disimpan pada *database* BigchainDB memiliki kemampuan *immutability* dengan perilaku *database* mengembalikan *error* dan transaksi tidak terbuat
7. Performa *reverse proxy* tidak cukup bagus karena dalam kasus terburuknya dapat meningkatkan *latency* sebesar 30 kali, hal ini karena penggunaan *library* netfilterQueue pada sistem *firewall controller* yang menggunakan *single*

thread walaupun arsitektur sistem *proxy controller* sudah diterapkan paradigma *multithreading*.

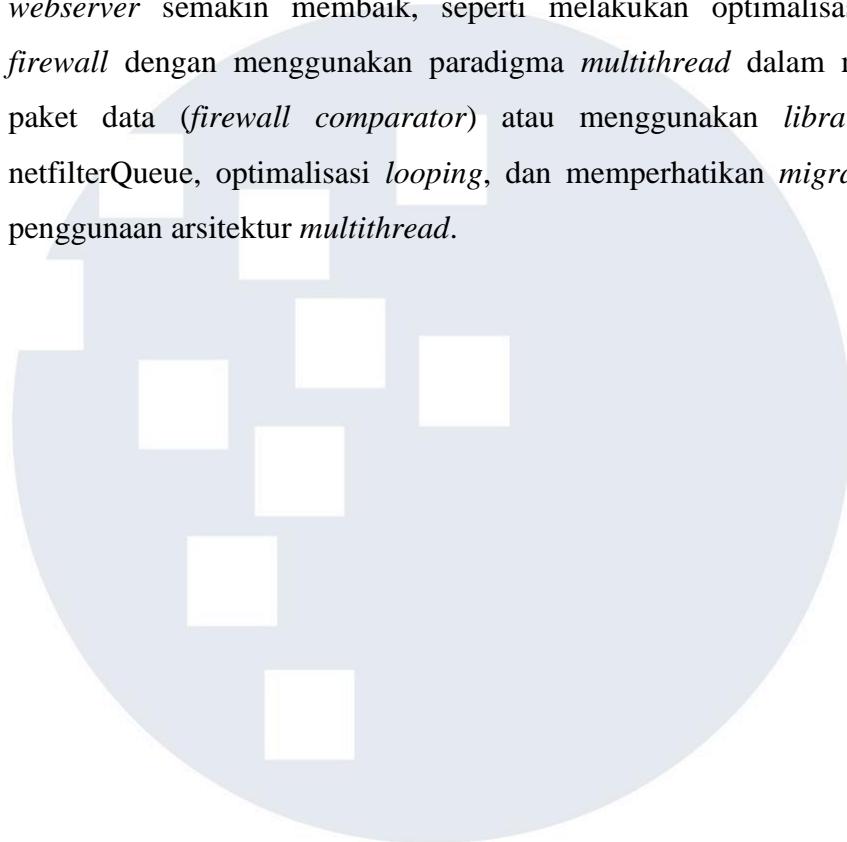
8. Penulis menyimpulkan bahwa sistem yang penulis bangun walaupun dengan waktu TTM dan penggunaan CPU yang lebih besar, namun sistem yang dibangun menunjukkan lebih efisien, keseluruhan, dan akurat dari sisi deteksi *false positive* IPS / IDS Snort, karena dengan sistem yang penulis bangun bisa dengan tetap memproses paket normal selagi memblokir serangan DDoS dan tingkat *false positive* yang kecil..

5.2 Saran

Berdasarkan penelitian yang telah dilakukan oleh penulis, terdapat beberapa saran untuk penelitian ini sebagai berikut :

1. Mencoba mengurangi waktu rangkum pada metode *machine learning* individual untuk mendapatkan nilai *time to mitigate* yang lebih kecil dibandingkan waktu rangkum 5 detik, sehingga didapatkan model *machine learning* yang lebih efisien secara proses komputasi dan nilai *time to mitigate* dibanding metode *timeseries* yang mendapatkan waktu rangkum 3 – 4 detik.
2. Melakukan penambahan dan pengambilan dataset dengan melakukan implementasi pada jaringan nyata, karena dataset yang didapatkan oleh penulis pada skenario buatan bisa saja tidak mencerminkan dengan skenario aslinya. Misalnya seperti perbedaan jumlah *client* yang mengakses, jumlah *connection rate* dalam satuan waktu, jenis serangan lain, kebiasaan kondisi normal yang berbeda, dan lain lain. Dan juga dataset yang digunakan penulis terjadi *overfitting* sesuai dengan analisis hasil evaluasi *machine learning*. Sehingga bisa menguji performa model *machine learning* pada kondisi dataset jaringan nyata.
3. Menggunakan suatu model data untuk menyebarkan *signature* paket serangan DDoS selain *IP address* pada sistem distribusi data, agar dapat digunakan pada protokol TCP, UDP, dan ICMP dan menghindari masalah *IP spoofing*. Seperti menyimpan model latih *machine learning* identifikasi di penyimpan terdistribusi IPFS.

4. Meningkatkan performa dari sistem *reverse proxy* agar *latency* akses *webserver* semakin membaik, seperti melakukan optimalisasi sistem *firewall* dengan menggunakan paradigma *multithread* dalam mengecek paket data (*firewall comparator*) atau menggunakan *library* selain netfilterQueue, optimalisasi *looping*, dan memperhatikan *migration cost* penggunaan arsitektur *multithread*.



UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA

DAFTAR PUSTAKA

- [1] F. A. Fadilla, “*Web Service*,”, Universitas Siliwangi, Maret 2020. [Online]. Tersedia: https://www.researchgate.net/publication/339676603_WEB_SERVICE_PENGERTIAN_WEB_SERVICE.
- [2] N. Tripathi and B. Mehtre, “*DoS and DDoS Attacks: Impact, Analysis and Countermeasures*,”, di Advances in Computing, Networking and Security, 2013 TEQIP II National Conference, 2013.
- [3] I. Cvitić, D. Peraković, M. Periša, and M. Botica, “*Novel approach for detection of IOT generated ddos traffic*,”, *Wireless Networks*, vol. 27, no. 3, pp. 1573–1586, 2019. doi:10.1007/s11276-019-02043-1.
- [4] Haryono and D. J. Ratnaningsih, “*Probabilitas*,”, Universitas Terbuka. [Online]. Tersedia: <https://pustaka.ut.ac.id/lib/wp-content/uploads/pdfmk/SATS4322-M1.pdf> (diakses 27 Mei 2023).
- [5] O. Yoachimik, “*DDoS attack trends for 2022 Q2*,”, The Cloudflare Blog. [Online]. Tersedia: <https://blog.cloudflare.com/ddos-attack-trends-for-2022-q2/> (diakses 16 Februari 2023).
- [6] M. Rezek, “*What is the difference between signature-based and behavior-based intrusion detection systems?*”, Accedian. [Online]. Tersedia: <https://accidian.com/blog/what-is-the-difference-between-signature-based-and-behavior-based-ids/> (diakses 16 Februari 2023).
- [7] Administrator, “*IDS signatures*,”, eTutorials.org. [Online]. Tersedia: <https://etutorials.org/Networking/Router+firewall+security/Part+VII+Detecting+and+Preventing+Attacks/Chapter+16.+Intrusion-Detection+System/IDS+Signatures/> (diakses 13 Februari 2023).
- [8] Administrator, “*Microservices in the enterprise, 2021: Real benefits, worth the challenges*,”, IBM Market Research. [Online]. Tersedia: <https://www.ibm.com/downloads/cas/OQG4AJAM> (diakses 14 Februari 2023).
- [9] Amarabha Banerjee et al., “*Are distributed networks and decentralized systems the same?*”, Packt Hub. [Online]. Tersedia: <https://hub.packtpub.com/are-distributed-networks-decentralized-systems-same/> (diakses 6 Juni 2023).

- [10] A. Gaurav, “*A Comprehensive Survey on DDoS Attacks on various Intelligent Systems and It’s Defense Techniques*,”, Mei 2022.
doi:<http://dx.doi.org/10.13140/RG.2.2.16769.12644>
- [11] J. Ye, X. Cheng, J. Zhu, L. Feng, and L. Song, “*A ddos attack detection method based on SVM in software defined network*,”, *Security and Communication Networks*, vol. 2018, pp. 1–8, 2018.
doi:[10.1155/2018/9804061](https://doi.org/10.1155/2018/9804061)
- [12] Y. Li and Y. Lu, “*LSTM-ba: Ddos detection approach combining LSTM and Bayes*,”, *2019 Seventh International Conference on Advanced Cloud and Big Data (CBD)*, 2019. doi:[10.1109/cbd.2019.00041](https://doi.org/10.1109/cbd.2019.00041)
- [13] B. Rodrigues, E. Scheid, C. Killer, M. Franco, and B. Stiller, “*Blockchain Signaling System (bloss): Cooperative signaling of distributed denial-of-service attacks*,”, *Journal of Network and Systems Management*, vol. 28, no. 4, pp. 953–989, 2020. doi:[10.1007/s10922-020-09559-4](https://doi.org/10.1007/s10922-020-09559-4)
- [14] S. Lupăiescu *et al.*, “*Centralized vs. decentralized: Performance comparison between bigchaindb and Amazon QLDB*,”, *Applied Sciences*, vol. 13, no. 1, p. 499, 2022. doi:[10.3390/app13010499](https://doi.org/10.3390/app13010499)
- [15] J. Cheng *et al.*, “*A ddos attack information fusion method based on CNN for multi-element data*,”, *Computers, Materials & Continua*, vol. 62, no. 3, pp. 131–150, 2020. doi:[10.32604/cmc.2020.06175](https://doi.org/10.32604/cmc.2020.06175)
- [16] R. Krishnamurthy and G. N. Rouskas, “*Performance evaluation of multi-core, multi-threaded SIP proxy servers (SPS)*,” *2016 IEEE International Conference on Communications (ICC)*, 2016.
doi:[10.1109/icc.2016.7511427](https://doi.org/10.1109/icc.2016.7511427)
- [17] Administrator, “*OSI model*,” Wikipedia. [Online]. Tersedia: https://en.wikipedia.org/wiki/OSI_model (diakses 10 Juni 2023).
- [18] Administrator, “*What is TCP/IP in networking?*,”, Fortinet. [Online]. Tersedia: <https://www.fortinet.com/resources/cyberglossary/tcp-ip> (diakses 10 Juni 2023).
- [19] K. Academy, “*Transmission control protocol (TCP) (article)*,”, Khan Academy. [Online]. Tersedia: <https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:the-internet/xcae6f4a7ff015e7d:transporting-packets/a/transmission-control-protocol--tcp> (diakses 10 Juni 2023).
- [20] K. Academy, “*User datagram protocol (UDP) (article)*,” Khan Academy. [Online]. Tersedia: <https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:the->

internet/xcae6f4a7ff015e7d:transporting-packets/a/user-datagram-protocol-udp (diakses 10 Juni 2023).

- [21] B. Lutkevich, “*What is ICMP (internet control message protocol): Definition from TechTarget*,”, Tech Target Networking. [Online]. Tersedia: <https://www.techtarget.com/searchnetworking/definition/ICMP> (diakses 10 Juni 2023).
- [22] Administrator, “*ICMP type and code IDs*,”, IBM. [Online]. Tersedia: <https://www.ibm.com/docs/en/qsip/7.4?topic=applications-icmp-type-code-ids> (diakses 10 Juni 2023).
- [23] R. Bhardwaj, “*HTTP VS TCP : Detailed comparison*,”, Network Interview. [Online]. Tersedia: <https://networkinterview.com/http-vs-tcp-know-the-difference/> (diakses 10 Juni 2023).
- [24] Administrator, “*What is ids and IPS: Juniper Networks Us*,”, Juniper Networks. [Online]. Tersedia: <https://www.juniper.net/us/en/research-topics/what-is-ids-ips.html> (diakses 10 Juni 2023).
- [25] S. Cooper, “*Snort review including alternatives*,”, Comparitech. [Online]. Tersedia: <https://www.comparitech.com/net-admin/snort-review/> (diakses 10 Juni 2023).
- [26] Administrator, “*What is a reverse proxy?*,”, Cloudflare. [Online]. Tersedia: <https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/> (diakses 11 Juni 2023).
- [27] Michael, “*Concurrent connections demystified*,”, LiteSpeed Blog. [Online]. Tersedia: <https://blog.litespeedtech.com/2013/04/26/concurrent-connections-demystified/> (diakses 11 Juni 2023).
- [28] T. McConaghy and M. Khan, “*Performance Study: Analysis of Transaction Throughput in a BigchainDB Network*,”, BigchainDB, [Online]. Tersedia: <https://github.com/bigchaindb/BEPs/tree/master/23> (diakses 12 Juni 2023).
- [29] E. Zvornicanin, “*Differences between bidirectional and Unidirectional LSTM*,”, Baeldung on Computer Science. [Online]. Tersedia: <https://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm> (diakses 20 Mei 2023).
- [30] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “*Smote: Synthetic minority over-sampling technique*,”, *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002. doi:10.1613/jair.953
- [31] Kaspersky, “*IP spoofing: How it works and how to prevent it*,”, www.kaspersky.com. [Online]. Tersedia:

<https://www.kaspersky.com/resource-center/threats/ip-spoofing> (diakses 10 Juni 2023).

- [32] Administrator, “*hping3_cheatsheet_v1.0*,” Hping3. [Online]. Tersedia: https://cyberwar.nl/d/cheatsheets/hping3_cheatsheet_v1.0-ENG.pdf (diakses 20 Mei 2023).
- [33] D. M, “*How to select the best set of features using SVM?*,” Analytics India Magazine. [Online]. Tersedia: <https://analyticsindiamag.com/feature-selection-using-svm-and-model-building/> (diakses 13 Juni 2023).



LAMPIRAN

Lampiran A. Hasil Pengecekan Turnitin

Skripsi Matthew Brandon Dani

ORIGINALITY REPORT

7% SIMILARITY INDEX 7% INTERNET SOURCES 3% PUBLICATIONS 2% STUDENT PAPERS

PRIMARY SOURCES

1	id.123dok.com Internet Source	<1 %
2	123dok.com Internet Source	<1 %
3	kc.umn.ac.id Internet Source	<1 %
4	Submitted to Napier University Student Paper	<1 %
5	Submitted to SDM Universitas Gadjah Mada Student Paper	<1 %
6	eprints.radenfatah.ac.id Internet Source	<1 %
7	repository.its.ac.id Internet Source	<1 %
8	www.semanticscholar.org Internet Source	<1 %
9	repository.unisba.ac.id:8080 Internet Source	<1 %

10	scholars.ttu.edu Internet Source	<1 %
11	www.coursehero.com Internet Source	<1 %
12	docplayer.info Internet Source	<1 %
13	www.zora.uzh.ch Internet Source	<1 %
14	sir.stikom.edu Internet Source	<1 %
15	dspace.uji.ac.id Internet Source	<1 %
16	www.scribd.com Internet Source	<1 %
17	etheses.uin-malang.ac.id Internet Source	<1 %
18	www.mdpi.com Internet Source	<1 %
19	Yan Li, Yifei Lu. "LSTM-BA: DDoS Detection Approach Combining LSTM and Bayes", 2019 Seventh International Conference on Advanced Cloud and Big Data (CBD), 2019 Publication	<1 %
20	repository.uin-suska.ac.id Internet Source	<1 %

21	adoc.pub Internet Source	<1 %
22	repository.ub.ac.id Internet Source	<1 %
23	Submitted to University of Maryland, University College Student Paper	<1 %
24	lib.unnes.ac.id Internet Source	<1 %
25	Luyt-Da Quach, Anh Nguyen Quynh, Khang Nguyen Quoc, An Nguyen Thi Thu. "Using the Term Frequency-Inverse Document Frequency for the Problem of Identifying Shrimp Diseases with State Description Text", International Journal of Advanced Computer Science and Applications, 2023 Publication	<1 %
26	Submitted to Universitas Pelita Harapan Student Paper	<1 %
27	roperzh.com Internet Source	<1 %
28	id.scribd.com Internet Source	<1 %
29	text-id.123dok.com Internet Source	<1 %

30	primadonakita.blogspot.com Internet Source	<1 %
31	Submitted to University of Limerick Student Paper	<1 %
32	eprints.utdi.ac.id Internet Source	<1 %
33	scholar.unand.ac.id Internet Source	<1 %
34	digilib.uinsby.ac.id Internet Source	<1 %
35	hudaanwar.wordpress.com Internet Source	<1 %
36	pt.scribd.com Internet Source	<1 %
37	repository.usd.ac.id Internet Source	<1 %
38	www.scilit.net Internet Source	<1 %
39	Submitted to Nottingham Trent University Student Paper	<1 %
40	digilib.unila.ac.id Internet Source	<1 %
41	jurnal.umpwr.ac.id Internet Source	<1 %

MULTIMEDIA
NUSANTARA

42	Submitted to AUT University Student Paper	<1 %
43	library.polmed.ac.id Internet Source	<1 %
44	teknologipintar.org Internet Source	<1 %
45	Submitted to Universitas Brawijaya Student Paper	<1 %
46	bm.technave.com Internet Source	<1 %
47	hdl.handle.net Internet Source	<1 %
48	mh4x0f.github.io Internet Source	<1 %
49	repository.unisma.ac.id Internet Source	<1 %
50	toyibofficial.wordpress.com Internet Source	<1 %
51	www.lawangtechno.com Internet Source	<1 %
52	jurnal.untan.ac.id Internet Source	<1 %
53	widuri.raharja.info Internet Source	<1 %
54	www.hindawi.com Internet Source	<1 %
55	www.researchgate.net Internet Source	<1 %
56	issuu.com Internet Source	<1 %
57	medium.com Internet Source	<1 %
58	repository.umsu.ac.id Internet Source	<1 %
59	Submitted to Politeknik Negeri Bandung Student Paper	<1 %
60	es.scribd.com Internet Source	<1 %
61	library.palcomtech.com Internet Source	<1 %
62	repository.dinamika.ac.id Internet Source	<1 %
63	repository.usahidsolo.ac.id Internet Source	<1 %
64	teknik.unpas.ac.id Internet Source	<1 %
65	www.pps.unud.ac.id Internet Source	<1 %

66	baixardoc.com Internet Source	<1 %
67	downloads.hindawi.com Internet Source	<1 %
68	e-repository.perpus.iainsalatiga.ac.id Internet Source	<1 %
69	garuda.kemdikbud.go.id Internet Source	<1 %
70	lib.buet.ac.bd:8080 Internet Source	<1 %
71	mti.binus.ac.id Internet Source	<1 %
72	pdfcookie.com Internet Source	<1 %
73	qdoc.tips Internet Source	<1 %
74	www.deletec.fr Internet Source	<1 %
75	www.tuts.web.id Internet Source	<1 %
76	ejournal.unjaya.ac.id Internet Source	<1 %
77	eudi.eu Internet Source	<1 %
78	jurnal.itats.ac.id Internet Source	<1 %
79	jurnal.polibatam.ac.id Internet Source	<1 %
80	lib.ui.ac.id Internet Source	<1 %
81	repository.president.ac.id Internet Source	<1 %
82	riaratnasari183.wordpress.com Internet Source	<1 %
83	www.ndrangsan.com Internet Source	<1 %
84	www.unl.pt Internet Source	<1 %
85	wwwdienz.blogspot.com Internet Source	<1 %
86	Dwi Intan Afidah, Dairoh Dairoh, Sharfina Febbi Handayani, Riszki Wijayatun Pratiwi, Susi Indah Sari, "Sentimen Ulasan Destinasi Wisata Pulau Bali Menggunakan Bidirectional Long Short Term Memory", MATRIK : Jurnal Manajemen, Teknik Informatika dan Rekayasa Komputer, 2022 Publication	<1 %
87	saripedia.wordpress.com	

UNIVERSITAS
MULTIMEDIA
NUSANTARA



UMN

UNIVERSITAS MULTIMEDIA NUSANTARA

159

Rancang Bangun Sistem Proxy Mitigasi DDoS berbasis Machine Learning dan Blockchain,
Matthew Brandon Dani, Universitas Multimedia Nusantara

FORMULIR KONSULTASI SKRIPSI – FAKULTAS TEKNIK & INFORMATIKA

Dosen Pembimbing : Samuel Hutagalung M.T.I.
 Jurusan : Teknik Komputer
 Semester : 8
 Nama : Matthew Brandon Dani
 NIM : 00000036391



Tanggal Konsultasi	Agenda/Pokok Bahasan	Saran Perbaikan	Paraf Dosen Pembimbing
3 Februari 2023	Konsultasi Topik dan Judul Penelitian	1. Latar belakang memakai blockchain	
5 Februari 2023	Konsultasi Topik dan Judul Penelitian	1. Mencari referensi penelitian terdahulu	
26 Mei 2023	Demo Reverse Proxy dan Pengambilan Dataset	1. Metode machine learning timeseries 2. Latar belakang machine learning	
2 Juni 2023	Demo Machine Learning dan Perbandingan Sistem IPS / IDS lain	1. Machine learning untuk identifikasi Signature 2. Cari Referensi cara kerja IPS atau IDS lain	
9 Juni 2023	Konsultasi Pengujian Sistem	1. Dapat dikurangin untuk opsi machine learning sebagai hyperparameter penelitian 2. Revisi bab 1	
14 Juni 2023	Konsultasi Pengujian Matriks	1. Perbandingan machine learning antar invidual dan timeseries tidak perlu menggunakan TTM	
15 Juni 2023	Konsultasi Laporan Bab 1, 2, dan 3	1. Tambahkan hipotesis H0 dan H1 2. Narasi state-of-the-art	
16 Juni 2023	Konsultasi Laporan Bab 4 dan 5	1. Revisi Final	

Catatan : Form ini wajib dibawa pada saat konsultasi & dilampirkan didalam skripsi (Minimal 8 kali Konsultasi)

Tangerang, 16 Juni 2023

Samuel Hutagalung M.T.I.

Dosen Pembimbing

FORMULIR KONSULTASI SKRIPSI – FAKULTAS TEKNIK & INFORMATIKA

Dosen Pembimbing : Daren Kusuma Halim S.Kom., M.Eng.Sc

Jurusan : Teknik Komputer

Semester : 8

Nama : Matthew Brandon Dani

NIM : 00000036391

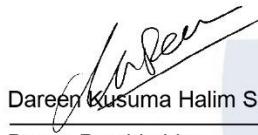


Tanggal Konsultasi	Agenda/Pokok Bahasan	Saran Perbaikan	Paraf Dosen Pembimbing
3 Februari 2023	Konsultasi Topik dan Judul Penelitian	1. Latar belakang memakai blockchain	
5 Februari 2023	Konsultasi Topik dan Judul Penelitian	1. Mencari referensi penelitian terdahulu	
26 Mei 2023	Demo Reverse Proxy dan Pengambilan Dataset	1. ML pendekatan secara timeseries 2. Latar belakang memakai ML 3. Perbandingan IPS Open-source	
2 Juni 2023	Demo Machine Learning dan Perbandingan Sistem IPS / IDS lain	1. Melakukan ML untuk DDoS signature 2. Research tentang IPS / IDS Signature 3. Research tentang cara kerja IDS / IPS selain Snort	
9 Juni 2023	Konsultasi Pengujian Sistem	1. Dapat dikurangi untuk opsi machine learning sebagai hyperparameter penelitian 2. Revisi bab 1	
14 Juni 2023	Konsultasi Pengujian Matriks	1. Perbandingan machine learning antar individual dan timeseries tidak perlu menggunakan TTM	
15 Juni 2023	Konsultasi Laporan Bab 1, 2, dan 3	1. Revisi bab 1 latar belakang 2. Revisi diagram state-of-the-art 3. Revisi bab 4 4. Revisi format laporan	

16 Juni 2023	Konsultasi Laporan Bab 4 dan 5	1. Revisi Final	

Catatan : Form ini wajib dibawa pada saat konsultasi & dilampirkan didalam skripsi (**Minimal 8 kali Konsultasi**)

Tangerang, 16 Juni 2023


Dareen Kusuma Halim S.kom.,M.Eng.Sc
Dosen Pembimbing

