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. *Bandwith 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 direspons 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

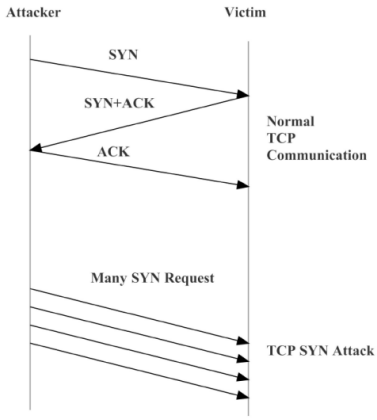
Tipe ICMP yang dipakai adalah ICMP_ECHO_REPLY, jika paket tersebut dikirimkan maka perangkat tujuan akan mengirimkan kembali paket tersebut. Namun penyerang akan mengirimkan paket ini terus menerus dengan selang waktu yang sangat kecil dan jumlah yang banyak.

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 :

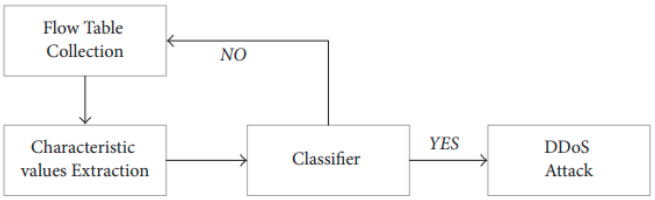
1. Penulis mengetahui tentang jenis serangan DDoS yang umum dilakukan dan cara kerja serangannya. Yaitu UDP flood, HTTP flood, ICMP flood, TCP Syn flood, TCP PUSH + ACK flood, Ping of Death Flood.
2. Dapat diimplementasi teknik honeypots, dimana melakukan *logging* setiap paket yang berlalu untuk nantinya 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.

4. 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. Pendeteksian 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 pendeteksian serangan DDoS dengan pendekatan individual yaitu menemukan suatu anomali pada *stream packet* dengan rentang waktu tertentu. Dilakukan beberapa proses pendeteksian 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.

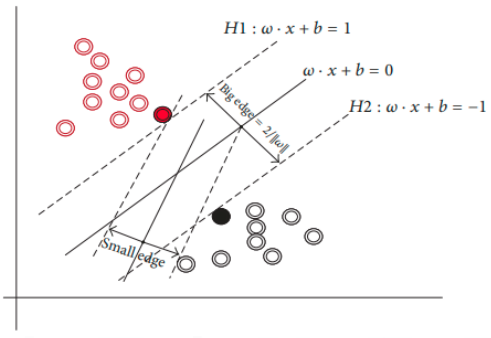
Tabel 2.1 Features yang Digunakan dalam Deteksi Serangan DDoS menggunakan SVM [11]

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

5	SFE (*Speed of Flow Entries*)	$SFE = \frac{N}{T}$	Jumlah keseluruhan paket dalam rentang satuan waktu
6	RPF (*Ratio of Pair Flow*)	$RPF = \frac{2 * Pair\_Sum}{N}$	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 pengklasifikasiannya 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

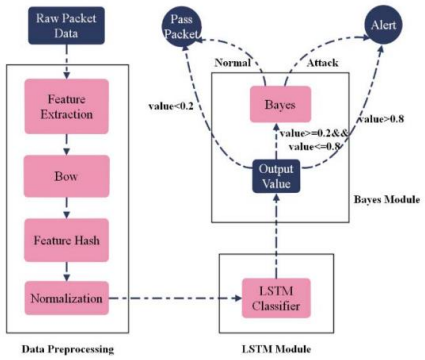
machine learning yaitu LSTM dan Bayes. Digunakan 2 algoritma *machine learning* karena jika saat fase deteksi pertama yang menggunakan LSTM memiliki keluaran level *confident* yang kecil maka akan dilakukan deteksi kedua menggunakan Bayes. Penelitian ini menggunakan publik dataset yaitu ISCX2012. Hasil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



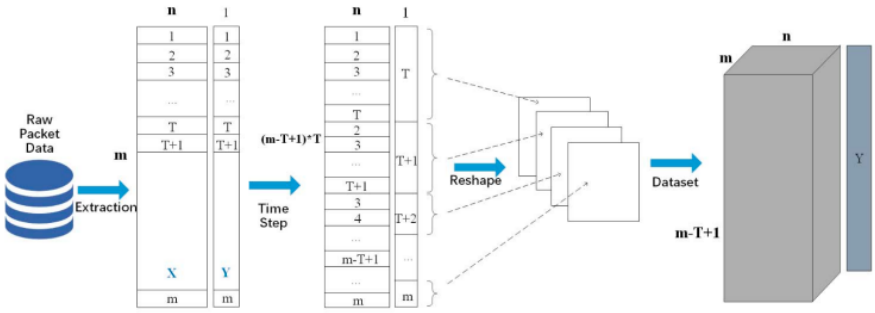
Gambar 2.4 Proses Deteksi menggunakan LSTM [12]

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]

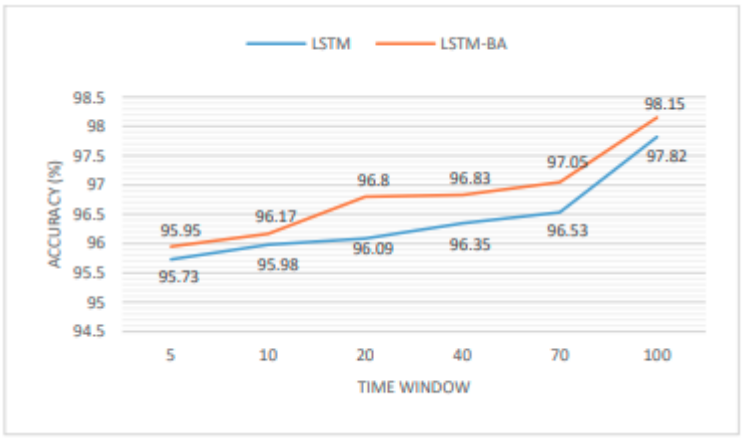
Jika dilihat pada gambar 2.5 diatas, contoh untuk packet 1,2,3 dijadikan 1, lalu label yang dipakai adalah label paket 3. Selanjutnya diulangi untuk paket 2, 3, dan 4 dan menggunakan label paket 4. Pada penelitian ini dipakai 6 buah konfigurasi *time window* yang berbeda yaitu 5, 10, 20, 40, 70, dan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

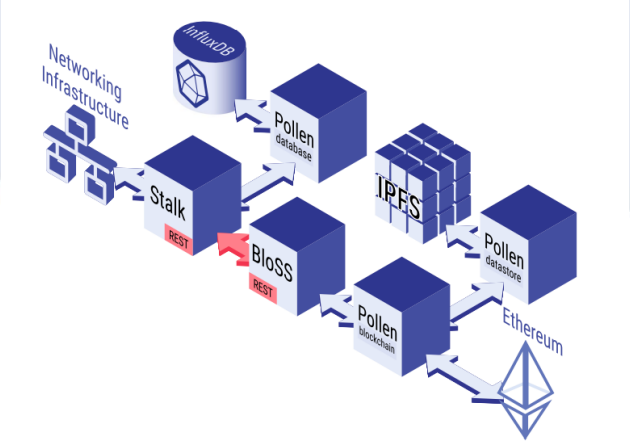
2. Digunakan 10 *features* dari dataset yaitu *Source IP Address*, *destination IP address*, *source port*, *destination port*, *protocol type*, *timestamp*, *duration*, *type of service*, *length*, dan *time to live*.
3. Model LSTM memerlukan *input* berupa 3 *dimensional matrix* yaitu *batch size*, *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.



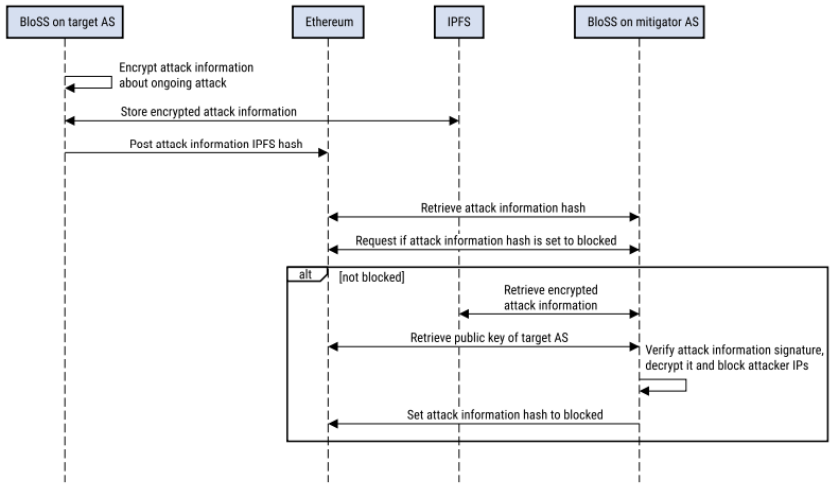
Gambar 2.7 Arsitektur Pembagian Informasi Penyerang menggunakan Blockchain dan IPFS [13]

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 mengonfigurasi *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

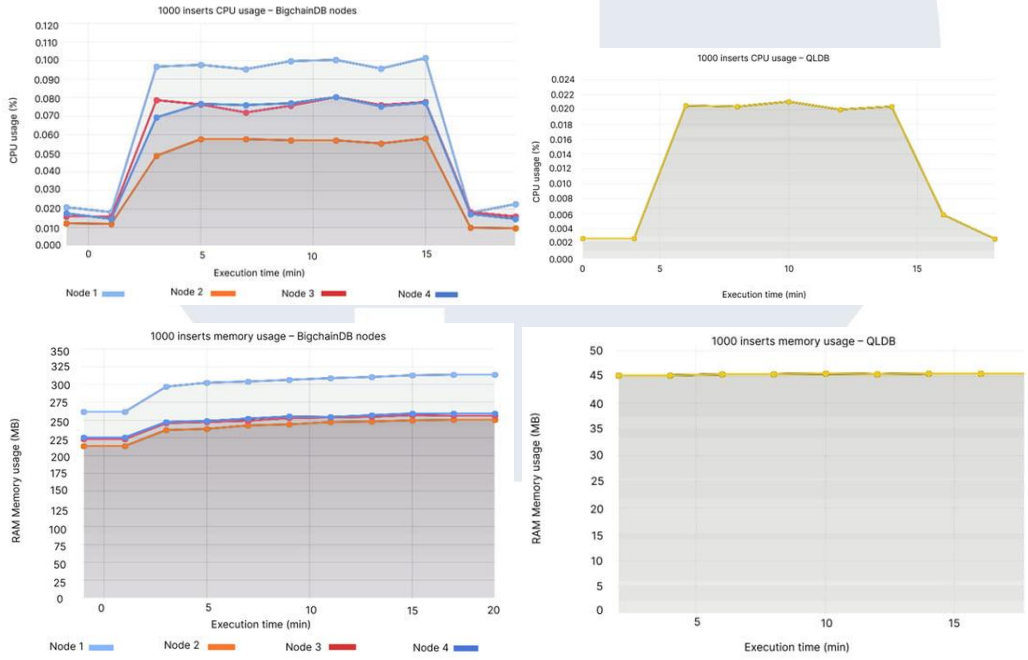
2.1.5 Centralized vs Decentralized: Performance Comparison between BigchainDB and Amazon QLDB [14]

Penelitian dengan judul "*Centralized vs Decentralized: Performance Comparion between BigchainDB and Amazon QLDB*" yang dilakukan oleh Sergiu Lupaiescu, Petru Cioata, Cristina Elena Turcu, Ovidiu Gherman, Corneliu 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 throughtput, 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.

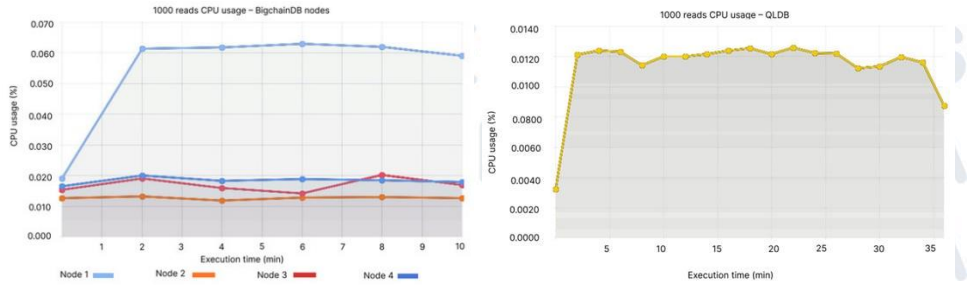
Disisi lain, Amazon QLDB menggunakan fitur *cryptography* agar datanya memiliki kemampuan *immutability*. namun sistemnya masih menggunakan arsitektur *centralize*. Dan Amazon QLDB tidak *open-source*.

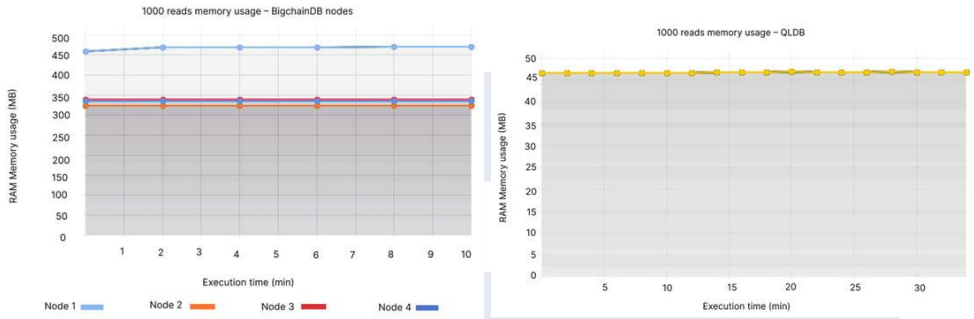
Pada penelitian ini ditemukan bahwa dengan BigchainDB dengan melakukan *input* 1000 data ke dalam *database*, ditampilkan penggunaan CPU dan *memory* pada gambar 2.9.



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 *throughtput* 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 konsesus yang cepat dan hemat dalam penggunaan sumber daya komputasi jika dibandingkan dengan konsensus POS atau POW Etherium.

5. BigchainDB walaupun implementasinya cukup rumit namun miliki 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]

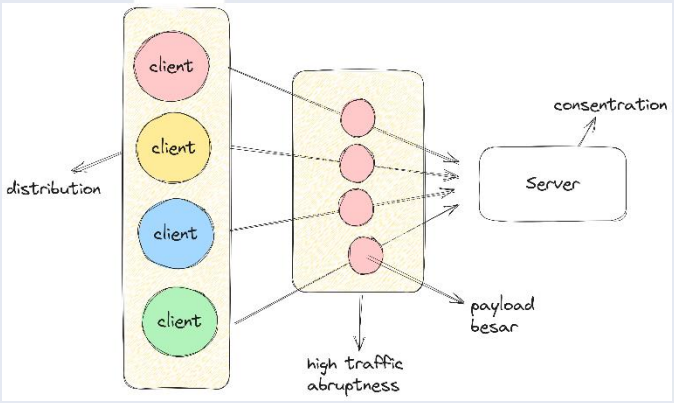
Penelitian dengan judul "*A DDoS Attack Information Fusion Method Based on CNN for Multi-Element Data*" yang dilakukan oleh Jieren Cheng, Canting Cai, Xiangyan Tang, Victor S. Sheng, Wei Gui, dan Mengyang Li memiliki tujuan untuk melakukan efisiensi proses komputasi, penggunaan *memory*, *running time*, 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 karakateristik 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.
   Rumus : $SIPAF = |\cup_{i=1}^{n}\{source\ ip\ i\}|$
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 sabil 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.

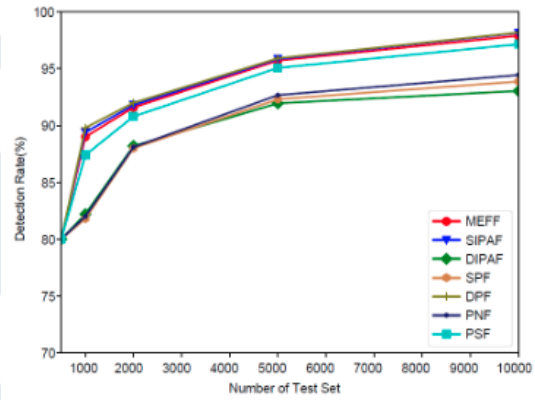
   Rumus : $PNF = n$

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**		**16 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_{sip} (μs)	54.41	53.55	83.62	85.34	101.28	116.15	125.78	139.26	218.07	262.32
K_{rcv} (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*.

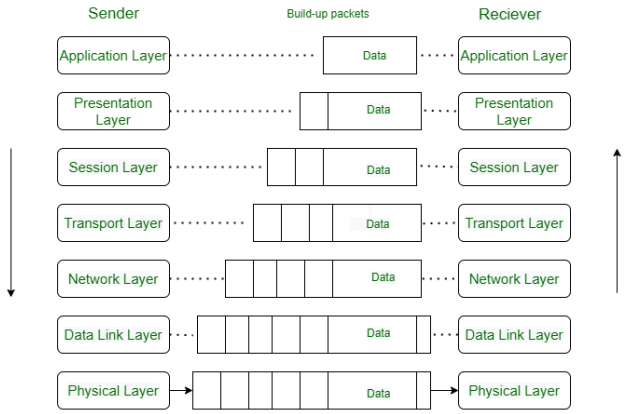
6. Menggunakan *control plane* SDN sebagai *listener* dan *logger* paket yang berlalu.
7. Diteliti pendekatan metode *machine learning* secara *timeseries* menggunakan algoritma *deep learning* LSTM. Digunakan LSTM karena bagus dalam mendeteksi secara *realtime* dan memiliki akurasi yang bagus karena 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 disebarkan 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 *throughtput* 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

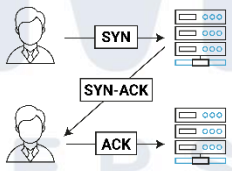
Ketujuh lapisan OSI layer beserta fungsinya adalah :

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 pendeteksi 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 mengakhir komunikasi 2 arah. Ilustrasi dari TCP *handshaking* dapat dilihat pada gambar 2.15.

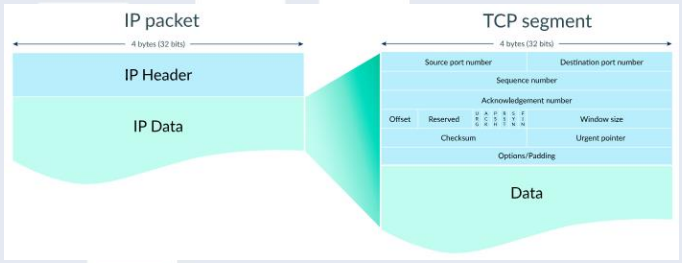


Gambar 2.15 Alur handshaking TCP [19]

Jadi pertama kali *client* akan mengirimkan TCP SYN lalu server akan menyimpan *IP address client* ke tabel antrean. 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]

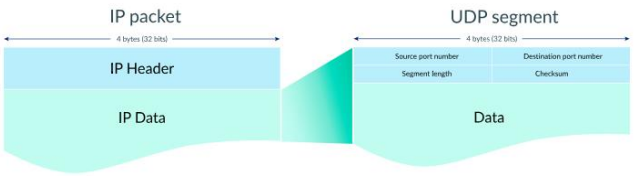
Untuk melakukan *handshake*, diperlukan beberapa data yang disimpan pada *IP Header*.

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. *Acknowledgent 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]

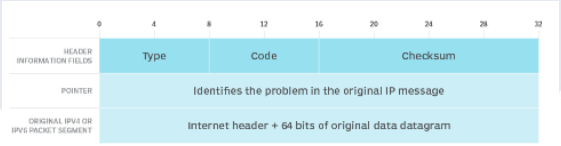
Penjabaran tentang isi dari *header* dari komunikasi UDP adalah sebagai berikut :

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 (*Intenet 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*.

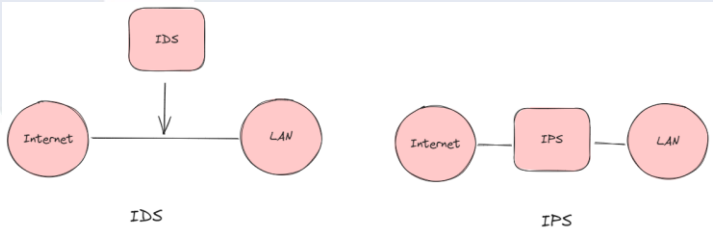
2.2.3 *Intrusion Prevention System* dan *Intrusion Detection System* [24]

IPS (*Intrusion Prevention System*) dan IDS (*Intrusion Detection System*) adalah *tools* yang penting dalam keamanan jaringan komputer khususnya serangan DDoS yang berfungsi untuk mendeteksi 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 kompuer 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 Log Pass Drop Reject	TCP UDP ICMP			< > <>			Msg Ttl Seq Ack 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.

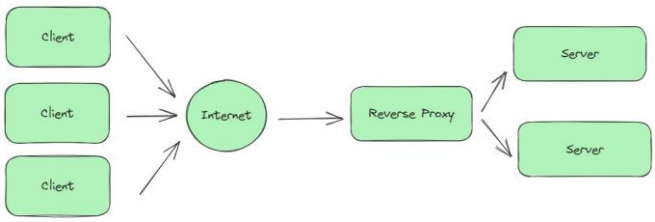
2.2.4 NetfilterQueue

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.

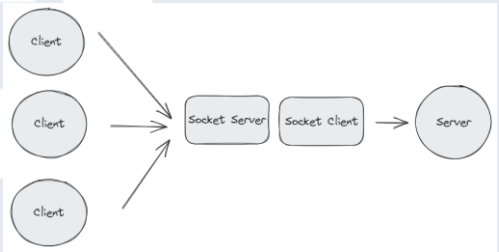
2.2.5.1 Concurent Connection [27]

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.

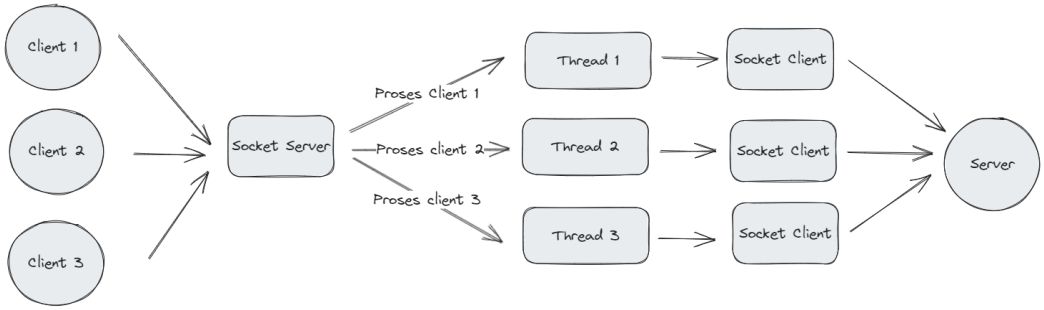


Gambar 2.21 Contoh Implementasi Socket untuk Reverse Proxy

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 kembalikan. 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

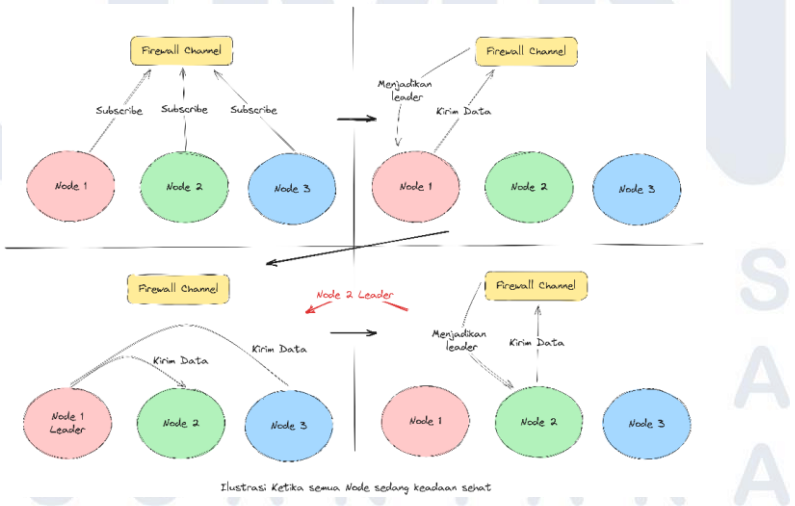
Database : CRUD	BigchainDB : CRAB
Create	**Create**
Read	**Retrieve**

Update	**Append** (membuat transaksi / data baru untuk memperbaharui suatu data)
Delete	**Burn** (membuat transaksi / data baru untuk memperbaharui 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 memperbaharui *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 *consentration* 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:

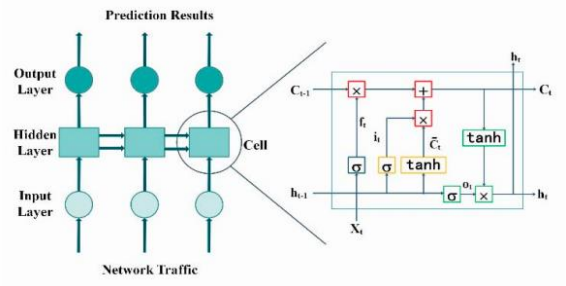
- K-Nearest Neighbors (K-NN): Algoritma ini mengklasifikasikan data dengan melihat tetangga terdekatnya dari suatu titik dengan mengalkulasikan *Euclidean distance*. K-NN bagus digunakan pada set *features* yang kecil dengan dataset yang sedikit, namun cocok juga pada set *features* yang besar dengan 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]

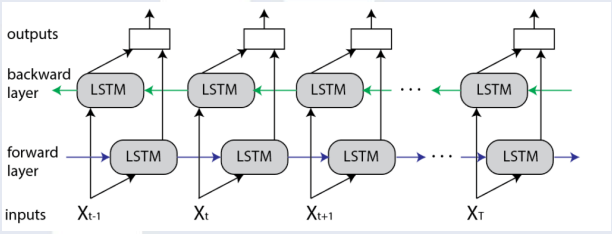
Bidirectional LSTM atau *Bidirectional* Long Short-Term *Memory* adalah salah satu algoritma dari *deep learning*. *Deep learning* sendiri merupakan cabang dari *machine learning* yang menggunakan saraf tiruan sebagai algoritma pembelajarannya. LSTM menggunakan jaringan saraf berulang atau yang disebut 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

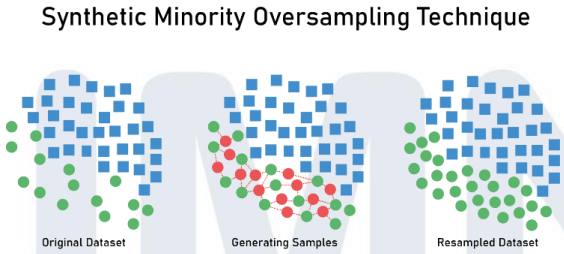
Ketika mengumpulkan dataset jaringan komputer pasti akan ditemukan ketidakseimbangan antara dataset normal dengan dataset serangan DDoS. Hal ini disebabkan karena beberapa faktor, yang pertama karena ketika serangan DDoS akan menghasilkan data yang lebih banyak karena data yang terkirim jauh lebih banyak 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, beberapanya 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 yang 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]

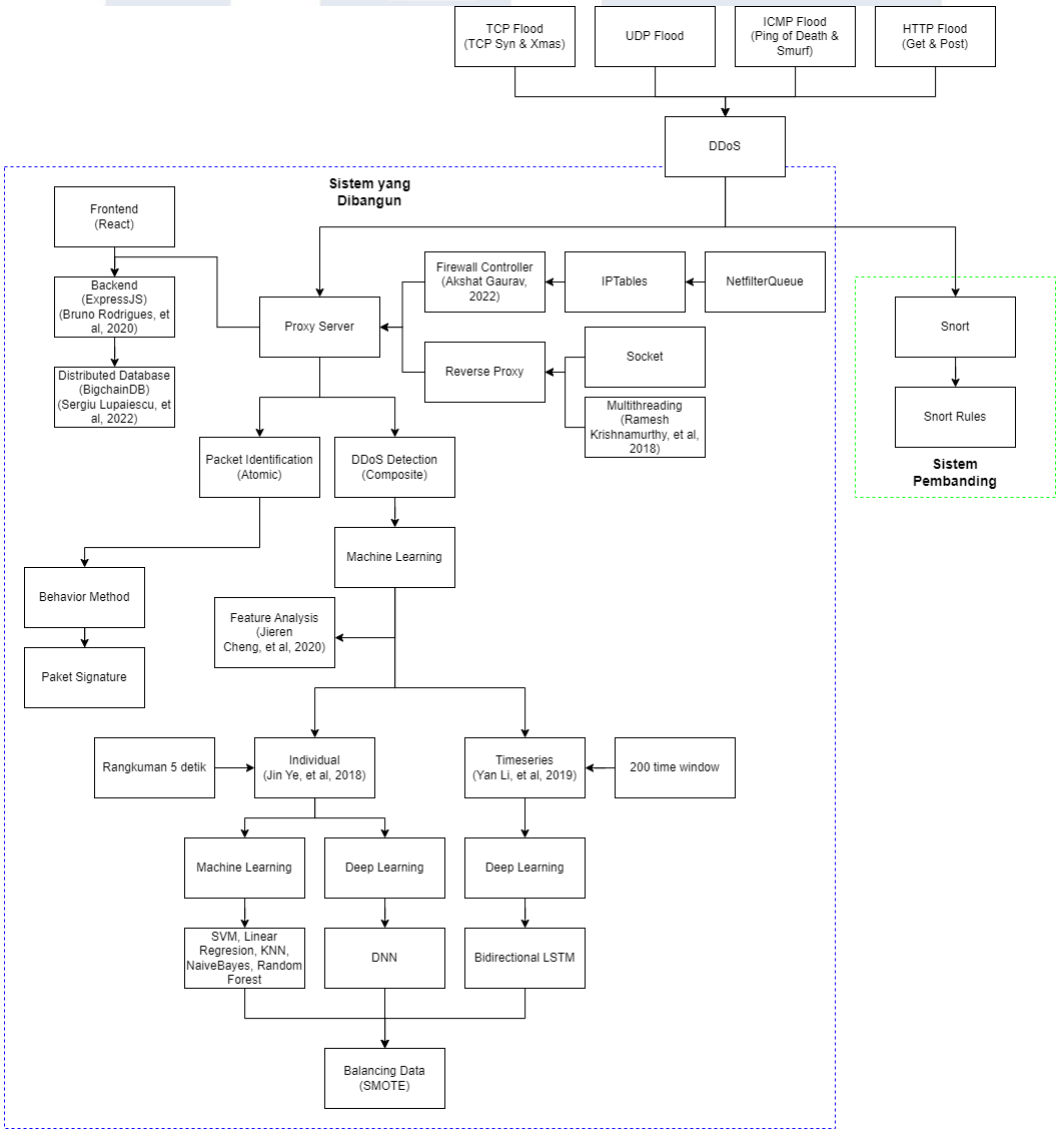
IP spoofing adalah praktik mengirim paket data dengan memalsukan alamat IP pengirimnya. Saat memproses data, komputer atau jaringan sering menggunakan alamat IP pengirim untuk mengidentifikasi asal paket dan memutuskan apakah harus menerima atau menolaknya. Dalam serangan IP spoofing, penyerang mencoba 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 Diagaram State of The Art