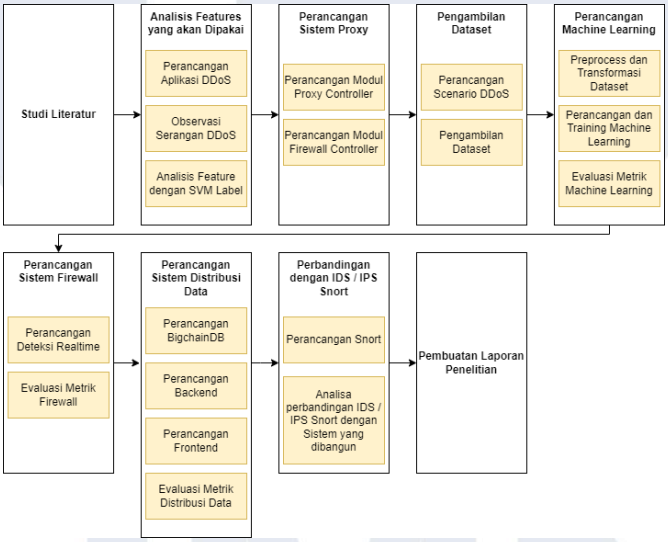
BAB III

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

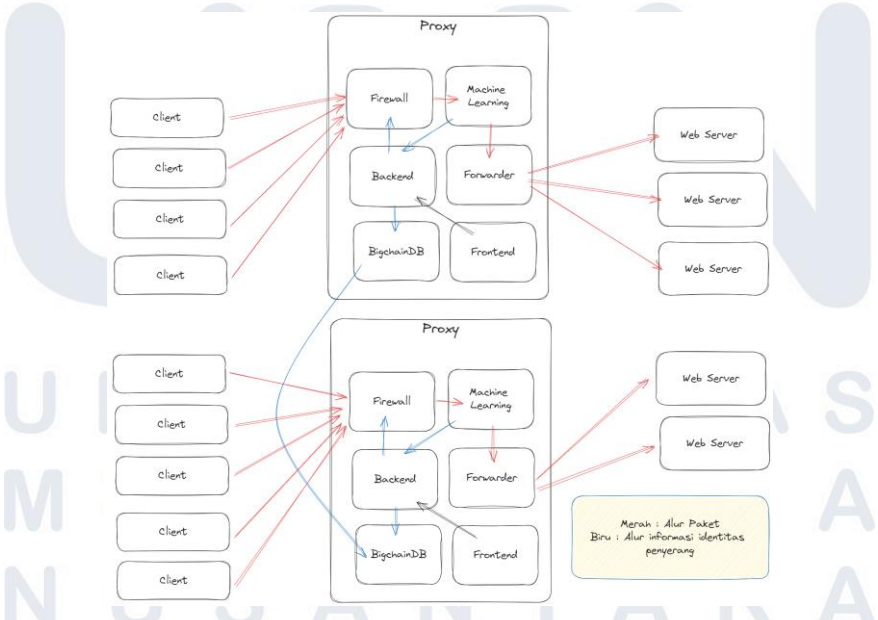
Studi literatur yang dilakukan penulis adalah dengan mencari informasi berupa penelitian terdahulu dan dokumentasi teknologi yang berkaitan dan mendukung penelitian ini. Dalam hal ini adalah penelitian yang terkait dengan proxy, *machine learning* untuk deteksi dan identifikasi serangan DDoS, dan *blockchain*. 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 pendeteksi 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.



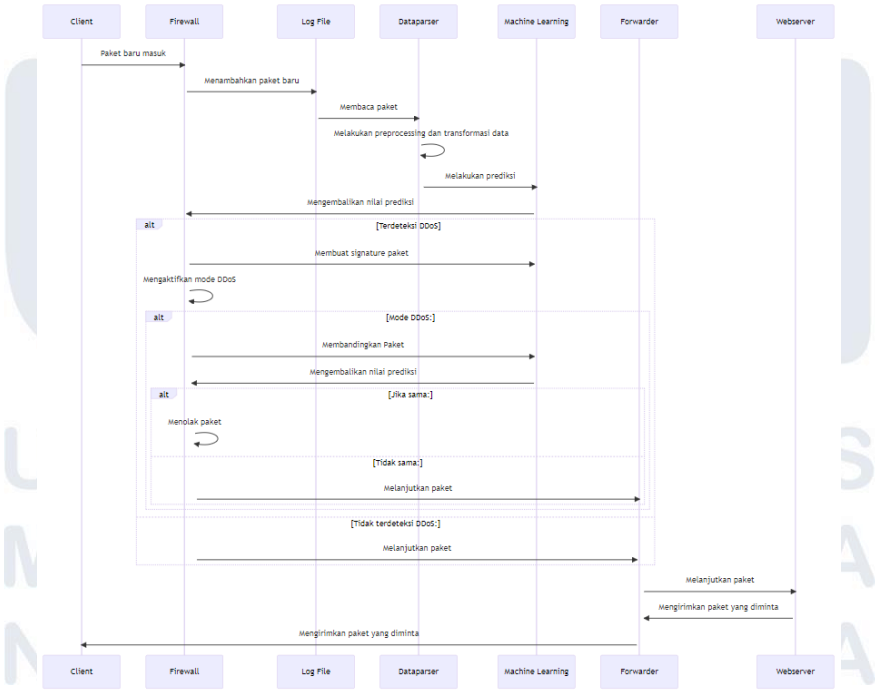
Gambar 3.2 Diagram Arsitektur Sistem secara General

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*.

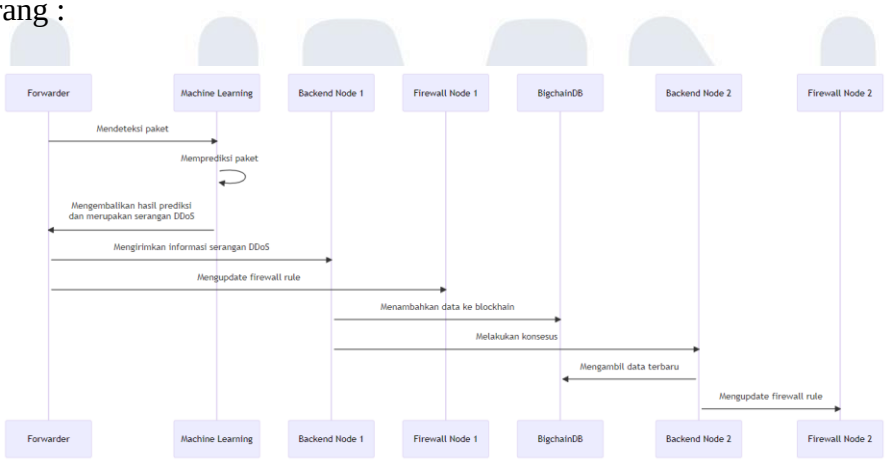


Gambar 3.3 Sequence Diagram untuk Deteksi Paket

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 :

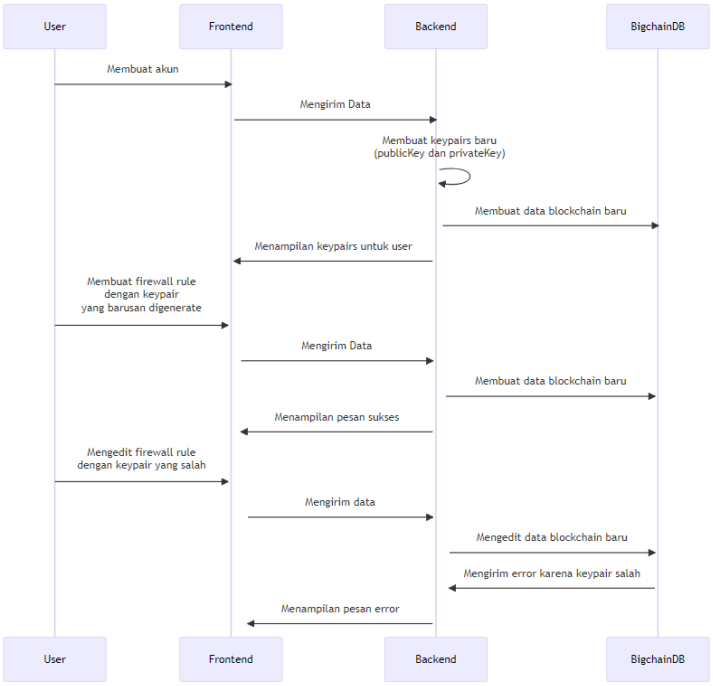


Gambar 3.4 Sequence Diagram untuk Persebaran Data Informasi Penyerang

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

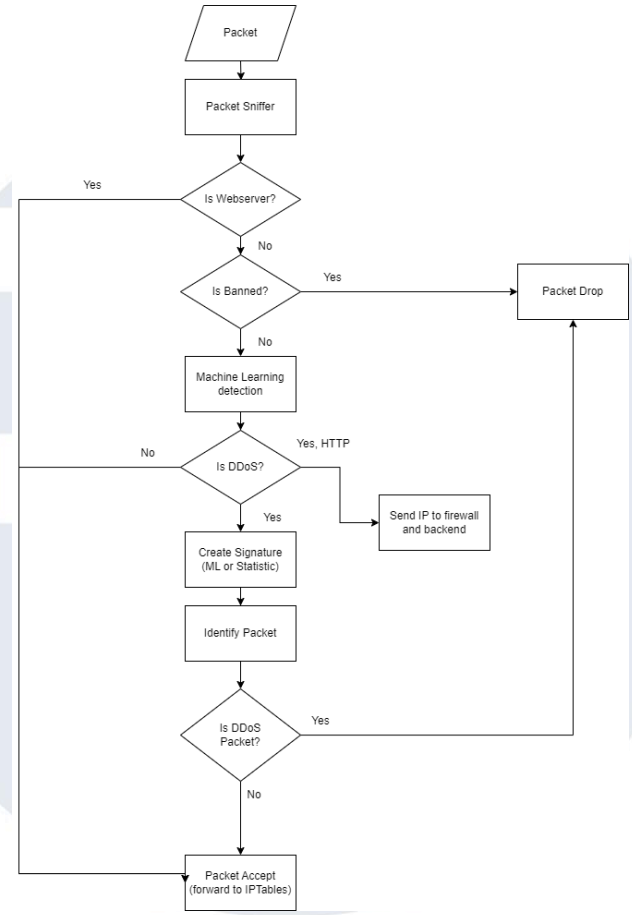
plane antara forwarder, BigchainDB, dan Firewall. Alur ini sesuai dengan solusi pada sumber referensi penelitian terdahulu nomor 2.1.4.

3. Admin Membuat Akun dan mengontrol Firewall Rule



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

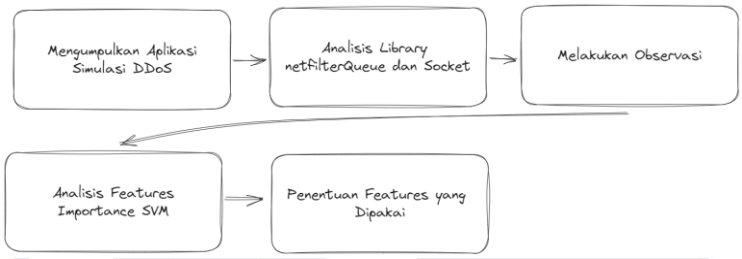
Penjelasan untuk *sequence diagram* pada gambar 3.5 adalah saat admin atau user yang mengoperasikan sistem ingin menambahkan *firewall rule* atau data pada sistem yang dibangun. Pertama kali user akan membuat akun terlebih dahulu. Setiap user yang baru mendaftar, akan mendapatkan *keypair* baru berupa *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 *blokchain*, 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 kodenya 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.

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 kodenya sebagai ditampilkan pada gambar 3.11.

```
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 memanggi 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

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.

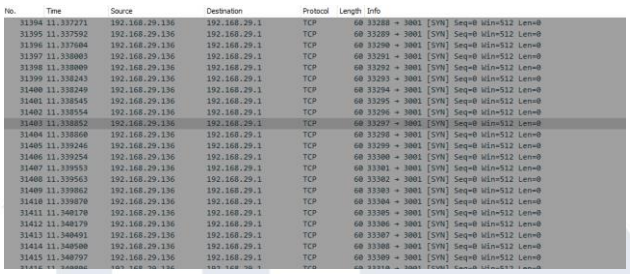
A. TCP: IP Source, IP Destination, Port Source, Port Destination, Sequence, Acknowledge, Dataofs, Reserved, Flags, Window, Chksum, Urgpt, Payload length
B. UDP : IP Source, IP Destination, Port Source, Port Destination, Len (hanya besar dari data saja tidak termasuk header), Chksum, Payload length
C. 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
31394	11.337271	192.168.29.136	192.168.29.1	TCP	60	33288 → 3001 [SYN] Seq=0 Win=512 Len=0
31395	11.337592	192.168.29.136	192.168.29.1	TCP	60	33289 → 3001 [SYN] Seq=0 Win=512 Len=0
31396	11.337604	192.168.29.136	192.168.29.1	TCP	60	33290 → 3001 [SYN] Seq=0 Win=512 Len=0
31397	11.338003	192.168.29.136	192.168.29.1	TCP	60	33291 → 3001 [SYN] Seq=0 Win=512 Len=0
31398	11.338009	192.168.29.136	192.168.29.1	TCP	60	33292 → 3001 [SYN] Seq=0 Win=512 Len=0
31399	11.338243	192.168.29.136	192.168.29.1	TCP	60	33293 → 3001 [SYN] Seq=0 Win=512 Len=0
31400	11.338249	192.168.29.136	192.168.29.1	TCP	60	33294 → 3001 [SYN] Seq=0 Win=512 Len=0
31401	11.338545	192.168.29.136	192.168.29.1	TCP	60	33295 → 3001 [SYN] Seq=0 Win=512 Len=0
31402	11.338554	192.168.29.136	192.168.29.1	TCP	60	33296 → 3001 [SYN] Seq=0 Win=512 Len=0
31403	11.338852	192.168.29.136	192.168.29.1	TCP	60	33297 → 3001 [SYN] Seq=0 Win=512 Len=0
31404	11.338860	192.168.29.136	192.168.29.1	TCP	60	33298 → 3001 [SYN] Seq=0 Win=512 Len=0
31405	11.339246	192.168.29.136	192.168.29.1	TCP	60	33299 → 3001 [SYN] Seq=0 Win=512 Len=0
31406	11.339254	192.168.29.136	192.168.29.1	TCP	60	33300 → 3001 [SYN] Seq=0 Win=512 Len=0
31407	11.339553	192.168.29.136	192.168.29.1	TCP	60	33301 → 3001 [SYN] Seq=0 Win=512 Len=0
31408	11.339563	192.168.29.136	192.168.29.1	TCP	60	33302 → 3001 [SYN] Seq=0 Win=512 Len=0
31409	11.339862	192.168.29.136	192.168.29.1	TCP	60	33303 → 3001 [SYN] Seq=0 Win=512 Len=0
31410	11.339870	192.168.29.136	192.168.29.1	TCP	60	33304 → 3001 [SYN] Seq=0 Win=512 Len=0
31411	11.340170	192.168.29.136	192.168.29.1	TCP	60	33305 → 3001 [SYN] Seq=0 Win=512 Len=0
31412	11.340179	192.168.29.136	192.168.29.1	TCP	60	33306 → 3001 [SYN] Seq=0 Win=512 Len=0
31413	11.340491	192.168.29.136	192.168.29.1	TCP	60	33307 → 3001 [SYN] Seq=0 Win=512 Len=0
31414	11.340500	192.168.29.136	192.168.29.1	TCP	60	33308 → 3001 [SYN] Seq=0 Win=512 Len=0
31415	11.340797	192.168.29.136	192.168.29.1	TCP	60	33309 → 3001 [SYN] Seq=0 Win=512 Len=0

Gambar 3.15 Contoh Observasi Menggunakan Software Wireshark

4. Analisis *features importance* SVM

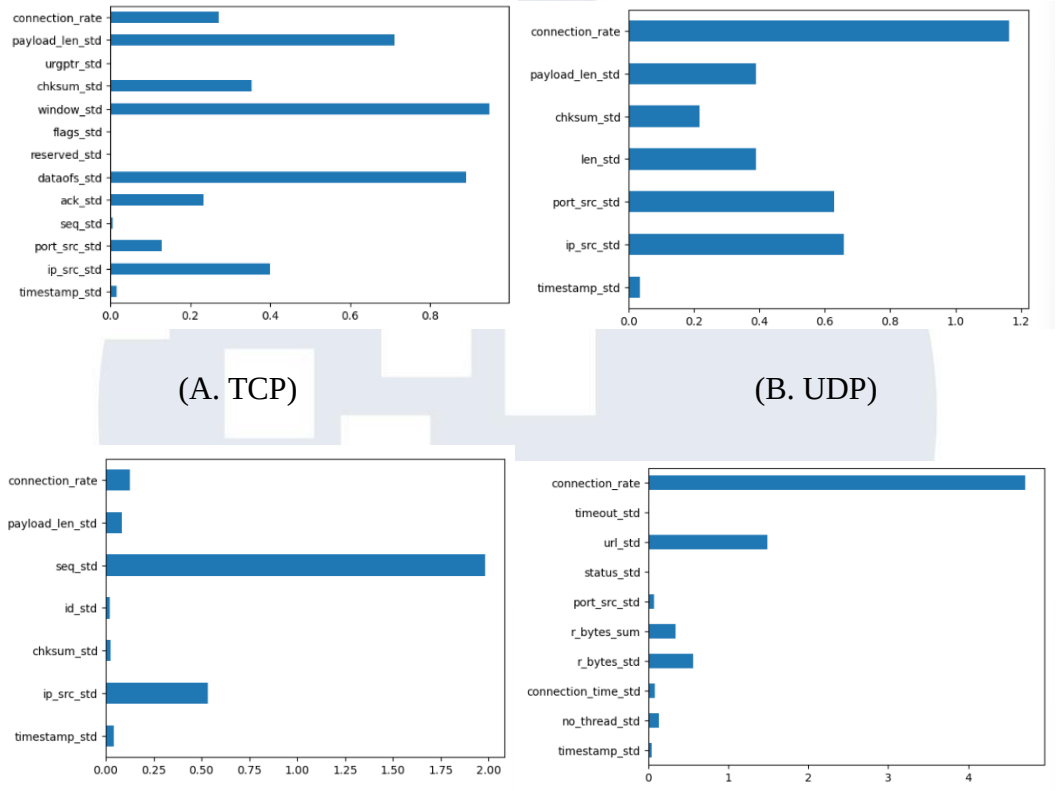
Penulis juga melakukan analisa pada model *machine learning* SVM yaitu *feature importance*. Dimana akan dilakukan *training* data pada algoritma SVM dan akan dilihat data pada model *machine learning* latihnya. Hal ini memiliki tujuan untuk melihat *features* apa yang paling berpengaruh pada saat algoritma 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, chksum_std, urgptr_std, payload_len_std, connection_rate
b. UDP : timestamp_std, ip_src_std, port_src_std, len_std, chksum_std, payload_len_std, connection_rate
c. ICMP : timestamp_std, ip_src_std, chksum_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)

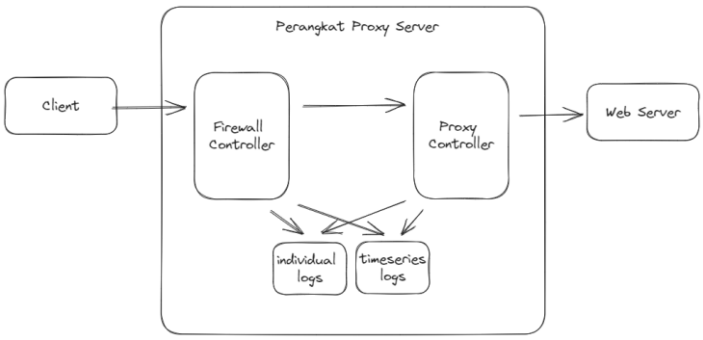
C. Comparator (untuk signature paket)
   a. TCP : dataofs, reserved, flags, window, urgptr, dan payload length
   b. UDP : len dan payload length
   c. ICMP : id dan payload length
   d. HTTP : IP source

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 :

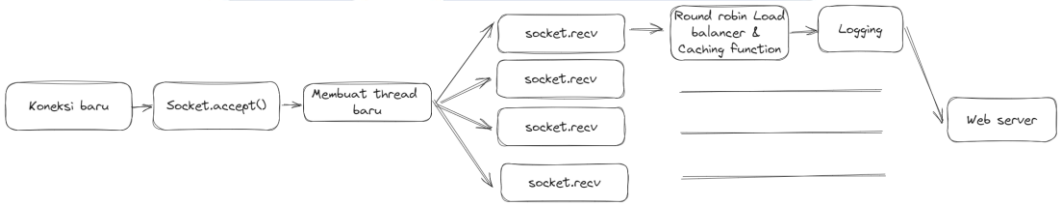
1. *Firewall controller* : Sebagai gerbang awal paket yang diterima, akan dilakukan logging terhadap paket layer 4 (*transport layer*) dengan protokol TCP, UDP, dan ICMP pada modul ini.
2. *Proxy controller* (sebelumnya disebut sebagai *forwarder*) : Sebagai inti dari program *reverse proxy*, dimana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

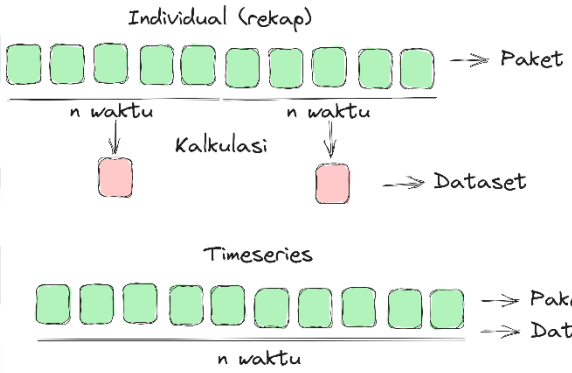
Karena fungsi *logging* dilakukan secara *multithreading* dan menggunakan 1 file log bersama maka agar tidak menyebabkan situasi *race condition* dilakukanlah metode *thread safe logging* dengan menggunakan *library* dari python.

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

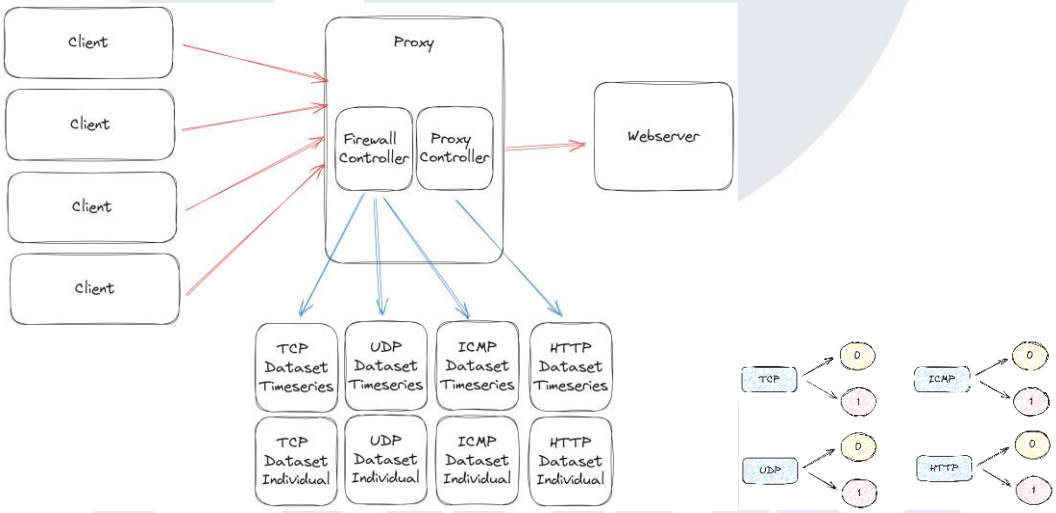
Karena deteksi *machine learning* akan dilakukan pada sistem yang dibangun sendiri, oleh karena itu untuk menghindari faktor-faktor pembeda situasi seperti perbedaan arsitektur data *logger*, infrastruktur dan lain-lain, maka dihindari menggunakan dataset publik. Sehingga dilakukan pengambilan dataset 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:

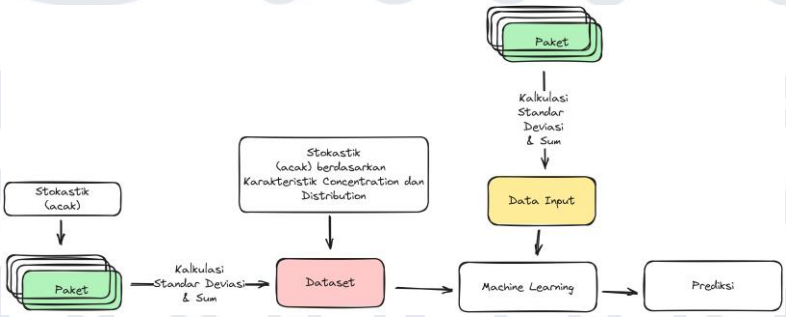
1. TCP
    a. TCP Flood SYN Single IP : Selama 4 menit
    b. TCP Flood SYN Random IP : Selama 4 menit
    c. TCP Flood XMAS Single IP : Selama 4 menit
    d. TCP Flood XMAS Random IP : Selama 4 menit
    e. Normal : Semua skenario untuk protokol HTTP. Karena serangan HTTP akan dideteksi menggunakan 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

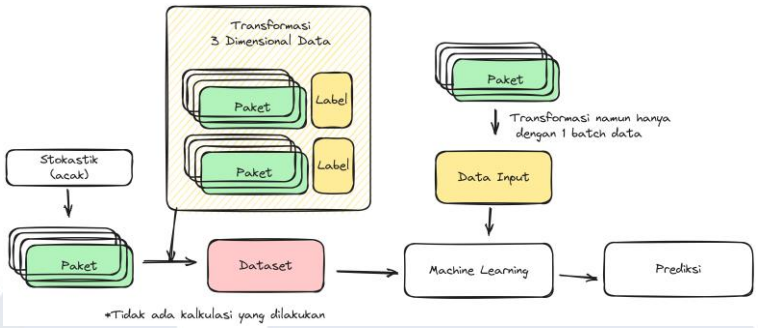
Semua skenario akan dijalankan dengan 4 *client* dan 1 *webserver*. Dataset akan berformat CSV yang nantinya akan digunakan untuk proses *training machine learning*. Dan untuk label penulis menentukan bahwa "0" adalah untuk label normal dan "1" adalah untuk label serang. Untuk arsitektur sistem yang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

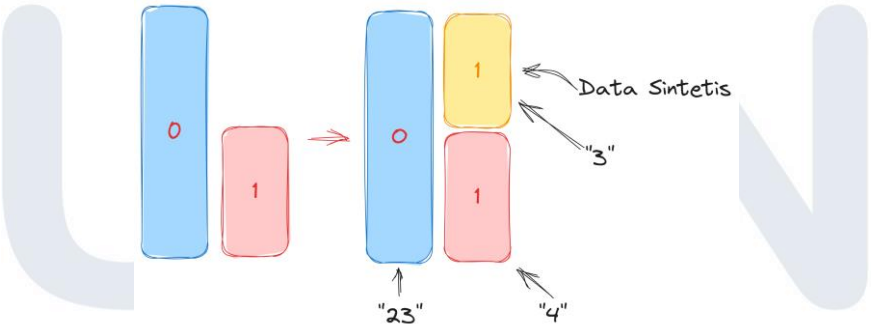
A. Balancing dataset

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 scalling*. 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 :

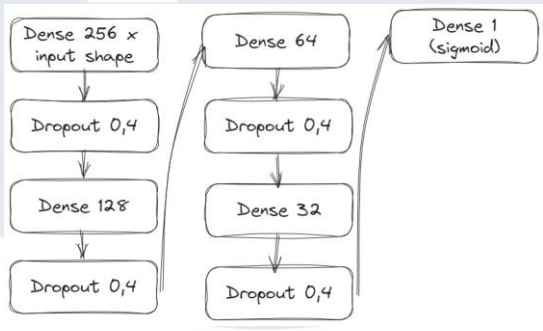
- *Machine learning*

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 learing* yang digunakan adalah DNN yaitu dengan arsitektur yang diilustrasikan pada gambar 3.28.



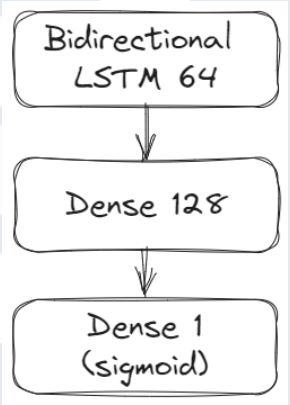
Gambar 3.28 Arsitektur Machine Learning DNN

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.

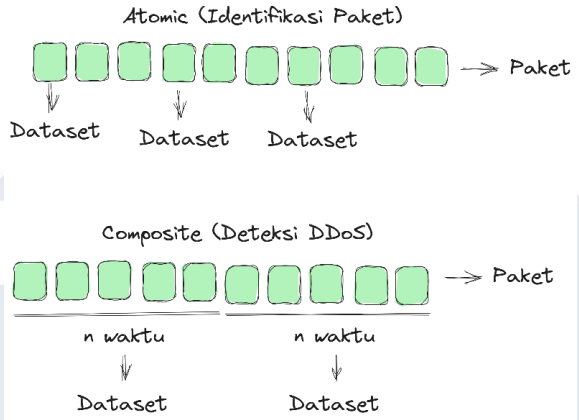


Gambar 3.29 Arsitektur Machine Learning Bidirectional LSTM

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.

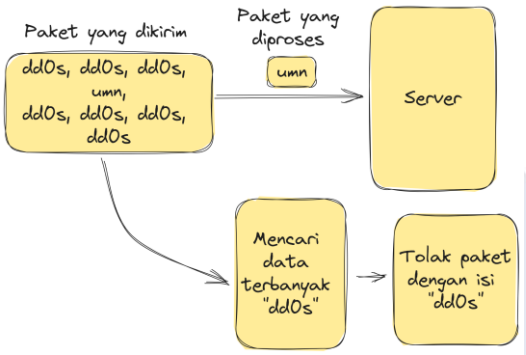


Gambar 3.30 Perbedaan *Atomic* dan *Composite*

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 clasification* 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.

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

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.

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

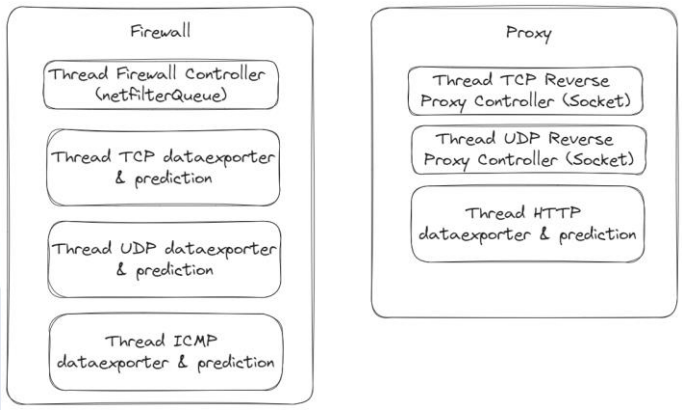
f. F-1 Score

Perbandingan antara rata-rata presisi dengan recall

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

3.9 Perancangan Sistem Firewall

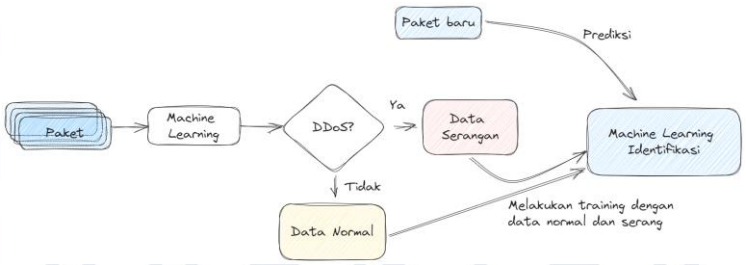
Untuk melakukan prediksi *machine learning* secara *realtime* tanpa mempengaruhi alur koneksi yang ada maka perlu dibuat program dengan paradigma *multiprocessing*. Arsitektur umum dari sistem firewall dan proxy diilustrasikan pada gambar 3.32.



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

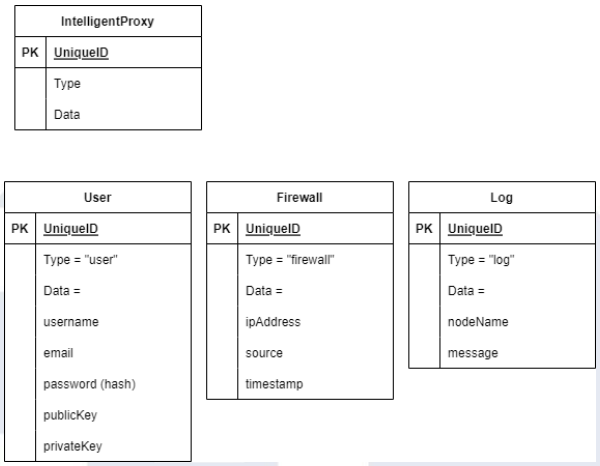
Perancangan sistem distribusi data terdiri dari beberapa proses yaitu perancangan *database* terdistribusi BigchainDB, perancangan sistem backend, dan perancangan sistem frontend, dan evaluasi dengan metrik pengujian sistem distribusi data. Untuk menambahkan informasi identitas penyerang atau dalam hal 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

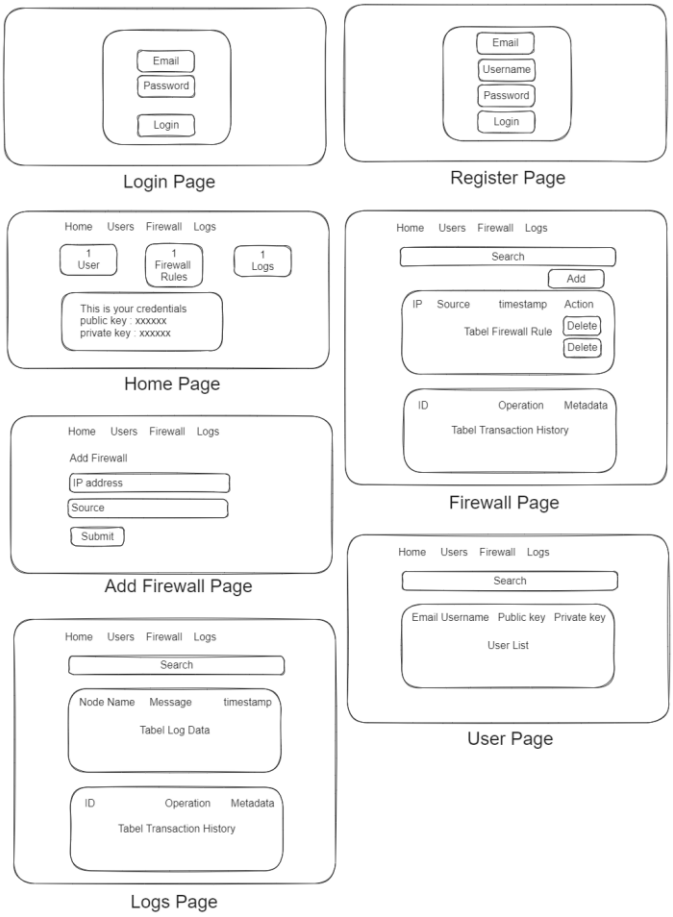
Function	Route	Endpoint	Database Operation	Payload	Pakai JWT ?
User	Membuat user baru (jika blockchain user belum ada maka dibuat)	POST /api/user	Create & Append	1.Keypair {publicKey & privateKey} 2.Username 3.Email 4.Password	Ya
	Menghapus user dengan username	POST /api/user /delete/{username}	Append	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	
	Mengapus 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.

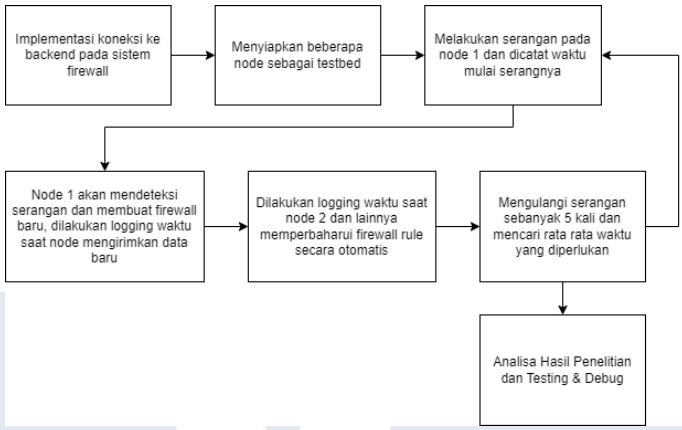


Gambar 3.35 Rancangan Frontend

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

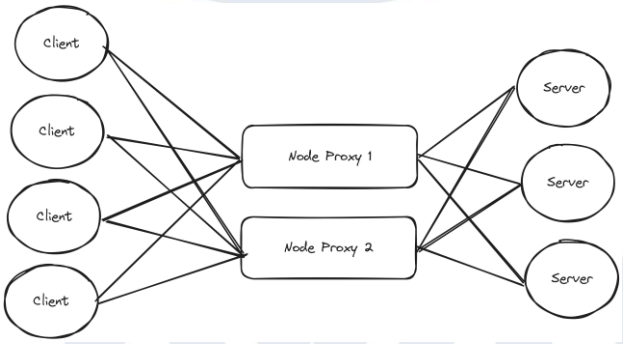
Penelitian ini memiliki tujuan untuk menguji immutabilitas data pada database BigchainDB. Langkah pada penelitian ini adalah melakukan insert data dan update data dengan melakukan *request* ke backend namun dengan *public key* dan *private key* yang salah. Akan diobservasi perilaku dari sistem backend 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 fire*wall rul*e 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.