BAB 3
METODE PENELITIAN

3.1 ALUR PENELITIAN

Penelitian ini dilakukan dalam beberapa tahapan dimulai dari studi literatur hingga tahap analisis data dan kesimpulan, seperti tahapan penelitian pada gambar 3.1.

Gambar 3.1 Diagram Alur Penelitian

Tahap pertama yang dilakukan adalah mencari studi literatur. Tahap ini bertujuan untuk mencari referensi yang dapat dijadikan sebagai acuan dalam pelaksanaan penelitian. Sumber referensi yang digunakan berupa jurnal, buku, artikel *conference*, dan *website* yang berhubungan dengan topik penelitian. Tahap berikutnya yaitu menganalisis kebutuhan *hardware* dan *software* yang diperlukan. Tahap ini bertujuan untuk mempersiapkan perangkat-perangkat yang dibutuhkan dalam perancangan untuk membangun sistem *server* media *streaming* dan *load balancing* hingga pengujian. Tahap ketiga yaitu perancangan topologi jaringan pada layanan *cloud computing*. Pada tahap selanjutnya dilakukan konfigurasi sistem *load balancing* dan *web server* untuk layanan *streaming* serta melakukan pengujian sistem untuk mengetahui sistem dapat berjalan dengan normal. Selanjutnya perancangan skenario pengujian yang bertujuan untuk menentukan metode pengujian yang akan dilakukan berdasarkan *variabel input* pada setiap skenario supaya mendapatkan hasil data yang diharapkan.

Tahap selanjutnya yaitu menganalisis data-data yang telah diperoleh untuk menentukan kinerja dari *web server* yang menerapkan metode *load balancing* untuk layanan *streaming* menggunakan algoritma *least connection* melalui parameter CPU *usage* dan *response time*. Kemudian untuk mengetahui kualitas layanan *streaming* dilakukan analisis saat sistem *streaming* diimplementasikan *load balancing* pada jaringan terhadap data *stream* video sehingga diperoleh parameter *throughput*, *delay*, dan *packet loss* dari sisi *client*. Kesimpulan dibuat dengan mempertimbangkan rumusan masalah dan tujuan penelitian supaya mendapatkan hasil yang sesuai. Saran dibuat untuk perbaikan maupun pengembangan pada penelitian lain tentang topik yang serupa.

3.2 STUDI LITERATUR

Pada diagram alur penelitian tahapan awal dalam penelitian ini yaitu mencari studi literatur yaitu mencari sumber referensi yang berkaitan dengan penelitian ini, mengenai implementasi *load balancing* pada *Openstack* untuk *server* media *streaming* menggunakan algoritma *least connection*. Referensi tersebut akan menjadi acuan pada saat penelitian dan membandingkan kajian teori pada penelitian sebelumnya sehingga dapat dikembangkan menjadi penelitian.

3.3 ANALISIS KEBUTUHAN *HARDWARE* DAN *SOFTWARE*

Pada proses perancangan dan pengambilan data yang akan dilakukan, diperlukan perangkat keras dan perangkat lunak yang akan digunakan sebagai alat dan bahan pendukung dalam penelitian ini, dengan spesifikasi sebagai berikut.

3.3.1 Perangkat Keras (*Hardware*)

Dalam penelitian ini terdapat perangkat keras yang digunakan yaitu berupa perangkat komputer, dengan spesifikasi seperti pada tabel 3.1.

Tabel 3.1 Spesifikasi Perangkat Keras

PC *Host* / *Server*	Sistem Operasi	Windows *Server* 2019 Datacenter
	Processor	Intel® *Core*™ i7-9700KF CPU @ 3.60GHz
	RAM	64 GB *Quad Channel*
	SSD	512 GB
	NIC	TP-LINK Gigabit Ethernet
	VGA	NVIDIA GeForce GTX 1660 Super
PC *Client*	Sistem Operasi	Windows 10 Pro
	Processor	Intel® *Core*™ i7-7700KF CPU @ 3.60GHz
	RAM	8 GB *Single Channel*
	HDD	1.8 TB
	NIC	*Fast* Ethernet Realtek PCIe GbE Family
	VGA	NVIDIA GeForce GTX 1650
PC Studio *Streaming*	Sistem Operasi	Windows 11 *Home Single Language*
	Processor	AMD Ryzen 5 4500U CPU @ 2.38GHz
	RAM	16 GB *Dual Channel*
	SSD	512 GB
	NIC	*Fast* Ethernet Realtek PCIe GbE Family
	VGA	AMD Radeon™ *Graphics*

Pada perangkat keras yang dipakai menggunakan NIC *fast* ethernet yang mempunyai kemampuan dalam mendukung kecepatan akses data sebesar 100

Mbps. Dengan kemampuan akses data tersebut sangat mendukung untuk layanan *streaming* yang akan diuji dengan nilai *bitrate* maksimum yang dikirim 4096 Kbps. Dalam layanan *streaming*, kecepatan data yang digunakan untuk mendapatkan pengalaman pengguna yang terbaik yaitu dengan menyediakan 3 kali dari kecepatan *upload* yang berarti kualitas *bandwidth* internet minimum dapat menyediakan 12 Mbps untuk *download*.

3.3.2 Perangkat Virtual

Dalam penelitian ini menggunakan beberapa perangkat virtual, yang terdiri dari 1 unit *server media streaming* serta mengimplementasikan *cloud environment Openstack* untuk 2 unit *web server streaming* dan 1 unit *load balancer*. Adapun spesifikasi perangkat virtual yang digunakan tertera pada tabel 3.2.

Tabel 3.2 Spesifikasi Perangkat Virtual

Openstack & *Media Streaming Server*	Sistem Operasi	Ubuntu *Server* 20.04
	RAM	12 GB
	Harddisk	250 GB
	Alamat IP	10.212.16.14
Web Server Streaming 1	Sistem Operasi	Ubuntu *Server* 18.04
	RAM	2 GB
	Harddisk	60 GB
	Alamat IP	192.168.100.12
Web Server Streaming 2	Sistem Operasi	Ubuntu *Server* 18.04
	RAM	2 GB
	Harddisk	60 GB
	Alamat IP	192.168.100.13
Load Balancer	Sistem Operasi	Ubuntu *Server* 18.04
	RAM	4 GB
	Harddisk	60 GB
	Alamat IP	192.168.100.15

3.3.3 *Software* Pendukung

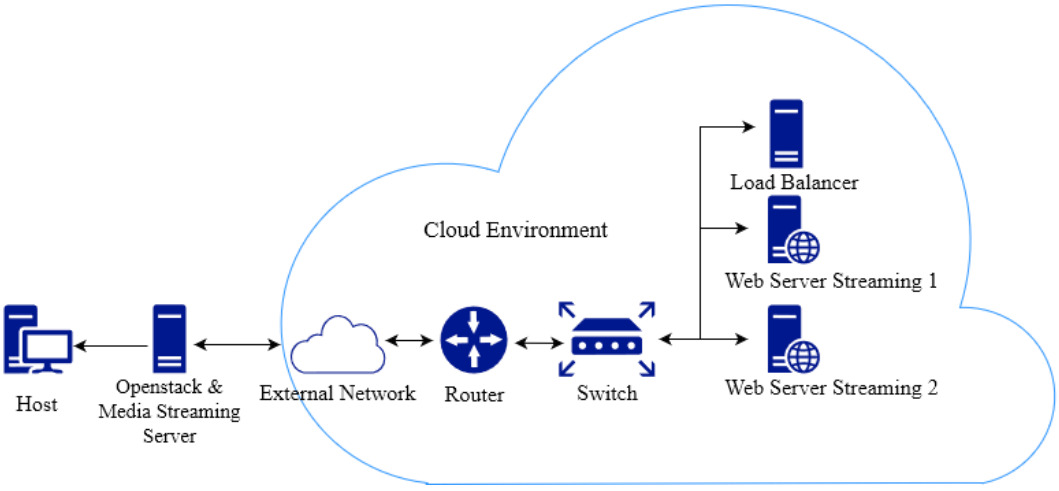
Software pendukung sebagai alat-alat (*tools*) yang diperlukan dalam perancangan hingga proses pengambilan data penelitian ini, ditunjukkan pada tabel 3.3.

Tabel 3.3 *Software* Pendukung

No	Nama *Software*	Versi	Fungsi
1	Virtual*Box*	6.1	Virtualisasi
2	*Openstack*	*Wallaby*	*Cloud computing* OS
3	*Nginx*	1.23.1	*Server* Media *Streaming*
4	*Haproxy*	1.8.8.1	*Load balancer*
5	*Apache*2	2.4.29	*Web server* untuk *streaming*
6	*HTTPerf*	0.9.1	*Tool* pengujian *web server*
7	*Wireshark*	3.6.6	*Network analyzer*
8	OBS Studio	27.2.4	Studio *streaming*

Dalam perancangan *layanan cloud computing* dibangun pada perangkat lunak virtualisasi Virtual*Box* untuk melakukan instalasi dan menjalankan *image* OS Ubuntu *Server* dan untuk *cloud computing* OS yang digunakan yaitu *Openstack Wallaby*. Kemudian menggunakan *Nginx* dalam mengembangkan layanan *streaming* sebagai perangkat lunak untuk melakukan *encoding* media berupa video dan audio menjadi aliran data. *Apache*2 berfungsi menjadi *web server* yang digunakan untuk mendistribusikan layanan *streaming* sebagai antarmuka yang menampilkan halaman *web* berupa media *player* pada *client*. Dengan mengimplemetasikan *Haproxy* sebagai *load balancer* digunakan dalam mendistribusikan beban trafik berdasarkan jumlah koneksi, agar saat terjadi peningkatan jumlah koneksi terhadap *web server* layanan *streaming* masih dapat dimuat. Dalam pengujian, OBS Studio berfungsi sebagai perangkat lunak yang mengirimkan video dan audio. Saat video dan audio dikirimkan dan di*encode*, aliran data akan di distribusikan ke *web server* dan akan dimuat oleh *client* menggunakan *browser*, pada proses tersebut akan di *capture* menggunakan *Wireshark* untuk mengetahui nilai QoS pada jaringan.

3.4 PERANCANGAN TOPOLOGI JARINGAN



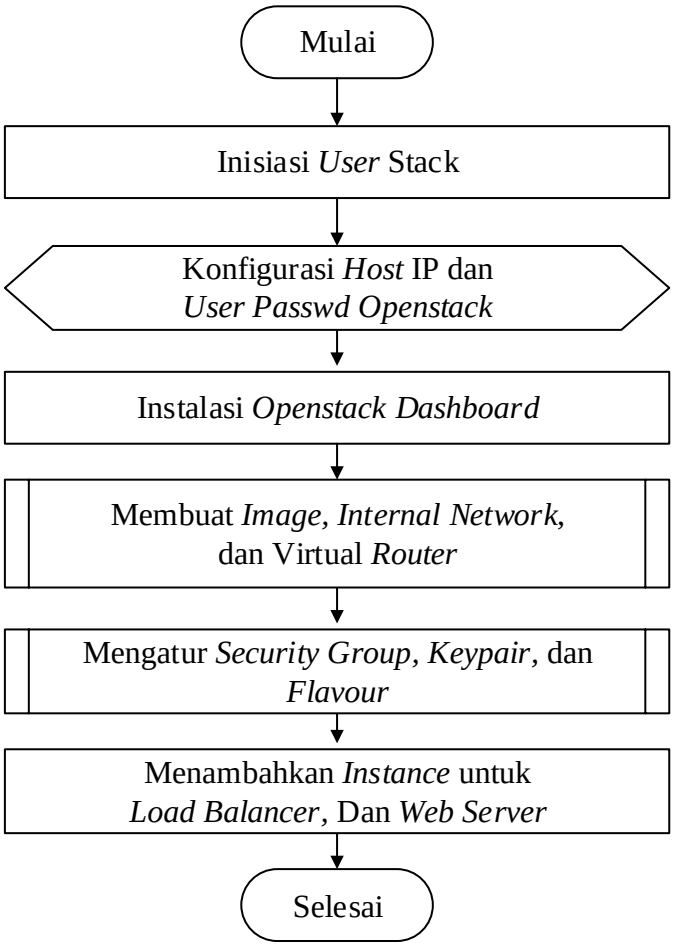
Gambar 3.2 Topologi Jaringan

Pada gambar 3.2 menunjukkan topologi jaringan yang akan digunakan pada penelitian ini terdiri dari perangkat *host* atau tempat implementasi *server*. *External network* dibuat untuk menghubungkan jaringan luar ke seluruh jaringan virtual *machine* di *cloud environment*. Virtual *router* dan *switch* digunakan untuk membuat jaringan internal dari seluruh *web server* dan *load balancer*. Topologi jaringan digunakan sebagai acuan dalam menjalankan sistem *load balancing* dan *web server* untuk menjalankan serta mendistribusikan layanan *streaming*. Proses instalasi *Openstack* berjalan disisi *node host* menggunakan OS *Ubuntu Server* 20.04.

Setelah berhasil melakukan instalasi *Openstack*, dapat dilanjutkan dengan menambahkan 1 virtual *machine* untuk *load balancer*, dan 2 virtual *machine* untuk *web server*. Berdasarkan perancangan topologi jaringan ini, setiap *client* yang terhubung kedalam 1 *network* yang sama dengan *openstack server* maka dapat digunakan untuk mengakses seluruh virtual *machine*. Dengan memanfaatkan *network* yang sama maka *client* dapat melakukan pengujian pada sistem yang telah diimplementasikan di *openstack.*

3.4.1 Konfigurasi *Openstack*

Proses konfigurasi *Openstack* bertujuan mempersiapkan infrastruktur layanan dalam mengimplementasikan topologi jaringan dan sistem *load balancing* dari *web server* serta layanan *streaming* seperti pada gambar 3.3.



Gambar 3.3 Diagram Alur Konfigurasi *Openstack*

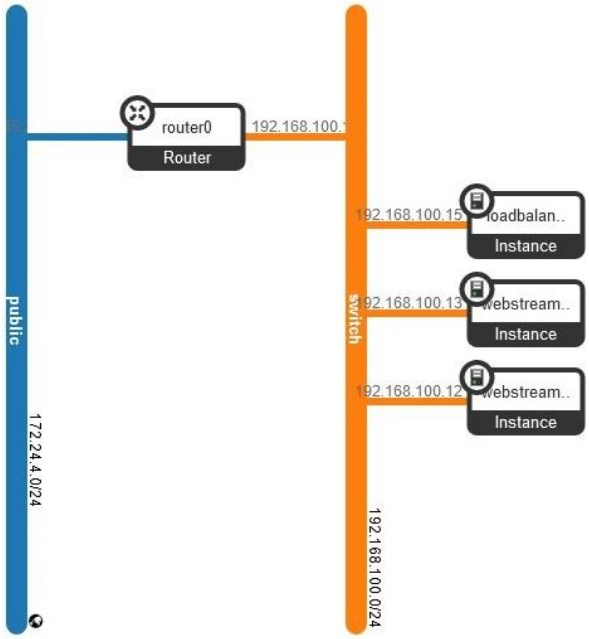
Konfigurasi dasar adalah membuat *user*, *password* dan *host* IP yang digunakan untuk mengakses *dashboard Openstack*. *File* yang digunakan untuk konfigurasi terletak pada direktori */home/devstack* dengan nama *file local.conf.* Perintah yang dipergunakan untuk mengkonfigurasi *Openstack*, dapat menggunakan perintah berikut:

```
[[local|localrc]]
ADMIN_PASSWORD=19101164
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
HOST_IP=10.212.16.14
```

Penggunaan IP statis sebagai *host* IP dari *Openstack server* supaya IP yang digunakan tidak berubah saat mengakses *Openstack dashboard* pada halaman *browser*. Dalam pengimplementasian *server* pada layanan *cloud computing* perlu menambahkan *image* OS yang akan digunakan sebagai dasar sistem operasi untuk virtual *machine* dengan cara menambahkan *image* OS yang telah disediakan. *Layanan server* yang dibuat dapat saling berkomunikasi dengan menggunakan internal *network* dan untuk menghubungkan ke jaringan publik digunakan *virtual router* pada *Openstack.* Pengaturan *security group* perlu diatur supaya *instance* dapat di akses karena *security group* digunakan sebagai *firewall* untuk memfilter protokol yang diizinkan seperti TCP, SSH dan ICMP. Selanjutnya mengatur *keypair* sebagai autentikasi berupa *public key* SSH yang akan digunakan untuk mengakses *instance*. Dan yang perlu dilakukan dalam membuat *instance* adalah mengatur penggunaan penyimpanan, CPU, dan RAM yang akan digunakan. Pada *Openstack* untuk mengatur penggunaan penyimpanan, CPU, dan RAM terdapat pada layanan *flavour*. Setelah selesai melakukan konfigurasi, dapat menambahkan *instance* sesuai kebutuhan berdasarkan kapasitas dari *resource Openstack.*

3.4.2 Implementasi Topologi Jaringan

Setelah membuat topologi jaringan seperti gambar 3.4 maka hasil implementasi *instance* yang telah dibuat dapat dilakukan konfigurasi dengan cara mengakses via SSH melalui perangkat *host*.



Gambar 3.4 Implementasi Topologi Jaringan *Logic* Di *Openstack*

Cara agar *instance* yang telah dibuat dapat terhubung ke jaringan luar atau terhubung ke internet diperlukan *floating* IP. Perangkat *host* dapat melakukan *remote* menggunakan IP yang telah didapatkan dari *floating* IP *web server* 1, dan *web server* 2 sebagai sistem *backend*. Sistem *backend* mempunyai fungsi untuk menampilkan halaman *web* dan menampilkan hasil video *streaming* berbasis HTML5 sebagai media *player*. *Load balancer* mempunyai fungsi sebagai *frontend* yang meneruskan *request client* menuju *web server*. *Switch* digunakan sebagai penghubung seluruh *instance* menggunakan *network* yang sama, kemudian *router* menghubungkan jaringan dengan alamat *network* yang berbeda. *Public* digunakan sebagai *external network* yang berfungsi menghubungkan seluruh *instance* ke jaringan luar. Perangkat *host* selain digunakan untuk mengakses *instance*, dapat juga digunakan sebagai studio untuk mengirimkan video dan audio. Setelah *instance* telah berhasil ditambahkan, dapat memberikan perintah IP add untuk mendapatkan keluaran berupa IP *address* dan MAC *address* dari *instance web streaming* 1 seperti tampilan berikut:

```
root@webstreaming-1:~# ip add
1: lo <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default qlen 1000
        link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host 1o
                valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
                valid_lft forever preferred_lft forever
3: ens7: <BROADCAST,MULITCAST,UP,LOWER_UP> mtu 1450 qdisc fq_codel
state UP group default qlen 1000
        link/ether fa:16:3e:9d:74:06 brd ff:ff:ff:ff:ff:ff
        inet 192.168.100.12/24 brd 192.168.100.255 scope global dynamic ens7
                valid_lft 82991sec preferred_lft 82991sec
        inet6 fe80::f816:3eff:fe9d:7406/64 scope link
                valid_lft forever preferred_lft forever
```

Perintah IP add digunakan untuk melihat keluaran berupa IP *address* dan MAC *address* dari *instance web streaming* 2 seperti tampilan berikut:

```
root@webstreaming-2:~# ip add
1: lo <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default qlen 1000
        link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host 1o
                valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
                valid_lft forever preferred_lft forever
3: ens7: <BROADCAST,MULITCAST,UP,LOWER_UP> mtu 1450 qdisc fq_codel
state UP group default qlen 1000
        link/ether fa:16:3e:9c:18:07 brd ff:ff:ff:ff:ff:ff
        inet 192.168.100.13/24 brd 192.168.100.255 scope global dynamic ens7
                valid_lft 85639sec preferred_lft 85639sec
        inet6 fe80::f816:3eff:fe9c:1807/64 scope link
                valid_lft forever preferred_lft forever
```

Setelah berhasil menambahkan *instance* untuk *load balancer*, kemudian dapat diakses dengan SSH dan perintah IP add untuk melihat keluaran berupa IP *address* dan MAC *address* dari *instance load balancer* seperti tampilan berikut:

```
root@loadbalancer:~# ip add
1: lo <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default qlen 1000
        link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host 1o
                valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
                valid_lft forever preferred_lft forever
3: ens7: <BROADCAST,MULITCAST,UP,LOWER_UP> mtu 1450 qdisc fq_codel
state UP group default qlen 1000
        link/ether fa:16:3e:b7:17:1c brd ff:ff:ff:ff:ff:ff
        inet 192.168.100.15/24 brd 192.168.100.255 scope global dynamic ens7
                valid_lft 86243sec preferred_lft 86243sec
        inet6 fe80::f816:3eff:feb7:171c/64 scope link
                valid_lft forever preferred_lft forever
```

3.4.3 Konfigurasi *Load Balancer*

Proses konfigurasi *load balancer* dilakukan untuk mempersiapkan sistem *load balancing* menggunakan *software HAProxy*. Konfigurasi *load balancer* dilakukan pada 1 *instance* yang telah ditambahkan, kemudian *file* konfigurasi terdapat pada direktori */etc/haproxy/* dengan nama *file haproxy.cfg*. Konfigurasi berfungsi untuk menentukan algoritma dari sistem *load balancing* yang akan digunakan, dan menentukan *web server* yang akan menjadi tujuan oleh sistem. Perintah yang dipergunakan untuk konfigurasi *Load Balancer* dapat menggunakan perintah berikut:

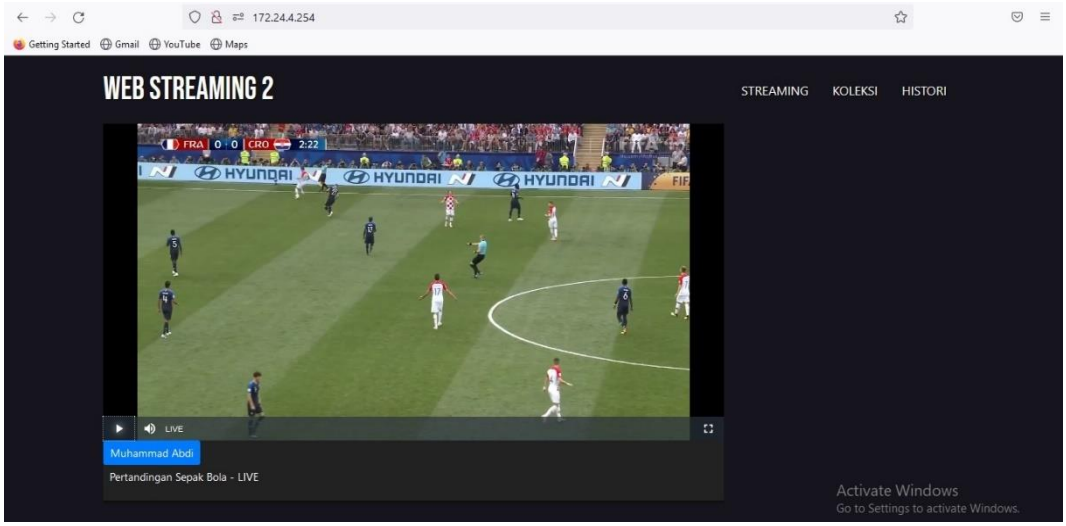
```
frontend header
    bind *:80
    mode http
    default_backend footer

backend footer
     mode http
     balance leastconn
     server web1 172.24.4.97:80 check
     server web2 172.24.4.233:80 check
```

Pada perintah konfigurasi *load balancer*, sistem akan bekerja menggunakan algoritma *least connetion*. Dalam konfigurasi juga menetapkan IP dari *web server* yang digunakan sebagai tujuan dan target dari sistem *load balancing* dengan

mengelola setiap trafik data yang masuk baik berupa *request* maupun koneksi yang terjadi dari *client* dan *server*. Trafik data yang masuk akan diteruskan berdasarkan algoritma yang digunakan kepada *web server* tujuan.

3.4.4 Konfigurasi *Web Server*

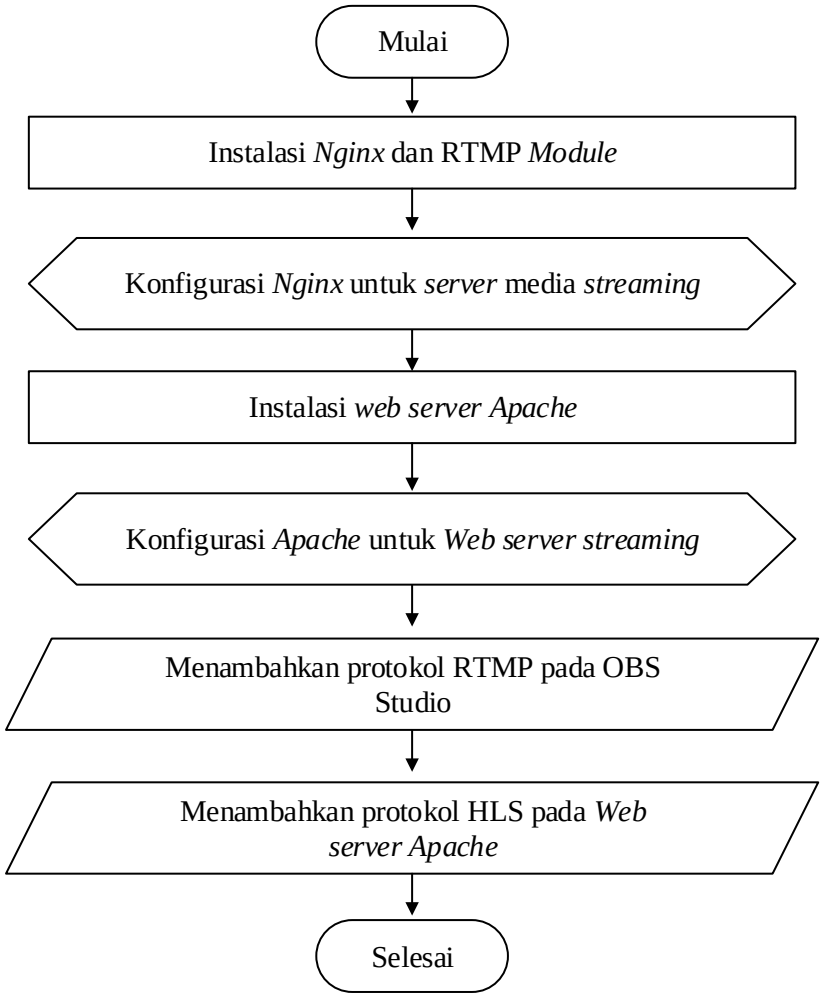
Proses konfigurasi *web server* dilakukan untuk mempersiapkan layanan *streaming* pada *web server*. Konfigurasi *web server* dilakukan pada 2 *instance* yang telah ditambahkan, kemudian *file* konfigurasi terdapat pada direktori */var/www/html/* dengan nama *file index.html*. Instalasi *web server* menggunakan *Apache* pada setiap *instance*. Dengan mengubah isi *file index.html* dapat digunakan untuk mengganti tampilan *web server* sebagai media *player* menggunakan HTML5. Dalam konfigurasi dilakukan penambahan *file playlist* yang digunakan untuk memutar media yang telah di segmentasi menggunakan protokol HLS.



Gambar 3.5 Tampilan *Web Server Streaming*

3.4.5 Konfigurasi *Server* Media *Streaming*

Layanan *streaming* dibuat menggunakan *Nginx* sebagai *server* media *streaming* pada perangkat virtual seperti alur pada gambar 3.6. Kemudian diperlukan konfigurasi pada *server* media *streaming* untuk mendistribusikan layanan *streaming* ke *web server* pada *instance Openstack* dan mengatur protokol transmisi media yang akan digunakan.



Gambar 3.6 Diagram Alur *Server* Media *Streaming*

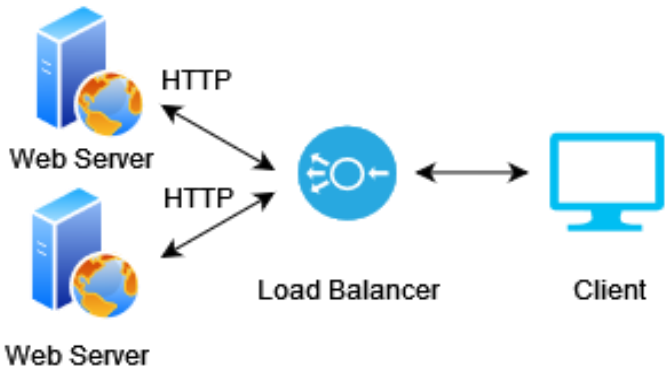
Dalam konfigurasi *server* media *streaming* diperlukan instalasi modul RTMP sebagai protokol yang akan digunakan yaitu protokol RTMP pada OBS studio dan protokol HLS pada *web server* untuk *streaming*. Konfigurasi dilakukan dengan menambahkan perintah pada direktori */usr/local/nginx/conf/* dengan nama *file nginx.conf*. Perintah yang dipergunakan untuk mengkonfigurasi *server* media *streaming* dapat menggunakan perintah berikut:

```
rtmp {
    server {
        listen 1935;
        application live {
            live on;
```

```
            interleave on;
            hls on;
            hls_path /usr/local/nginx/html/live;
            hls_fragment 2s;
        }
    }
}
http {
    server {
        listen 8080;
        location /tv {
            root /usr/local/nginx/html/live;
        }
    }
    types {
        application/vnd.apple.mpegurl m3u8;
        video/mp2t ts;
        text/html html;
    }
}
```

3.5 UJI COBA SISTEM *LOAD BALANCING*

Uji coba dilakukan untuk memantau pembagian beban trafik pada sistem *load balancing* dapat berjalan dengan normal. Sebelum menjalankan sistem *load balancing*, perlu terlebih dahulu melakukan konfigurasi, sehingga hasil yang didapatkan saat *client* mengakses IP milik *load balancer* maka akan langsung diarahkan ke *web server streaming* yang telah disiapkan. Sehingga dengan sistem *load balancing* ini, maka sistem akan secara otomatis mengalokasi *request* berdasarkan koneksi yang terhubung dalam jaringan.



Gambar 3.7 Sistem Kerja *Load Balancing*

Pada gambar 3.7 menunjukkan *client* mengirimkan *request* kemudian *load balancer* akan meneruskan *request* ke *server* berdasarkan algoritma yang digunakan yaitu *least connection* dengan melihat koneksi atau sambungan aktif dari *client* dan *server* yang paling sedikit pada setiap *server* nya dengan menghitung kapasitas koneksi saat ini pada *web server* 1 dan *web server* 2.

Tabel 3.4 Pengujian Akses Sistem *Load Balancing* Dari *Web Server*

Pengujian	***Web Server* Yang Melayani**	
	***Web Server* 1**	***Web Server* 2**
Click Ke-1	√	
Click Ke-2	√	
Click Ke-3		√
Click Ke-4	√	
Click Ke-5		√
Click Ke-6	√	
Click Ke-7		√
Click Ke-8	√	
Click Ke-9		√
Click Ke-10	√	

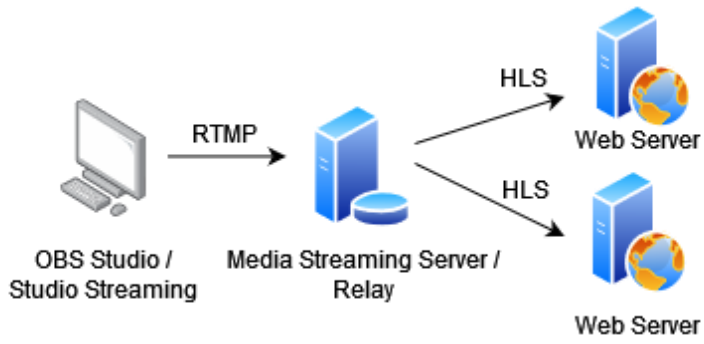
Uji coba dilakukan dengan mengakses halaman dari *web server streaming* menggunakan IP *load balancer* melalui *browser web* pada PC *client*. Dengan mengakses *load balancer* maka diketahui saat *click* ke-1 yang melayani yaitu *web server* 1, kemudian pada *click* ke-3 yang melayani yaitu *web server* 2 seperti tabel 3.4 yang ditunjukkan dengan notasi (√). *Web server* yang melayani permintaan *client* akan terus berganti sesuai dengan algoritma *least connection* yang digambarkan seperti alur algoritma berikut:

```
server = 1,2,…….,n                          -> n : menunjukkan jumlah web server
max connection capacity (web server n) = 200 conn/s
incoming connection = X conn/s               -> X : jumlah koneksi yang masuk

load balancer system information
for (int C = X; C > 0; C--)
{
If (current connection capacity (web server1) < current connection capacity (web
                                                server2)) {
    current connection capacity (web server2) = max connection capacity (web server2)
                                                – 1
    go to web server2
```

```
    }

If (current connection capacity (web server1) > current connection capacity (web
                                                    server2)){
    current connection capacity (web server1) = max connection capacity (web server1)
                                                    – 1
    go to web server1
    }
}
```

3.6 UJI COBA SISTEM MEDIA *STREAMING*

Uji coba dilakukan dengan menggunakan PC studio *streaming* dan *software* OBS studio untuk mengirimkan video. Setelah dilakukan konfigurasi pada *server* media *streaming*, layanan *streaming* sudah dapat berjalan. *Server* media *streaming* digunakan untuk memproses video *stream* dengan menjalankan protokol RTMP pada perangkat lunak *Nginx* yang nantinya akan diubah menjadi protokol HLS untuk menyediakan layanan *streaming* dengan tampilan antarmuka HTML5 sebagai media *player* pada *web server* seperti pada gambar 3.8. Uji coba dilakukan untuk memantau layanan *streaming* telah tersedia dan dapat berjalan dengan lancar.



Gambar 3.8 Sistem Kerja *Server* Media *Streaming*

3.7 SKENARIO PENGUJIAN

3.7.1 Pengujian *Web Server* Berdasarkan Jumlah Koneksi

Pengujian ini bertujuan untuk mendapatkan hasil dari kinerja *web server* saat layanan *streaming* dalam keadaan sepi pengunjung dan ramai pengunjung. Terdapat 3 variasi pengujian berdasarkan jumlah koneksi yang disimulasikan dari

client berbeda-beda. Dari pengujian *web server* akan didapatkan nilai rata-rata dari setiap parameter yang telah ditentukan berdasarkan dari kinerja *server* dalam melayani *request client*. Pengujian beban kerja dilakukan menggunakan *software HTTPerf* untuk mendapatkan nilai performansi dari penggunaan CPU dan *response time* sistem *load balancing* menggunakan variasi jumlah koneksi yang disimulasikan.

Tabel 3.5 Pengujian Berdasarkan Jumlah Koneksi

Jumlah Koneksi	Koneksi per detik	Jumlah Pengujian
500	100	30
2.000	100	30
5.000	100	30

3.7.2 Pengujian Kualitas Layanan *Streaming* Pada Jaringan Berdasarkan *Frame Rate* Dan *Bitrate*

Pengujian ini bertujuan untuk mendapatkan hasil dari kualitas layanan *streaming* ketika diterapkan *load balancing* pada jaringan. Pengujian dilakukan dengan cara meng-*capture* transmisi paket-paket video *streaming* dari *server* ke *client* menggunakan *software Wireshark*. Pengujian dilakukan dengan jumlah maksimum *client* secara real sebanyak 10 PC untuk pengujian yaitu dengan mengubah *frame rate* video mulai dari 25 fps dan 30 fps. Selain, mengubah *frame rate* juga dilakukan uji coba dengan mengubah *bitrate* mulai dari 512 Kbps, 1024 Kbps, 2048 Kbps dan 4096 Kbps dengan durasi video *streaming* yang ditayangkan selama 1 jam dengan nilai kualitas resolusi 1080p, 30 fps dan ukuran video *input* 3GB. Pengujian dilakukan untuk mendapatkan nilai kualitas layanan *streaming* berdasarkan koneksi yang terhubung dengan *client* sehingga didapatkan nilai *throughput*, *delay*, dan *packet loss*.

3.8 PROSES PENGUJIAN

Dalam penelitian ini, simulasi pengujian *streaming* menggunakan PC yang sudah di instalasi *software* OBS Studio sebagai studio *streaming* untuk mengirimkan video dan melakukan proses *upload* pada jaringan laboratorium

programming. Sebelum melakukan pengujian dilakukan pengukuran *speedtest* jaringan laboratorium *programming* dan didapatkan hasil *upload* rentang 17 – 19 Mbps dan hasil *download* rentang 18 – 20 Mbps. Adapun rekomendasi [37] yang dapat digunakan dalam melakukan layanan *streaming* dengan mengatur nilai kecepatan bit atau nilai *bitrate* dari variabel video seperti tabel 3.6.

Tabel 3.6 Rekomendasi Rentang Nilai Bitrate Video [37]

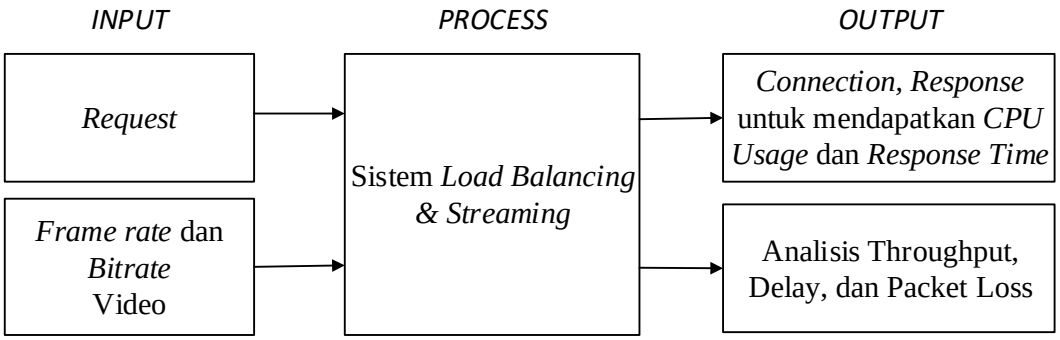
Resolusi Video	**Kecepatan Bit Video**
1080p	3000 – 6000 Kbps
720p	1500 – 4000 Kbps
480p	500 – 2000 Kbps

Dengan mengatur nilai *bitrate*, sistem akan secara otomatis mengetahui resolusi dan kecepatan *frame* berdasarkan hasil video yang di *streaming*. Selain menyesuaikan nilai *bitrate* untuk mengatur kualitas video yang dikirimkan dalam *streaming*, adapun setelan studio yang dapat digunakan untuk proses *encode* video seperti tabel 3.7 dalam mendistribusikan layanan *streaming*.

Tabel 3.7 Rekomendasi Setelan *Encoder* [37]

Protokol	*Streaming* RTMP
***Codec* Video**	H.264
Kecepatan *Frame*	Hingga 60 fps
***Codec* Audio**	AAC atau MP3
Kecepatan Bit Audio	128 Kbps Stereo

Dengan layanan *streaming* menggunakan HTML5 dapat menjadi rekomendasi karena mendukung konten audio dan video serta didukung berbagai jenis *browser* seperti *mozilla firefox* dan *google chrome* [37].



Gambar 3.9 Blok Diagram Pengujian

Pada gambar 3.9 menujukkan proses pengujian dan pengambilan data yang digunakan untuk mengetahui kinerja dari *web server streaming*. Proses pengujian menggunakan *HTTPerf* dengan cara mensimulasikan jumlah koneksi yang dikirimkan ke sistem *load* balancing dan pengujian menggunakan 10 *client* secara real untuk mendapatkan kualitas layanan *streaming* berdasarkan pengaruh *frame rate* dan *bitrate*.

HTTPerf merupakan perangkat lunak untuk melakukan pengujian terhadap daya tahan dari *web server* dalam menangani permintaan dan merespon. Penggunaan perangkat lunak tersebut untuk mendapatkan nilai CPU *Usage* dan *response time* dari *load balancer* dalam mendistribusikan permintaan *client-server* dan proses dari *server-client*. Pengujian sistem *load balancing* menggunakan simulasi dengan *stress test* berdasarkan jumlah koneksi yang terjadi pada sistem *load balancing* menggunakan metode *request* HTTP GET. Pengujian dilakukan dengan dengan mengakses IP *floating* milik *load balancer* dan diarahkan ke port 80 yang merupakan protokol HTTP. Adapun perintah yang digunakan pada *HTTPerf* seperti perintah –rate digunakan untuk mengalokasikan jumlah koneksi dalam setiap detiknya sedangkan perintah –num-conn merupakan total jumlah koneksi yang akan disimulasikan dengan membuat jumlah koneksi sesuai kebutuhan pengujian. Adapun perintah yang dipergunakan untuk melakukan pengujian sistem *load balancing* sehingga didapatkan keluaran berupa hasil CPU *usage* dan *response time* seperti tampilan berikut:

```
ubuntu@client-testing:~$ httperf --server 172.24.4.254 --port 80 --rate 100 --num-conn 5000
httperf --client=0/1 --server=172.24.4.254 --port=80 --uri=/ --rate=100 --send-buffer=4096 --recv-buffer=16384 --num-conns=5000 --num-calls=1
httperf: warning: open file limit > FD_SETSIZE; limiting max. # of open files to FD_SETSIZE
Maximum connect burst length: 59

Total: connections 4864 requests 4864 replies 4809 test-duration 132.741 s

Connection rate: 36.6 conn/s (27.3 ms/conn, <=1022 concurrent connections)
Connection time [ms]: min 8.7 avg 6582.4 max 46247.6 median 6091.5 stddev 4071.2
Connection time [ms]: connect 776.8
Connection length [replies/conn]: 1.000

Request rate: 36.6 req/s (27.3 ms/req)
Request size [B]: 65.0

Reply rate [replies/s]: min 0.0 avg 36.5 max 118.4 stddev 44.0 (26 samples)
Reply time [ms]: response 5526.4 transfer 282.9
Reply size [B]: header 254.0 content 4499.0 footer 0.0 (total 7429.0)
Reply status: 1xx=0 2xx=4809 3xx=0 4xx=0 5xx=0

CPU time [s]: user 11.88 system 11.39 (user 8.8% system 83.9% total 92.7%)
Net I/O: 170.5 KB/s (1.4*10^6 bps)

Errors: total 191 client-timo 0 socket-timo 0 connrefused 0 connreset 55
Errors: fd-unavail 136 addrunavail 0 ftab-full 0 other 0
```

Saat pengujian maupun setelah pengujian dilakukan, dapat dipantau pada *report* statistik dari sistem *load balancing* seperti pada tabel 3.8 dengan cara mengakses IP *load balancer* dan *port* yang sudah dikonfigurasi yaitu 172.24.4.254:1936 di *browser*. Dari hasil *report* diketahui bahwa tabel *header* menunjukkan *frontend* adalah tempat *client* terhubung. Saat *request* masuk ke *load balancer* dan saat *respons*e dikembalikan ke *client*, paket data akan melewati *frontend*. Tabel *footer* menunjukkan *trafik* jaringan yang didistribusikan oleh *load balancer* ke *web server* dan diketahui dari kolom *session* memperlihatkan jumlah *koneksi* antara *client* dan *server*. Kolom *Max* menampilkan sesi terbanyak yang pernah dibuat secara bersamaan. Kolom limit menampilkan maksimum koneksi yang dapat ditangani dalam periode waktu tertentu. Dari hasil tersebut, *load balancer* melakukan proses total 5000 koneksi dan mendistribusikan trafik yang masuk ke *web server* 1 sebanyak 2503 *session* dan *web server* 2 sebanyak 2497

session berdasarkan algoritma *least connection*. Data yang masuk dan keluar dari *load balancer* maupun dari *web server* dapat dilihat pada kolom *Bytes*.

Tabel 3.8 Hasil *Report* Dari Sistem *Load Balancing*

header							
	Sessions					Bytes	
	Max	Limit (conn/s)	Total	LbTot	Last	*In*	*Out*
frontend	89	2.000	5.000			325.000	23.765.000

footer							
	Sessions					Bytes	
	Max	Limit (conn/s)	Total	LbTot	Last	*In*	*Out*
web1	29	-	2.503	2.503	1s	162.692	11.890.759
web2	28	-	2.497	2.497	1s	162.305	11.868.241
Backend	57	200	5.000	5.000	1s	325.000	23.765.000

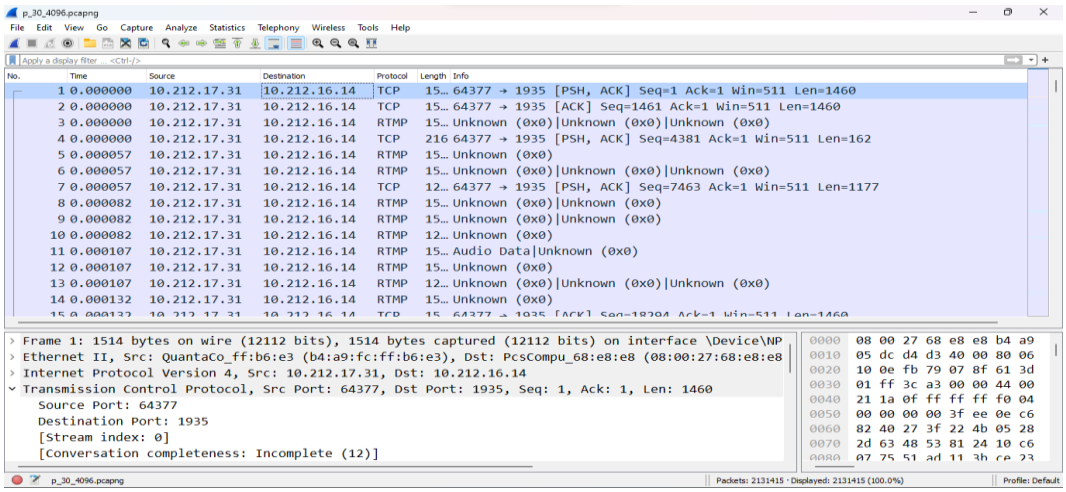
Saat dilakukan pengujian dengan jumlah koneksi yang disimulasikan, maka dapat dilakukan pemantauan proses yang berjalan pada setiap *web server*. Nilai persentase proses pada sistem dapat dilihat dengan memberikan perintah sar 2 seperti tabel 3.9 yang menampilkan persentase kerja sistem dan akan terus diperbarui setiap 2 detik. Ketika sistem tidak sedang berjalan maka nilai yang muncul pada %*system* akan mendekati persentase 0% dan sistem akan dalam kondisi *idle* karena tidak terdapat permintaan yang masuk untuk diproses oleh *web server*. Kondisi %*user* menunjukkan persentasi penggunaan CPU untuk aplikasi di tingkat pengguna dan kondisi %io*wait* menunjukkan persentase waktu CPU menganggur dengan kondisi I/O *disk* yang tertunda. Dan apabila setiap komponen pada sistem baik dari %*user*, %*system*, %io*wait*, dan %*idle* dijumlahkan maka nilai yang didapatkan akan 100%.

Tabel 3.9 Pemantauan *System* Pada *Web Server*

ubuntu@webstreaming1:`$ sar 2					
Linux 4.15.0-197-generic (webstreaming1) 01/10/23 _x86_64(2 CPU)					
05:26:37	**CPU**	**%user**	**%system**	**%iowait**	**%idle**
05:26:39	all	4,74	12,66	0,00	82,59
05:26:41	all	2,46	7,13	0,00	90,42
05:26:43	all	0,25	0,50	0,00	99,24

ubuntu@webstreaming1:`$ sar 2					
Linux 4.15.0-197-generic (webstreaming1) 01/10/23 _x86_64(2 CPU)					
05:26:37	**CPU**	**%user**	**%system**	**%iowait**	**%idle**
05:26:45	all	12,77	33,42	0,00	53,80
05:26:47	all	4,07	12,47	0,00	83,47
05:26:49	all	4,59	8,65	14,50	72,70
05:26:51	all	4,61	14,41	5,76	75,22
05:26:53	all	5,83	20,25	0,00	73,93
05:26:55	all	4,29	13,94	0,00	88,97
05:26:57	all	5,43	14,67	0,00	79,89
05:26:59	all	3,8	13,04	0,00	83,15
05:27:01	all	5,74	15,3	0,00	78,96
05:27:03	all	5,16	15,76	0,00	79,08

Pengambilan data menggunakan *Wireshark* dengan cara meng-*capture packet* yang ada pada jaringan berupa protokol TCP sehingga didapatkan *statistics* yang menunjukkan nilai *measurement* berupa *throughput, delay* dan *packet loss.* Berdasarkan *client* secara real sebanyak 10 PC pada sisi *download*. Dilakukan analisa dari hasil yang didapatkan untuk mengetahui nilai dari kualitas layanan *streaming* terhadap koneksi yang terjadi pada jaringan.



Gambar 3.10 *Capture* Paket Data Menggunakan *Wireshark*

Pada gambar 3.10 menunjukkan proses *upload* data video untuk *streaming*. Pada pengujian *wireshark* terlihat paket yang digunakan yaitu RTMP sebagai protokol transmisi media secara *realtime* dengan menggunakan TCP sebagai

protokol transmisi pada layer *transport*. Protokol RTMP secara sistem menggunakan *port* 1935 sehingga saat mengirimkan video, yang menjadi *destination port* yaitu 1935 terlihat pada hasil *capture* paket.