Volume 3, No.1 2022

JISE

Journal of Internet and Software Engineering

ISSN 2797-9016

https://ugm.id/jise

The journal published by
Department of Electrical Engineering and Informatics
Vocational College, Universitas Gadjah Mada

JISE
Journal of Internet and Software Engineering
Volume 3, No.1 2022

EDITORAL BOARD
Journal of Internet and Software Engineering (JISE)

https://ugm.id/jise

The journal published by
Department of Electrical Engineering and Informatics
Vocational College, Universitas Gadjah Mada
Sekip unit III, Caturtunggal, Terban,
Kec. Gondokusuman, Kab. Sleman, D.I. Yogyakarta 55281

JISE
Journal of Internet and Software Engineering
Volume 3, No.1 2022

https://ugm.id/jise

The journal published by
Department of Electrical Engineering and Informatics
Vocational College, Universitas Gadjah Mada

Analisis dan Implementasi Wireless Access Point Berbasis Raspberry Pi dan Pemberitahuan Akses Pengguna Menggunakan Telegram

Nashruddin Putra Pratama[1], Unan Yusmaniar Oktiawati[1,*]
[1]Departemen Teknik Elektro dan Informatika, Sekolah Vokasi, Universitas Gadjah Mada;
nashruddin.putra.p@mail.ugm.ac.id
[*]Korespondensi: unan_yusmaniar@ugm.ac.id;

Abstract – Internet network technology has become a human need in almost every activity. Communities in rural areas experiencing a lack of fiber optic network infrastructure need a tool to access the internet that can be used together in an affordable and efficient manner. This study tries to provide a solution to these problems. Raspberry Pi-based Wireless Access Point uses a USB modem which functions to build a Wireless Local Area Network. Then, a network quality test is carried out by calculating QoS on network traffic including packet loss, throughput and latency with scenarios based on a distance of 0 meter, 4 meter and 8 meter between the Wireless Access Point and the client by conducting online meetings and video streaming. Furthermore, to make it easier to monitor the Raspberry Pi device that acts as a Wireless Access Point, it will send user information that is connected via the Telegram bot.

Keywords – WLAN, USB modem, Wireless Access Point, QoS, Telegram

Intisari – Teknologi jaringan internet menjadi kebutuhan manusia hampir disetiap aktifitasnya. Masyarakat di daerah yang mengalami kekurangan infrastruktur jaringan serat optik membutuhkan sebuah alat dalam mengakses internet yang dapat digunakan bersama-sama secara terjangkau dan efisien. Penelitian ini mencoba memberikan solusi atas permasalahan tersebut. Wireless Access Point berbasis Raspberry Pi menggunakan USB modem yang berfungsi membangun Wireless Local Area Network. Kemudian, dilakukan pengujian kualitas jaringan dengan menghitung QoS pada network traffic meliputi packet loss, throughput dan latency dengan skenario berdasarkan jarak 0 meter, 4 meter dan 8 meter antara Wireless Acces Point dan client dengan melakukan kegiatan online meeting dan video streaming. Selanjutnya, untuk mempermudah pengawasan perangkat Raspberry Pi yang berperan sebagai Wireless Access Point akan mengirim informasi pengguna yang terhubung melalui bot Telegram.

Kata kunci – WLAN, USB modem, Wireless Access Point, QoS, Telegram

I. PENDAHULUAN

Dewasa ini peran Wi-Fi sangatlah penting dalam jasanya menghubungkan perangkat komputer dan *smartphone* secara nirkabel karena fleksibel dan mudah digunakan, secara berdampingan teknologi ini terus mengalami peningkatan instalasinya di kota-kota besar seperti di area Sekolah, Universitas, Perkantoran, hingga ditempat wisata kuliner dan hiburan layaknya *Mall, Cafe* dan lainnya. Tentunya tanpa kita sadari masih terdapat kekurangan infrastruktur jaringan kabel serat optik atau telepon di daerah yang jauh dari kawasan perkotaan.

Maka dari itu alat yang bisa menjadi solusi pada persoalan tersebut adalah *Wireless Access Point* (WAP) berbasis Raspberry Pi yang berkolaborasi dengan USB modem sebagai penghubung ke penyedia jasa ISP (*Internet Service Provider*) melalui jaringan seluler dengan maksud membuat perangkat *Wi-Fi* yang tidak bergantung pada jaringan kabel serat optik serta dapat mengirim pemberitahuan akses pengguna berupa nama dan *ip address* melalui bot Telegram.

II. KAJIAN PUSTAKA

A. Jaringan Komputer

Jaringan komputer adalah hubungan dua simpul yang umumnya berupa komputer atau yang lebih ditujukan untuk melakukan pertukaran data dan informasi atau untuk berbagi perangkat lunak, perangkat keras hingga dapat berbagi kekuatan pemrosesan [1]. Jaringan komputer pada umumnya terdapat kumpulan beberapa komputer dan perangkat lain seperti *router*, *switch*, dan sebagainya yang saling terhubung satu sama lain melalui kabel dan nirkabel. Tujuan utama dibangunnya suatu sistem jaringan komputer adalah untuk membawa data serta informasi dari sudut pengirim menuju ke penerima secara cepat dan tepat tanpa adanya kerancuan dari media transmisi atau media komunikasi tertentu [2]. *Wireless Local Area Network* secara pemahaman sebenarnya hampir serupa dengan jaringan *Local Area Network*, hanya saja setiap *node* pada jaringan WLAN membutuhkan *wireless device* agar saling terhubung dengan *node* pada WLAN serta menggunakan kanal frekuensi yang selaras. Tidak seperti jaringan LAN (*Local Area Network*) yang membutuhkan kabel disetiap *node* untuk saling terhubung [3].

B. *Quality of Service*

1) *Throughput*

Throuhgput adalah istilah yang menjelaskan banyak bit yang diterima dalam jangka waktu tertentu dengan satuan bit per *second* yang diperoleh dari nilai data sebenarnya.

$$throughput = \frac{\sum paket\ yang\ berhasil\ dikirim\ x\ ukuran\ paket}{total\ waktu\ pengiriman} \quad (1)$$

2) *Packet Loss Ratio*

Packet Loss Ratio adalah jumlah paket yang tidak diterima dibandingkan dengan jumlah seluruh paket yang dikirimkan atau di transmisikan.

$$PLR = \frac{paket\ terkirim - paket\ diterima}{\sum paket\ terkirim}\ x\ 100\% \quad (2)$$

3) *Latency*

Latency adalah jumlah waktu keseluruhan dalam proses satu kali pengamatan, waktu yang dibutuhkan dalam satu kali pengiriman paket data dibagi dengan keseluruhan usaha pengiriman paket yang berhasil dalam pengamatan.

$$Latency = \frac{\sum waktu\ pengiriman\ dalam\ satu\ kali\ pengamatan}{\sum usaha\ pengiriman\ paket\ berhasil} \quad (3)$$

C. Modem

Modem merupakan akronim dari Modulator Demodulator. Modulator adalah suatu alat dengan fungsi melakukan proses modulasi yang bertujuan mengubah sinyal informasi kedalam sinyal pembawa (*carrier signal*) untuk dikirimkan ke alamat tujuan, sedangkan Demodulator ialah bagian yang melakukan proses demodulasi dengan tujuan memisahkan sinyal informasi (berisi berbagai jenis data) dari *carrier signal* yang diterima sehingga informasi tersebut dapat di terima dengan baik tanpa ada kerusakan pada datanya. Selain itu Modem adalah bentuk gabungan dari modulator dan demodulator yang berarti dapat melakukan komunikasi dua arah sekaligus [4].

D. Wireless Access Point

Wireless Access Point adalah sebuah *device* tambahan dari suatu jaringan komputer lokal yang terhubung dengan *router* dan lainya untuk tujuan menambah jumlah pengguna yang terhubung dan memperluas cakupan penyebaranya melalui media perantara sinyal radio atau biasa disebut dengan nirkabel, frekuensi sinyal radio yang digunakan kisaran 2.4Ghz hingga 5Ghz, selain itu terdapat komponen antena dan *transceiver* pada perangkat *Wireless Access Point* (WAP) yang berfungsi untuk media komunikasi untuk memancarkan sinyal dan menerima sinyal [5].

E. Raspberry Pi

Raspberry Pi (biasa disebut sebagai RasPi) adalah sebuah SBC (*Single Board Computer*) seukuran kartu ATM yang dikembangkan oleh Yayasan Raspberry Pi di Inggris (UK) dengan harapan untuk memicu pengajaran ilmu komputer dasar di sekolah-sekolah [6].

F. Telegram

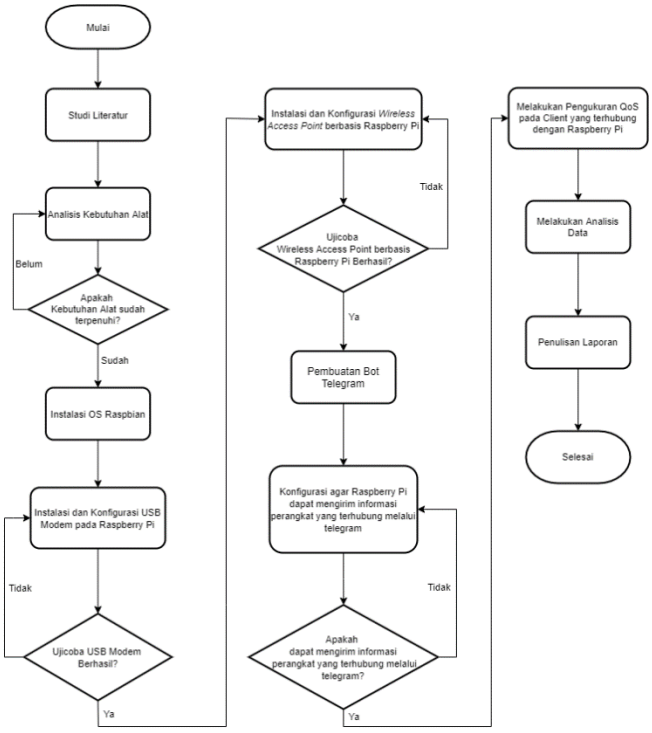
Telegram merupakan aplikasi *instant messaging* yang dapat digunakan secara gratis. Selain itu, aplikasi Telegram menyediakan sebuah *source code* yang dapat digunakan atau aplikasi dengan sifat *open source* [7].

G. Wireshark

Wireshark adalah salah satu perangkat lunak yang digunakan untuk menganalisa jaringan, memfilter protokol, paket [8]. Aplikasi ini banyak digunakan oleh *Network Engineer, Network Analyst, Network Administrator*.

III. METODE PENELITIAN

A. Tahapan Penelitian

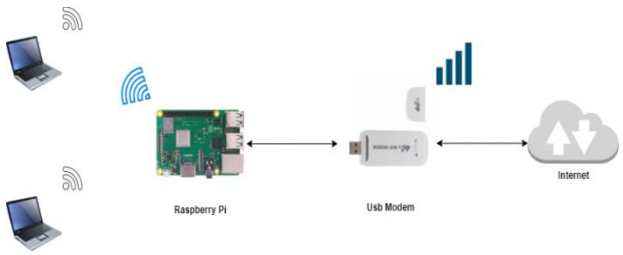


Gambar 1. Tahapan Penelitian

B. Alat dan Bahan

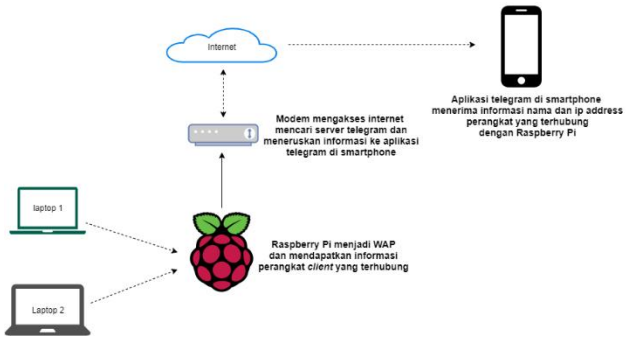
1) Perangkat Keras
   - Raspberry Pi 3 Model b
   - USB Modem
   - Laptop
2) Perangkat Lunak
   - Wireshark
   - Telegram

C. Topologi Implementasi



Gambar 2. Topologi Implementasi

D. Skema Jaringan Penelitian

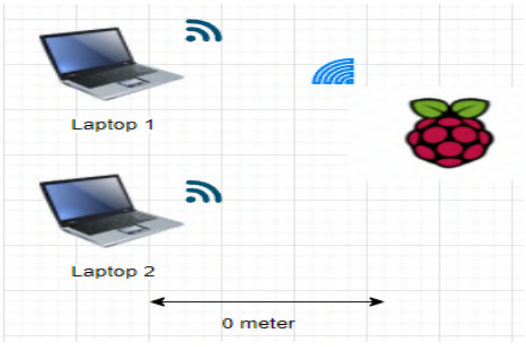


Gambar 3. Skema Jaringan Penelitian

E. Skenario Pengujian

1) *Skenario Pengujian 1 (Pengujian dilakukan dengan jarak 0 meter)*

Pengujian berdasarkan jarak 0 meter tanpa halangan diantara *Wireless Access Point* berbasis Raspberry Pi dengan 2 *client* ketika melakukan sistem pengujian *online meeting* dan *video streaming*. Jadi yang diuji adalah proses pengiriman data dari *Wireless Access Point* berbasis Raspberry Pi menuju setiap *client*, sehingga dapat diketahui *Quality of Service* dari proses pengiriman data tersebut.

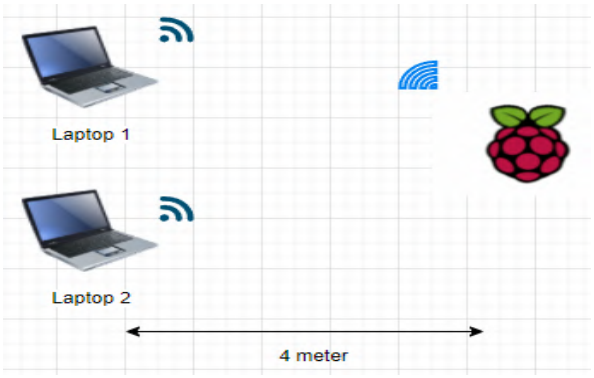


Gambar 4. Jarak antar perangkat WAP dan client 0 meter

Pada *Gambar* 4 merupakan tampilan topologi antar *Wireless Access Point* berbasis Raspberry Pi dan 2 *client* dengan jarak 0 meter.

2) *Skenario Pengujian 2 (Pengujian dilakukan dengan jarak 4 meter)*

Pengujian berdasarkan jarak 4 meter tanpa halangan diantara *Wireless Access Point* berbasis Raspberry Pi dengan 2 *client* ketika melakukan sistem pengujian *online meeting* dan *video streaming*. Jadi yang diuji adalah proses pengiriman data dari *Wireless Access Point* berbasis Raspberry Pi menuju setiap *client*, sehingga dapat diketahui *Quality of Service* dari proses pengiriman data tersebut.

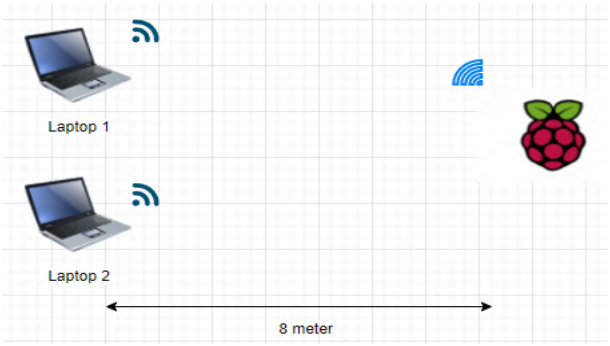


Gambar 5. Jarak antar perangkat *WAP* dan *client* 4 meter

Pada Gambar 5 merupakan tampilan topologi antar Wireless *Access Point* berbasis Raspberry Pi dan 2 *client* dengan jarak 4 meter.

3) *Skenario Pengujian 3 (Pengujian dilakukan dengan jarak 8 meter)*

Pengujian berdasarkan jarak 8 meter tanpa penghalang diantara *Wireless Access Point* berbasis Raspberry Pi dengan 2 *client* ketika melakukan sistem pengujian *online meeting* dan *video streaming*. Jadi yang diuji adalah proses pengiriman data dari *Wireless Access Point* berbasis Raspberry Pi menuju setiap *client*, sehingga dapat diketahui *Quality of Service* dari proses pengiriman data tersebut.

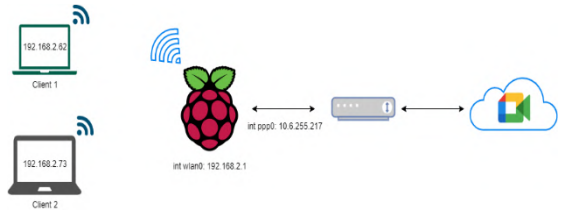


Gambar 6. Jarak antar perangkat *WAP* dan *client* 8 meter

Pada Gambar 6 merupakan tampilan topologi antar *Wireless Access Point* berbasis Raspberry Pi dan 2 *client* dengan jarak 8 meter.

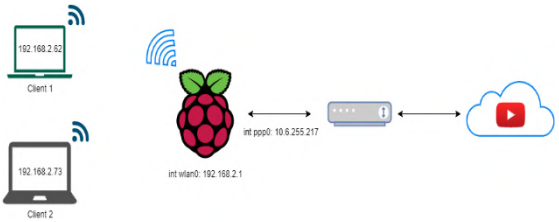
F. Pengujian Sistem Implementasi

Skenario pengujian pada penelitian ini, diawali dengan membangun WLAN berdasarkan topologi yang telah dibuat. Skenario pengujian menggunakan Raspberry Pi sebagai *Wireless Access Point* dengan sistem operasi Raspbian Buster. *Client* (laptop 1 dan laptop 2) sebagai alat melakukan sistem pengujian *online meeting* dan *video streaming* berdasarkan skenario pengujian pada jarak 0 meter, 4 meter dan 8 meter antara *Wireless Access Point* dengan *client*.



Gambar 7. Pengujian *Online Meeting* Google Meet

Sistem pengujian dilaksanakan berdasarkan percobaan *online meeting* menggunakan Google Meet oleh antar pengguna dalam rentang waktu sekitar 2 menit, adapun dapat diketahui *Quality of Service (QoS)* proses pengiriman datanya dengan memperhatikan protokol TCP dan TSL.



Gambar 8. Pengujian *Video Streaming* Youtube

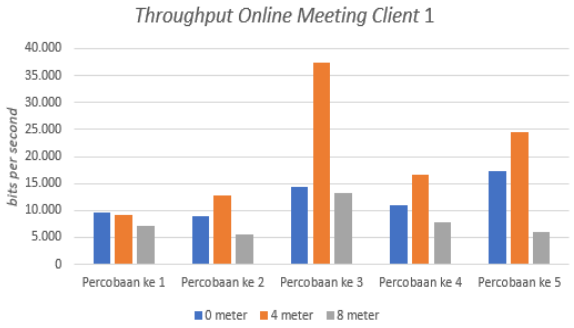
Sistem pengujian dilaksanakan berdasarkan percobaan *video streaming* Youtube oleh pengguna dengan resolusi video 360p dan 720p dalam rentang waktu sekitar 2 menit, sehingga dapat diketahui *Quality of Service (QoS)* proses pengiriman datanya dengan memperhatikan protokol TCP dan TSL.

IV. ANALISIS DAN PEMBAHASAN

A. *Skenario 1*

Tabel 1. Hasil *throughput online meeting client* 1 dari skenario pengujian

User	**Skenario**	***Throughput* (*bits per second*)**				
		Percobaan ke 1	**Percobaan ke 2**	**Percobaan ke 3**	**Percobaan ke 4**	**Percobaan ke 5**
Pc 1	1	9.705	9.034	14.426	11.030	17.336
	2	9.235	12.728	37.304	16.504	24.448
	3	7.031	5.558	13.136	7.695	6.026

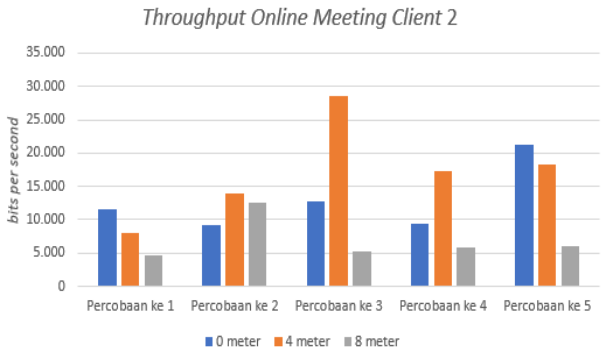


Gambar 9. Grafik *Throughput Online Meeting Client* 1 dari Skenario Pengujian

Pada Gambar 9 merupakan grafik *throughput online meeting* client 1 dari skenario pengujian yang menampilkan hasil dari percobaan ke 1 sampai percobaan ke 5 dari setiap skenario pengujian 0 meter, 4 meter dan 8 meter. Pada grafik menunjukan nilai *throughput* keseluruhan skenario pengujian yang paling rendah di percobaan ke 1 dari percobaan lainya. Kemudian pada skenario pengujian 8 meter nilai *throughput* cenderung lebih rendah dari skenario pengujian 0 meter dan 4 meter di setiap percobaan dikarenakan jarak sangat berpengaruh dalam proses transmisi data.

Tabel 2. Hasil *throughput online meeting client* 2 dari skenario pengujian

User	**Skenario**	***Throughput (bits per second)***				
		Percobaan ke 1	**Percobaan ke 2**	**Percobaan ke 3**	**Percobaan ke 4**	**Percobaan ke 5**
Pc 2	1	11.585	9.192	12.678	9.428	21.264
	2	7.930	13.848	28.608	17.344	18.256
	3	4.717	12.576	5.162	5.930	6.054



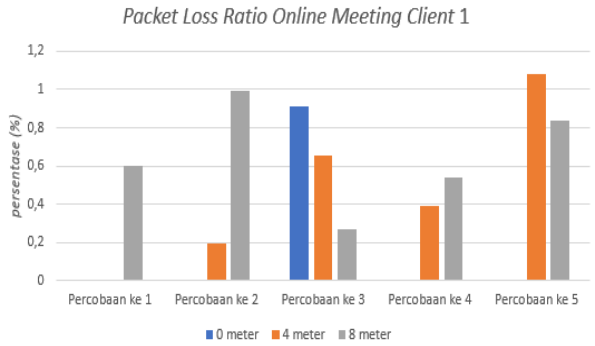
Gambar 10. Grafik *Throughput Online* Meeting *Client* 2 dari Skenario Pengujian

Pada Gambar 10 merupakan grafik *throughput online meeting client* 2 dari skenario pengujian yang menampilkan hasil dari percobaan ke 1 sampai percobaan ke 5 dari setiap skenario pengujian 0 meter, 4 meter dan 8 meter. Pada grafik menunjukan nilai *throughput* keseluruhan skenario pengujian yang paling rendah di percobaan ke 1 dari percobaan lainya.

Kemudian pada skenario pengujian 8 meter nilai *throughput* cenderung lebih rendah dari skenario pengujian 0 meter dan 4 meter di setiap percobaan dikarenakan jarak sangat berpengaruh dalam proses transmisi data.

Tabel 3. Hasil *packet loss ratio online meeting client* 1 dari skenario pengujian

User	**Skenario**	***Packet Loss Ratio* (%)**				
		Percobaan ke 1	**Percobaan ke 2**	**Percobaan ke 3**	**Percobaan ke 4**	**Percobaan ke 5**
Pc 1	1	0	0	0,909	0	0
	2	0	0,196	0,656	0,387	1,079
	3	0,598	0,995	0,271	0,536	0,839

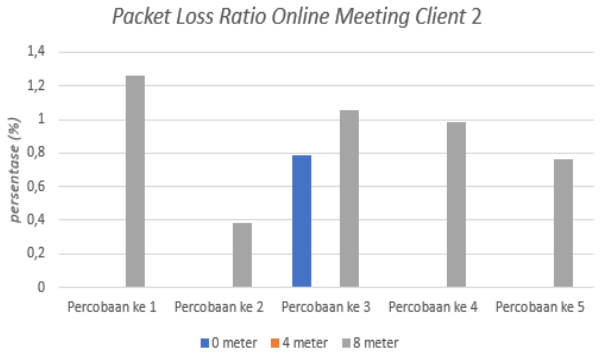


Gambar 11. Grafik *Packet Loss Ratio Online Meeting Client* 1 dari Skenario Pengujian

Pada Gambar 11 merupakan grafik *packet loss ratio online meeting client* 1 dari skenario pengujian yang menampilkan hasil dari percobaan ke 1 sampai percobaan ke 5 dari setiap skenario pengujian 0 meter, 4 meter dan 8 meter. Pada grafik menunjukan nilai *packet loss ratio* keseluruhan skenario pengujian yang paling tinggi pada percobaan ke 3 dari percobaan lainya. Kemudian pada skenario pengujian 8 meter nilai *packet loss ratio* lebih besar dibanding skenario pengujian 0 meter dan 4 meter di setiap percobaan karena dapat dipengaruhi beberapa hal seperti jarak, halangan serta interferensi ketika proses transmisi data berlangsung.

Tabel 4. Hasil *packet loss ratio online meeting client* 2 dari skenario pengujian

User	**Skenario**	***Packet Loss Ratio* (%)**				
		Percobaan ke 1	**Percobaan ke 2**	**Percobaan ke 3**	**Percobaan ke 4**	**Percobaan ke 5**
Pc 2	1	0	0	0,786	0	0
	2	0	0	0	0	0
	3	1,261	0,385	1,059	0,982	0,764

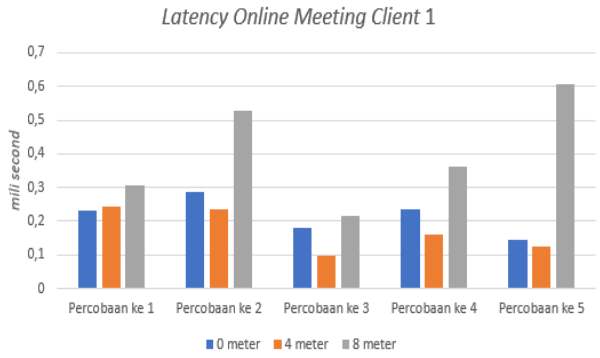


Gambar 12. Grafik *Packet Loss Ratio Online Meeting Client* 2 dari Skenario Pengujian

Pada Gambar 12 merupakan grafik *packet loss ratio online meeting client* 2 dari skenario pengujian yang menampilkan hasil dari percobaan ke 1 sampai percobaan ke 5 dari setiap skenario pengujian 0 meter, 4 meter dan 8 meter. Pada grafik menunjukan nilai *packet loss ratio* keseluruhan skenario pengujian yang paling tinggi di percobaan ke 3 dari percobaan lainya. Kemudian pada skenario pengujian 8 meter nilai *packet loss ratio* lebih besar dibanding skenario pengujian 0 meter dan 4 meter di setiap percobaan karena dapat dipengaruhi beberapa hal seperti jarak, halangan serta interferensi ketika proses transmisi data berlangsung.

Tabel 5. Hasil *latency online meeting client* 1 dari skenario pengujian

User	**Skenario**	***Latency* (*mili second*)**				
		Percobaan ke 1	**Percobaan ke 2**	**Percobaan ke 3**	**Percobaan ke 4**	**Percobaan ke 5**
Pc 1	1	0,231	0,286	0,179	0,234	0,145
	2	0,245	0,236	0,099	0,160	0,125
	3	0,307	0,527	0,214	0,360	0,606



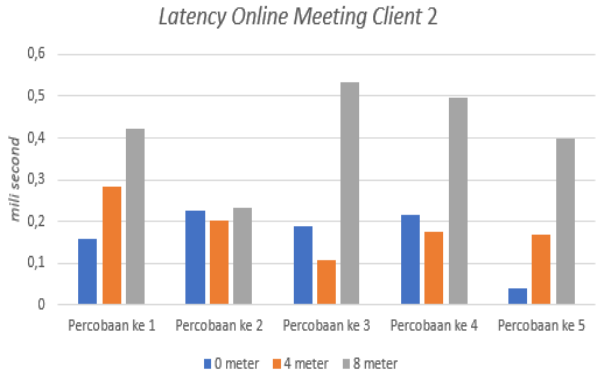
Gambar 13. Grafik *Latency Online Meeting Client* 1 dari Skenario Pengujian

Pada Gambar 13 merupakan grafik *latency online meeting client* 1 dari skenario pengujian yang menampilkan hasil dari percobaan ke 1 sampai percobaan ke 5 dari setiap skenario pengujian 0 meter, 4 meter dan 8 meter. Pada grafik menunjukan nilai *latency* keseluruhan skenario pengujian

yang paling tinggi di percobaan ke 2 dari percobaan lainya. Kemudian pada skenario pengujian 8 meter nilai *latency* cenderung lebih besar dari skenario pengujian 0 meter dan 4 meter di setiap percobaan karena semakin jauh jarak antara *Wireless Access Point* dan *client* semakin besar juga *latency* dalam pengiriman datanya.

Tabel 6. Hasil *latency online meeting client* 2 dari skenario pengujian

User	**Skenario**	***Latency* (*mili second*)**				
		Percobaan ke 1	**Percobaan ke 2**	**Percobaan ke 3**	**Percobaan ke 4**	**Percobaan ke 5**
Pc 2	1	0,157	0,226	0,188	0,217	0,040
	2	0,282	0,202	0,106	0,176	0,168
	3	0,423	0,232	0,534	0,498	0,399



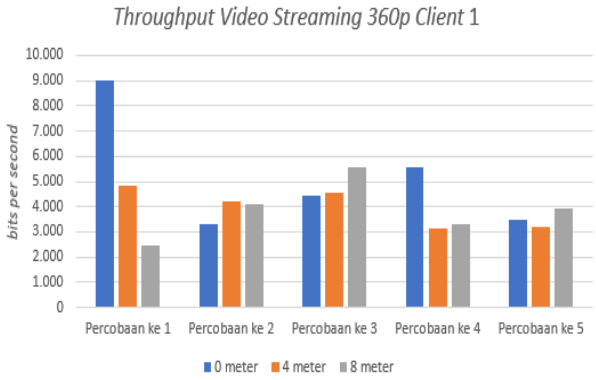
Gambar 14. Grafik *Latency Online Meeting Client* 2 dari Skenario Pengujian

Pada Gambar 14 merupakan grafik *latency online meeting client* 2 dari skenario pengujian yang menampilkan hasil dari percobaan ke 1 sampai percobaan ke 5 dari setiap skenario pengujian 0 meter, 4 meter dan 8 meter. Pada grafik menunjukan nilai *latency* keseluruhan skenario pengujian yang paling tinggi di percobaan ke 4 dari percobaan lainya. Kemudian pada skenario pengujian 8 meter nilai *latency* cenderung lebih besar dari skenario pengujian 0 meter dan 4 meter di setiap percobaan karena semakin jauh jarak antara *Wireless Access Point* dan *client* semakin besar juga *latency* dalam pengiriman datanya.

B. Tabel dan Grafik Sistem Pengujian Video Streaming 360p

Tabel 7. Hasil *throughput video streaming 360p client* 1 dari skenario pengujian

User	**Skenario**	***Throughput* (*bits per second*)**				
		Percobaan ke 1	**Percobaan ke 2**	**Percobaan ke 3**	**Percobaan ke 4**	**Percobaan ke 5**
Pc 1	1	9.026	3.302	4.446	5.557	3.471
	2	4.805	4.237	4.553	3.158	3.214
	3	2.476	4.117	5.545	3.303	3.931

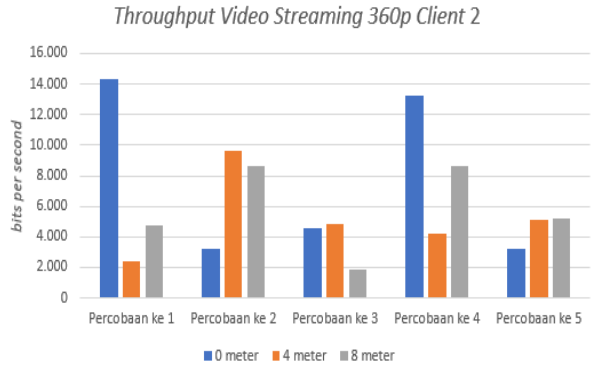


Gambar 15. Grafik *Throughput Video Streaming 360p Client* 1 dari Skenario Pengujian

Pada Gambar 15 merupakan grafik *throughput video streaming 360p client* 1 dari skenario pengujian yang menampilkan hasil dari percobaan ke 1 sampai percobaan ke 5 dari setiap skenario pengujian 0 meter, 4 meter dan 8 meter. Pada grafik menunjukan nilai *throughput* keseluruhan pada skenario pengujian yang paling rendah di percobaan ke 5 dari percobaan lainya. Kemudian pada skenario pengujian 4 meter dan 8 meter nilai *throughput* cenderung lebih rendah dari skenario pengujian 0 meter meter di setiap percobaan dikarenakan jarak sangat berpengaruh dalam proses transmisi data.

Tabel 8. Hasil *throughput video streaming 360p client* 2 dari skenario pengujian

User	**Skenario**	***Throughput* (*bits per second*)**				
		Percobaan ke 1	**Percobaan ke 2**	**Percobaan ke 3**	**Percobaan ke 4**	**Percobaan ke 5**
Pc 2	1	14.297	3.214	4.560	13.232	3.214
	2	2.429	9.654	4.846	4.222	5.112
	3	4.742	8.644	1.846	8.635	5.192



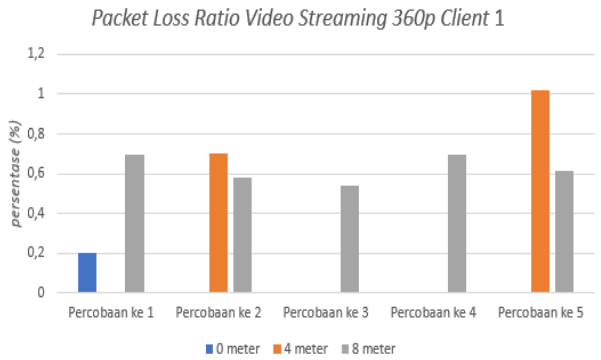
Gambar 16. Grafik *Throughput Video Streaming 360p Client* 2 dari Skenario Pengujian

Pada Gambar 16 merupakan grafik *throughput video streaming 360p client* 2 dari skenario pengujian yang

menampilkan hasil dari percobaan ke 1 sampai percobaan ke 5 dari setiap skenario pengujian 0 meter, 4 meter dan 8 meter. Pada grafik menunjukan nilai *throughput* keseluruhan pada skenario pengujian yang paling rendah di percobaan ke 3 dari percobaan lainya. Kemudian pada skenario pengujian 8 meter nilai *throughput* cenderung lebih rendah dari skenario pengujian 0 meter dan 4 meter di setiap percobaanya dikarenakan jarak sangat berpengaruh dalam proses transmisi data.

Tabel 9. Hasil *packet loss ratio video streaming 360p client* 1 dari skenario pengujian

User	**Skenario**	***Packet Loss Ratio* (%)**				
		Percobaan ke 1	**Percobaan ke 2**	**Percobaan ke 3**	**Percobaan ke 4**	**Percobaan ke 5**
Pc 1	1	0,199	0	0	0	0
	2	0	0,704	0	0	1,020
	3	0,698	0,579	0,538	0,694	0,617

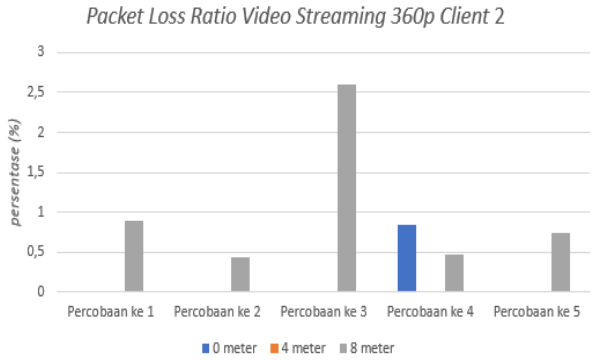


Gambar 17. Grafik *Packet Loss Ratio Video Streaming 360p Client* 1 dari Skenario Pengujian

Pada Gambar 17 merupakan grafik *packet loss ratio video streaming 360p client* 1 dari skenario pengujian yang menampilkan hasil dari percobaan ke 1 sampai percobaan ke 5 dari setiap skenario pengujian 0 meter, 4 meter dan 8 meter. Pada grafik menunjukan nilai *packet loss ratio* keseluruhan pada skenario pengujian yang paling tinggi di percobaan ke 5 dari percobaan lainya. Kemudian pada skenario pengujian 8 meter nilai *packet loss ratio* lebih besar dibanding skenario pengujian 0 meter dan 4 meter disetiap percobaan karena dapat dipengaruhi beberapa hal seperti jarak, halangan serta interferensi ketika proses transmisi data berlangsung.

Tabel 10. Hasil *packet loss ratio video streaming 360p client* 2 dari skenario pengujian

User	**Skenario**	***Packet Loss Ratio* (%)**				
		Percobaan ke 1	**Percobaan ke 2**	**Percobaan ke 3**	**Percobaan ke 4**	**Percobaan ke 5**
Pc 2	1	0	0	0	0,847	0
	2	0	0	0	0	0
	3	0,895	0,439	2,594	0,466	0,739

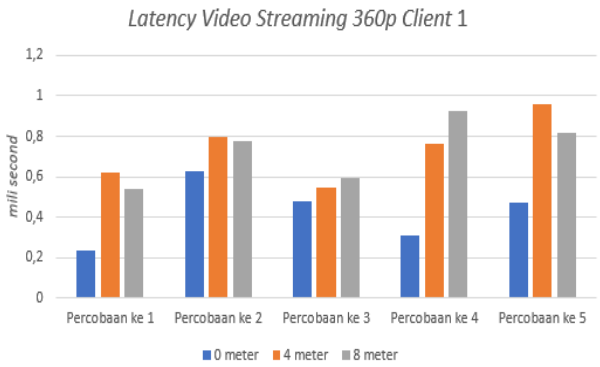


Gambar 18. Grafik *Packet Loss Ratio Video Streaming 360p Client* 2 dari Skenario Pengujian

Pada Gambar 18 merupakan grafik *packet loss ratio video streaming 360p client* 2 dari skenario pengujian yang menampilkan hasil dari percobaan ke 1 sampai percobaan ke 5 dari setiap skenario pengujian 0 meter, 4 meter dan 8 meter. Pada grafik menunjukan nilai *packet loss ratio* keseluruhan pada skenario pengujian yang paling tinggi di percobaan ke 4 dari percobaan lainya. Kemudian pada skenario pengujian 8 meter nilai *packet loss ratio* lebih besar dibanding skenario pengujian 0 meter dan 4 meter disetiap percobaan karena dapat dipengaruhi beberapa hal seperti jarak, halangan serta interferensi ketika proses transmisi data berlangsung.

Tabel 11. Hasil *latency video streaming 360p client* 1 dari skenario pengujian

User	**Skenario**	***Latency* (*mili second*)**				
		Percobaan ke 1	**Percobaan ke 2**	**Percobaan ke 3**	**Percobaan ke 4**	**Percobaan ke 5**
Pc 1	1	0,235	0,626	0,477	0,309	0,474
	2	0,623	0,796	0,545	0,764	0,958
	3	0,537	0,779	0,592	0,926	0,817



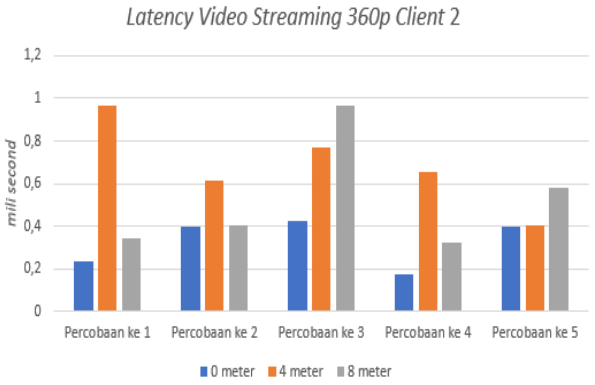
Gambar 19. Grafik *Latency Video Streaming 360p Client* 1 dari Skenario Pengujian

Pada Gambar 19 merupakan grafik *latency video streaming 360p client* 2 dari skenario pengujian yang menampilkan hasil dari percobaan ke 1 sampai percobaan ke 5 dari setiap skenario pengujian 0 meter, 4 meter dan 8 meter.

Pada grafik menunjukan nilai *latency* keseluruhan pada skenario pengujian yang paling tinggi di percobaan ke 5 dari percobaan lainya. Kemudian pada skenario pengujian 4 meter dan 8 meter nilai *latency* cenderung lebih besar dari skenario pengujian 0 meter di setiap percobaan karena semakin jauh jarak antara *Wireless Access Point* dan *client* semakin besar juga *latency* dalam pengiriman datanya.

Tabel 12. Hasil *latency video streaming 360p client* 2 dari skenario pengujian

User	**Skenario**	***Latency* (*mili second*)**				
		Percobaan ke 1	**Percobaan ke 2**	**Percobaan ke 3**	**Percobaan ke 4**	**Percobaan ke 5**
Pc 2	1	0,237	0,398	0,426	0,172	0,398
	2	0,967	0,617	0,772	0,651	0,405
	3	0,345	0,404	0,965	0,321	0,582



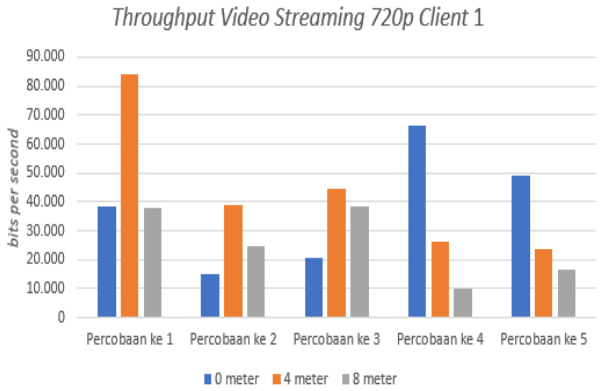
Gambar 20. Grafik *Latency Video Streaming 360p Client* 2 dari Skenario Pengujian

Pada Gambar 20 merupakan grafik *latency video streaming 360p client* 2 dari skenario pengujian yang menampilkan hasil dari percobaan ke 1 sampai percobaan ke 5 dari setiap skenario pengujian 0 meter, 4 meter dan 8 meter. Pada grafik menunjukan nilai *latency* keseluruhan pada skenario pengujian yang paling tinggi di percobaan ke 3 dari percobaan lainya. Kemudian pada skenario pengujian 4 dan 8 meter nilai *latency* cenderung lebih besar dari skenario pengujian 0 meter di setiap percobaan karena semakin jauh jarak antara *Wireless Access Point* dan *client* semakin besar juga *latency* dalam pengiriman datanya.

C. Tabel dan Grafik Sistem Pengujian *Video Streaming* 720p

Tabel 13. Hasil *throughput video streaming 720p client* 1 dari skenario pengujian

User	**Skenario**	***Throughput* (*bits per second*)**				
		Percobaan ke 1	**Percobaan ke 2**	**Percobaan ke 3**	**Percobaan ke 4**	**Percobaan ke 5**
Pc 1	1	38.208	14.988	20.602	66.364	48.891
	2	83.950	38.936	44.656	26.208	23.488
	3	37.675	24.504	38.568	10.008	16.784

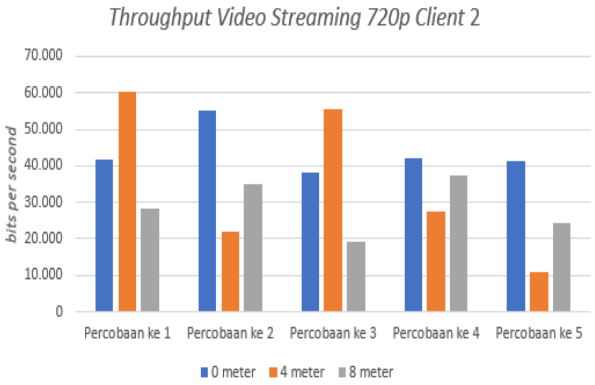


Gambar 21. Grafik *Throughput Video Streaming 720p Client* 1 dari Skenario Pengujian

Pada Gambar 21 merupakan grafik *throughput video streaming 720p client* 1 dari skenario pengujian yang menampilkan hasil dari percobaan ke 1 sampai percobaan ke 5 dari setiap skenario pengujian 0 meter, 4 meter dan 8 meter. Pada grafik menunjukan nilai *throughput* keseluruhan pada skenario pengujian yang paling rendah di percobaan ke 2 dari percobaan lainya. Kemudian pada skenario pengujian 8 meter nilai *throughput* cenderung lebih rendah dari skenario pengujian 0 meter dan 4 meter disetiap percobaan dikarenakan jarak sangat berpengaruh dalam proses transmisi data.

Tabel 14. Hasil *throughput video streaming 720p client* 2 dari skenario pengujian

User	**Skenario**	***Throughput* (*bits per second*)**				
		Percobaan ke 1	**Percobaan ke 2**	**Percobaan ke 3**	**Percobaan ke 4**	**Percobaan ke 5**
Pc 2	1	41.770	55.088	38.124	41.955	41.292
	2	60.232	22.016	55.632	27.320	10.744
	3	28.328	35.128	19.016	37.240	24.344

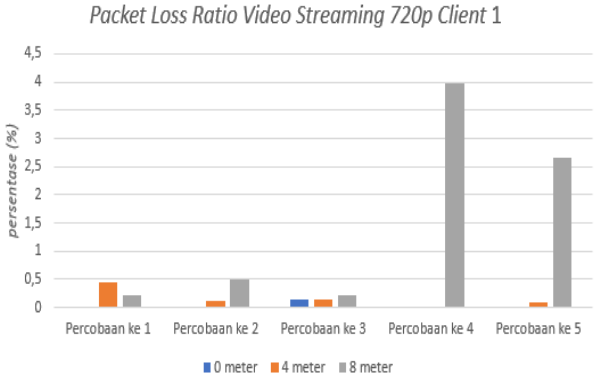


Gambar 22. Grafik *Throughput Video Streaming 720p Client* 2 dari Skenario Pengujian

Pada Gambar 22 merupakan grafik *throughput video streaming 720p client* 2 dari skenario pengujian yang menampilkan hasil dari percobaan ke 1 sampai percobaan ke 5 dari setiap skenario pengujian 0 meter, 4 meter dan 8 meter. Pada grafik menunjukan nilai *throughput* keseluruhan pada skenario pengujian yang paling rendah di percobaan ke 5 dari percobaan lainya. Kemudian pada skenario pengujian 8 meter nilai *throughput* cenderung lebih rendah dari skenario pengujian 0 meter dan 4 meter di setiap percobaan dikarenakan jarak sangat berpengaruh dalam proses transmisi data.

Tabel 15. Hasil *packet loss ratio video streaming 720p client* 1 dari skenario pengujian

User	**Skenario**	***Packet Loss Ratio* (%)**				
		Percobaan ke 1	**Percobaan ke 2**	**Percobaan ke 3**	**Percobaan ke 4**	**Percobaan ke 5**
Pc 1	1	0	0	0,139	0	0
	2	0,459	0,107	0,141	0	0,097
	3	0,226	0,498	0,221	3,987	2,657

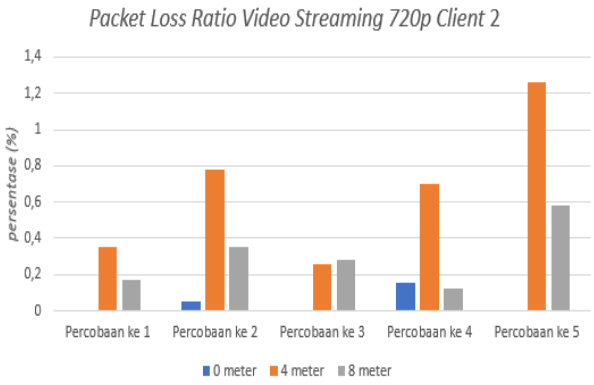


Gambar 23. Grafik *Packet Loss Ratio Video Streaming 720p Client* 1 dari Skenario Pengujian

Pada Gambar 23 merupakan grafik *packet loss ratio video streaming 720p client* 1 dari skenario pengujian yang menampilkan hasil dari percobaan ke 1 sampai percobaan ke 5 dari setiap skenario pengujian 0 meter, 4 meter dan 8 meter. Pada grafik menunjukan nilai *packet loss ratio* keseluruhan pada skenario pengujian yang paling tinggi di percobaan ke 4 dari percobaan lainya. Kemudian pada skenario pengujian 8 meter nilai *packet loss ratio* lebih besar dibanding skenario pengujian 0 meter dan 4 meter disetiap percobaan karena dapat dipengaruhi beberapa hal seperti jarak, halangan serta interferensi ketika proses transmisi data berlangsung.

Tabel 16. Hasil *packet loss ratio video streaming 720p client* 2 dari skenario pengujian

User	**Skenario**	***Packet Loss Ratio* (%)**				
		Percobaan ke 1	**Percobaan ke 2**	**Percobaan ke 3**	**Percobaan ke 4**	**Percobaan ke 5**
Pc 2	1	0	0,053	0	0,155	0
	2	0,355	0,776	0,261	0,702	1,265
	3	0,174	0,356	0,278	0,121	0,578

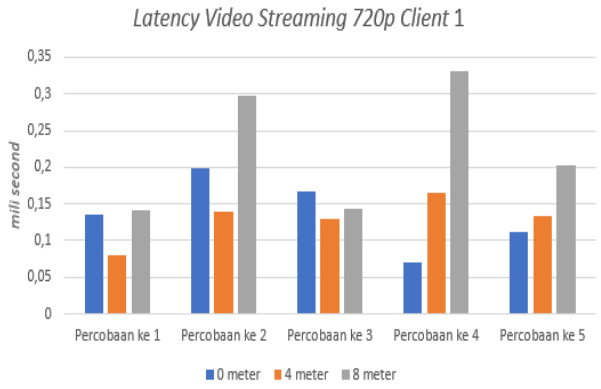


Gambar 24. Grafik *Packet Loss Ratio Video Streaming 720p Client* 2 dari Skenario Pengujian

Pada Gambar 24 merupakan grafik *packet loss ratio video streaming 720p client* 2 dari skenario pengujian yang menampilkan hasil dari percobaan ke 1 sampai percobaan ke 5 dari setiap skenario pengujian 0 meter, 4 meter dan 8 meter. Pada grafik menunjukan nilai *packet loss ratio* keseluruhan pada skenario pengujian yang paling tinggi di percobaan ke 5 dari percobaan lainya. Kemudian pada skenario pengujian 8 meter nilai *packet loss ratio* lebih besar dibanding skenario pengujian 0 meter dan 4 meter di setiap percobaan karena dapat dipengaruhi beberapa hal seperti jarak, halangan serta interferensi ketika proses transmisi data berlangsung.

Tabel 17. Hasil *latency video streaming 720p client* 1 dari skenario pengujian

User	**Skenario**	***Latency* (*mili second*)**				
		Percobaan ke 1	**Percobaan ke 2**	**Percobaan ke 3**	**Percobaan ke 4**	**Percobaan ke 5**
Pc 1	1	0,135	0,199	0,168	0,071	0,111
	2	0,080	0,139	0,129	0,165	0,133
	3	0,141	0,298	0,144	0,331	0,202

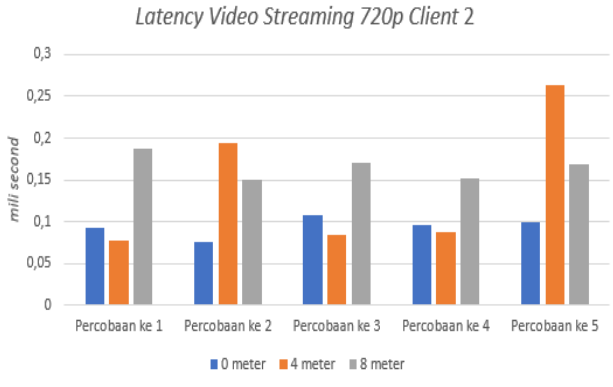


Gambar 25. Grafik *Latency Video Streaming 720p Client* 1 dari Skenario Pengujian

Pada Gambar 25 merupakan grafik *latency video streaming 720p client* 1 dari skenario pengujian yang menampilkan hasil dari percobaan ke 1 sampai percobaan ke 5 dari setiap skenario pengujian 0 meter, 4 meter dan 8 meter. Pada grafik menunjukan nilai *latency* keseluruhan pada skenario pengujian yang paling tinggi di percobaan ke 2 dari percobaan lainya. Kemudian pada skenario pengujian 8 meter nilai *latency* cenderung lebih besar dari skenario pengujian 0 meter dan 4 meter di setiap percobaan karena semakin jauh jarak antara *Wireless Access Point* dan *client* semakin besar juga *latency* dalam pengiriman datanya.

Tabel 18. Hasil *latency video streaming 720p client 2* dari skenario pengujian

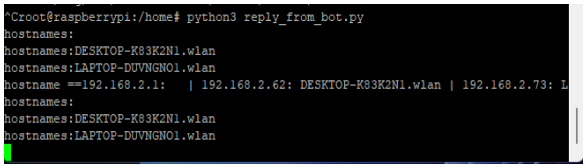
User	**Skenario**	***Latency* (*mili second*)**				
		Percobaan ke 1	**Percobaan ke 2**	**Percobaan ke 3**	**Percobaan ke 4**	**Percobaan ke 5**
Pc 2	1	0,092	0,075	0,108	0,095	0,099
	2	0,077	0,194	0,084	0,088	0,264
	3	0,187	0,150	0,171	0,151	0,169



Gambar 26. Grafik *Latency Video Streaming 720p Client* 2 dari Skenario Pengujian

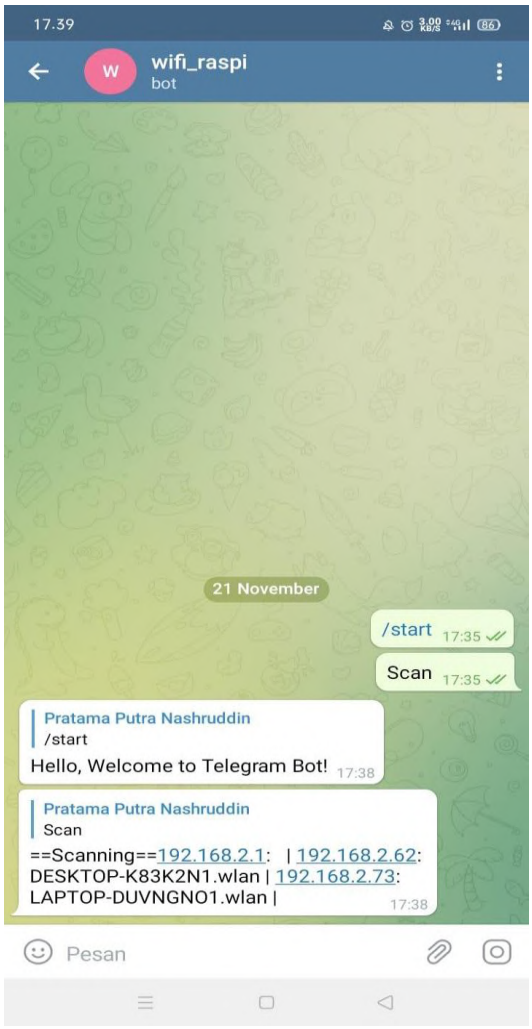
Pada Gambar 26 merupakan grafik *latency video streaming 720p client* 2 dari skenario pengujian yang menampilkan hasil dari percobaan ke 1 sampai percobaan ke 5 dari setiap skenario pengujian 0 meter, 4 meter dan 8 meter. Pada grafik menunjukan nilai *latency* keseluruhan pada skenario pengujian yang paling tinggi di percobaan ke 5 dari percobaan lainya. Kemudian pada skenario pengujian 8 meter nilai *latency* cenderung lebih besar dari skenario pengujian 0 meter dan 4 meter di setiap percobaan karena semakin jauh jarak antara *Wireless Access Point* dan *client* semakin besar juga *latency* dalam pengiriman datanya.

D. Pengiriman Informasi *Client* ke bot Telegram



Gambar 27. Proses *Scanning Client* pada Raspberry Pi

Pada Gambar 27. adalah proses perangkat Raspberry Pi untuk scanning pada jaringan lokal *interface wlan0* yang telah terintegrasikan pada bot telegram untuk mengirim informasi *client* pada jaringan tersebut berupa nama dan *ip address*.



Gambar 28. Hasil Informasi Perangkat *Client* di bot Telegram pada *Smartphone*

Pada Gambar 28 merupakan hasil respon bot telegram wifi_raspi kepada user Pratama Putra Nashruddin yaitu "Hello, Welcome to Telegram Bot!" untuk permintaan /start dan melakukan proses scanning informasi perangkat *client* yang terhubung dengan Raspberry Pi sebagai *Wireless Access Point* berupa nama dan *ip address* perangkat untuk permintaan "Scan".

V. KESIMPULAN

Berdasarkan hasil penelitian performa implementasi Wireless Access Point berbasis Raspberry Pi dan menggunakan USB modem sebagai sumber internet dapat diambil kesimpulan sebagai berikut :

- Hasil penggunaan *Wireless Access Point* berbasis Raspberry Pi menggunakan USB modem dapat diimplementasikan didaerah yang masih kekurangan akan infrastruktur jaringan serat optik.
- *Throughput* yang dihasilkan dari proses pengambilan data dipengaruhi oleh jarak dan halangan dari Raspberry Pi sebagai *WAP ke client. Throughput* yang diperoleh *client 1* dan *client 2* hampir serupa di setiap sistem pengujian dalam skenario pengujian dan pada skenario pengujian jarak 4 meter dan 8 meter nilai throughput cenderung lebih rendah dibanding dengan skenario pengujian 0 meter.
- *Packet loss ratio* yang dihasilkan dari proses pengambilan data dipengaruhi oleh jarak dan halangan dari Raspberry Pi sebagai *WAP* ke *client. Packet loss ratio* yang diperoleh *client 1* dan *client 2* pada skenario pengujian 1, skenario 2 dan skenario 3 mengalami peningkatan di setiap skenarionya karena jarak, halangan serta interferensi jaringan Raspberry Pi dan client sangat mempengaruhi nilai *packet loss ratio.* Nilai *packet loss ratio online meeting client 1* dan *client 2* pada skenario 1, skenario 2 dan skenario 3 didapatkan antara 0% sampai dengan 1,261%, kemudian nilai *packet loss ratio video streaming 360p client 1* dan *client 2* pada skenario 1,skenario 2 dan skenario 3 didapatkan antara 0% sampai dengan 2,594% dan nilai *packet loss ratio video streaming 720p client 1* dan *client 2* pada skenario 1, skenario 2 dan skenario 3 didapatkan antara 0% sampai dengan 3,987%. Sehingga mengacu pada standarisasi TIPHON mendapatkan hasil dalam kategori sangat bagus untuk pengujian *online meeting, video streaming kualitas 360p dan video streaming kualitas 720p* pada skenario 1, skenario 2 dan skenario 3.
- *Latency* yang dihasilkan dari proses pengambilan data dipengaruhi oleh jarak dan halangan dari Raspberry Pi sebagai *WAP* ke *client.* Latency yang diperoleh *client 1* dan *client 2* pada skenario pengujian 1, skenario pengujian 2 dan skenario pengujian 3 mengalami peningkatan di setiap skenarionya karena jarak dan halangan serta interferensi jaringan Raspberry Pi dan *client* sangat mempengaruhi nilai *latency.* Nilai *latency online meeting client 1* dan *client 2* pada skenario 1, skenario 2 dan skenario 3 didapatkan antara 0,040% sampai dengan 0,606%, kemudian nilai *latency video streaming 360p client 1* dan *client 2* pada skenario 1, skenario 2 dan skenario 3 didapatkan antara 0,235ms sampai dengan 0,967ms dan nilai *latency video streaming 720p client 1* dan *client 2* pada skenario 1, skenario 2 dan skenario 3 didapatkan antara 0,075ms sampai dengan 0,298ms. Sehingga didapatkan kualitas dalam kategori yang telah di standarisasi oleh TIPHON adalah sangat bagus untuk pengujian *online meeting, video streaming kualitas 360p* dan *video streaming kualitas 720p* pada skenario 1, skenario 2 dan skenario 3.
- Implementasi *Wireless Access Point* berbasis Raspberry Pi berhasil mengirim informasi perangkat yang terhubung berupa nama dan *ip address* menuju bot Telegram di *smartphone*.

REFERENSI

[1] A. Kadir, *Pengenalan Sistem Jaringan Komputer*, Yogyakarta: ANDI, 2003.

[2] B. D. Oetomo, *Konsep dan Dasar Perancangan Jaringan Komputer*, Yogyakarta: Andi, 2003.

[3] S. Rumalutur, "Analisis Keamanan Jaringan Wireless LAN Pada PT. PLN (Persero) Wilayah P2B Area Sorong," *Jurnal Teknologi Dan Rekayasa*, vol 19, no 3, 2014.

[4] A. Micro, *Dasar-dasar jaringan komputer*. Banjarbaru. ClearOsIndonesia. Edisi revisi 2012.

[5] D. Sharma, A. Gupta, A. K. Layek and S. Ghosh, "Movable Wireless Access Point for IoT-Based Home Automation," 2018 15th IEEE India Council International Conference (INDICON), 2018, pp. 1-6, doi: 10.1109/INDICON45594.2018.8987002.

[6] N. S. Ismail, N. A. Rashid, N. A. Zakaria, Z. I. Khan and A. R. Mahmud, "Low Cost Extended Wireless Network Using Raspberry Pi 3B+," 2020 IEEE Symposium on Industrial Electronics & Applications (ISIEA), 2020, pp. 1-4, doi: 10.1109/ISIEA49364.2020.9188215.

[7] C. Huda, F. A. Bachtiar and A. A. Supianto, "Reporting Sleepy Driver into Channel Telegram via Telegram Bot," 2019 International Conference on Sustainable Information Engineering and Technology (SIET), 2019, pp. 251-256, doi: 10.1109/SIET48054.2019.8986000.

[8] S. Sandhya, S. Purkayastha, E. Joshua and A. Deep, "Assessment of website security by penetration testing using Wireshark," 2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS), 2017, pp. 1-4, doi: 10.1109/ICACCS.2017.8014711.

Rancangan Sistem Otomatisasi Packet Filtering berdasar Sinkronasi Data pada IP Profile Database menggunakan Python

Tri Multy Rizkilina[1], Nur Rohman Rosyid[1,*]
[1]Departemen Teknik Elektro dan Informatika, Sekolah Vokasi, Universitas Gadjah Mada;
trimulty98@mail.ugm.ac.id
[*]Korespondensi: nrohmanr@ugm.ac.id;

Abstract – Data traffic on a dense network is a threat to cybercrime and a high vulnerability for technology companies so that the challenges to prevent it will be more diverse. Adding a strengthening of the boundary wall or firewall and sorting data through packet filtering plus writing firewall rules to prevent malware and attacks from outside at the network device level is an alternative to protect the traffic you have. When heavy traffic makes data exchange uncontrollable, this research will create an automation design so that the sorting of incoming data packets through selection based on the specified rules runs in real-time so that the prevention of crime that enters the network is more swiftly handled. The system is running successfully by connecting the MQTT Collector as a subscriber that uses the python programming language to retrieve profiling data from the IP Profile database. The system was tested on a mikrotik router RB951Ui-2HnD which then the blocking track record will be stored in the Dynamic Firewall Data database in MongoDB. Also added a tool for controlling data in storage in the form of a program release. The results of the test show that data with an average base score above 20 is blocked and then stored in the block list in checking the collection in the database every 30 seconds. In addition, the data in the database will be checked every day within 30 days which will then be released and recorded in the release log in MongoDB.

Keywords – Packet Filltering, Firewall Rules, Python, Router Mikrotik

Intisari – Lalu lintas data pada jaringan yang padat menjadi ancaman kejahatan dunia maya serta kerentanan yang tinggi bagi perusahaan teknologi sehingga tantangan untuk mencegahnya akan semakin beragam. Menambahkan penguatan dinding batas atau firewall serta memilah data melalui packet filtering ditambah menuliskan firewall rules untuk mencegah malware dan serangan dari luar di tingkat perangkat jaringan menjadi salah satu alternatif melindungi lalu lintas yang dimiliki. Saat traffic padat membuat tidak terkontrolnya pertukaran data maka penelitian ini akan membuat suatu rancangan otomatisasi agar pemilahan packet data yang masuk melalui seleksi berdasar rules yang ditentukan berjalan real-time sehingga pencegahan kejahatan yang masuk dalam jaringan lebih sigap ditangani. Sistem berhasil berjalan dengan menghubungkan MQTT Collector selaku subscriber yang memanfaatkan bahasa pemrograman python untuk mengambil data hasil profiling dari IP Profile database. Sistem diujicobakan pada router mikrotik RB951Ui-2HnD yang kemudian rekam jejak pemblokiran akan disimpan pada database di MongoDB. Hasil dari pengujian menunjukkan data dengan average base score diatas 20 terblokir kemudian disimpan pada block list pada pengecekan collection di database setiap 30 detik. Selain itu, data pada database akan dicek setiap hari dalam kurun waktu 30 hari yang kemudian akan di release dan tercatat pada log release di MongoDB.

Kata kunci – Packet Filltering, Firewall Rules, Python, Router Mikrotik

I. PENDAHULUAN

Pesatnya perkembangan teknologi membuat lalu lintas data pada jaringan semakin padat yang menyebabkan pertukaran informasi rawan akan serangan. Ancaman kejahatan dunia maya adalah kenyataan sehari-hari bagi perusahaan dan membuat tantangan menjadi dinamis dan berkelanjutan. Celah dari kerentanan dapat dimanfaatkan oleh orang-orang yang tidak bertanggung jawab untuk melakukan tindakan ilegal [1]. Dampak buruk bila pernah atau terus terkena serangan dari pihak yang tidak bertanggungjawab tanpa adanya langkah tepat untuk mencegahnya adalah pengaksesan ilegal sehingga terjadi kebocoran data yang menyebabkan data rahasia terekspos, terjadinya kelumpuhan sistem hingga kerusakan fatal [2]. Salah satu sistem yang dapat membantu dalam pecegahan serangan jahat pada lalu lintas jaringan adalah *firewall*. *Firewall* mampu memonitor dan mengontrol semua kegiatan yang terjadi pada lalu lintas jaringan yang masuk dan keluar berdasar peraturan keamanan yang ditetapkan.

Sistem keamanan jaringan komputer dengan metode *port blocking* menggunakan *router* mikrotik mampu menghasilkan kestabilan jaringan serta meminimalkan risiko masuknya *malware* dan serangan dari luar yang dapat memicu terjadinya kelumpuhan jaringan lokal [3]. Sistem yang dibuat cenderung manual sehingga antisipasi serangan jahat tidak diperbaharui secara sigap. Penerapan sistem *packet filtering* melalui konfigurasi akan mendapatkan akses untuk mendeteksi data mencurigakan berdasar alamat IP karena masuk berkali-kali. Konfigurasi akan diterapkan pada *router* mikrotik yang terhubung pada program otomatisasi dengan bahasa *python* menggunakan *library paramiko*. Data serangan akan di proses dengan menuliskan *firewall rules* yang diatur secara dinamis sehingga data-data dari *IP Profile database* yang sesuai dengan ketentuan dapat terus diperbaharui.

Melalui penelitian ini, tujuan sistem otomatisasi yang dijalankan pada *router* mikrotik berdasar sinkronasi data dari *IP Profile database* mampu melakukan pemblokiran otomatis terhadap data *malware* tanpa harus di konfigurasi manual. Pengkonfigurasian manual akan mengabiskan waktu, tenaga dan juga kurang tepatnya data-data serangan yang tercatat. Otomatisasi *packet filltering* pada penelitian ini akan sangat mempermudah administrator jaringan karena pengambilan data dari *IP Profile database* akan dilakukan secara rel-time untuk pengecekan database sehingga pembaruan data terkontrol, kemudian riwayat pemblokiran serta pelepasan

data yang sudah baik akan disimpan rapi di dalam *log collection* pada MongoDB.

II. TINJAUAN PUSTAKA

A. Keamanan Jaringan

Keamanan jaringan yaitu bagaimana suatu jaringan mampu mengamankan jaringannya, dalam penerapan keamanan tersebut perlu dibuatkan kebijakan teknis yang digunakan untuk mengelola *user*, mencegah akses yang tidak perlu yang nantinya dapat membebani jaringan [4]. *Traffic* jaringan yang ramai akan menimbulkan bermacam celah kejahatan seperti pencurian data atau peretasan yang mampu melumpuhkan sumber daya jaringan. Langkah untuk meminimalisi terjadinya penyalahgunaan jaringan dengan meningkatkan keamanan menggunakan beragam cara seperti penguatan *firewall*, memperkuat *password* dan memasang *anti virus* ataupun cara lainnya.

Salah satu penguatan keamanan *network traffic* dalam mengatisipasi terjadinya penyalahgunaan, menggunakan *filtering rule* dengan menerapkan metode *filtering rule* yang mampu melakukan *block url* yang ada pada *protocol* HTTP maupun HTTPS dengan hasil cukup baik. Hasil analisa yang didapatkan melalui simulasi menggunakan *tool network packet anayzer wireshark* menunjukkan setiap paket yang dikirim tidak dapat dibaca (blokir) baik pada *protocol* http maupun https [5].

B. *Firewall Packet Filtering*

Firewall jaringan adalah sistem yang membatasi akses dari dalam maupun keluar jaringan. Biasanya diposisikan diantara jaringan pribadi tepercaya dan terlindungi dan jaringan publik yang tidak terpercaya. *Firewall* hanya mengizinkan lalu lintas yang disetujui masuk dan keluar sesuai dengan kebijakan atau *rules firewall* yang dibuat [6].

Firewall umumnya terdiri dari bagian *filtering* yang berfungsi untuk membatasi akses untuk mengurangi fungsi jaringan, mempersempit kanal, atau untuk memblok kelas trafik tertentu dan bagian *gateway* (*gate*). Cara untuk mengoptimalisasi suatu *firewall* dengan menentukan kebijakannya atau *rules* seperti membatasi apa dan siapa saja yang perlu dilayani, layanan-layanan apa yang dibutuhkan oleh tiap pengguna jaringan serta menerapkan konfigurasi menggunakan beberapa parameter yang tercantum dalam *header* paket data: arah (*inbound atau outbound*), *address* asal dan tujuan, *port* asal dan tujuan, serta jenis *protocol transport* yang tepat sehingga kebijakan dapat diterapkan dengan baik dan keamanan jauh lebih baik dari ancaman yang ada [7].

C. *MongoDatabase*

Basis data relasional tidak digunakan dalam manajemen data NoSQL karena kerangka kerja ini menggunakan struktur penyimpanan data non-konvensional. Biasanya, sistem non-konvensional ini tidak mendukung operasi gabungan dan cara eksekusi kuerinya secara produktif. MongoDB adalah *database* NoSQL berbasis dokumen yang dibuat oleh MongoDB Inc, yang tersedia sebagai *open source*. Sistem yang digunakan oleh MongoDB ini merupakan sistem basis data berbasis dokumen yang digambarkan oleh penyimpanan data yang sangat besar, pada saat yang sama dengan kinerja kueri yang tinggi dan lebih baik [8].

IP Profile Database memanfaatkan layanan yang disediakan MongoDB untuk menyimpan data hasil *profiling*. Data yang tersimpan berupa hasil *generate* IOC *(Indicators of Compromise*) yang didapat dari MISP yang telah diolah berdasar perhitungan tingkat kejahatan. Memanfaatkan hasil profiling pada *IP Profile Database* akan mempermudah pengolahan data pemblokiran berdasar *average base score* yang di tentukan.

D. Otomasi Jaringan

Otomatisasi jaringan adalah proses berkelanjutan untuk menghasilkan dan menerapkan perubahan konfigurasi, manajemen, dan pengoperasian ditingkat perangkat jaringan [9]. Memanfaatkan otomatisasi akan mampu mempercepat waktu operasional, menghemat biaya serta menekan *human error* yang sering terjadi. Otomatisasi jaringan yang diimplementasikan membutuhkan kombinasi *hardware* dan *software* yang diatur agar dapat menjalankan tugas secara otomatis dan berulang dalam jaringan yang ada.

Menggunakan bahasa *scripting* yang secara luas digunakan oleh *administrator* jaringan dan *administrator* sistem untuk mengotomatisasikan tugas atau pekerjaan. Bahasa pemrograman yang biasa digunakan oleh administrator untuk proses otomatisasi adalah bahasa *phyton.* Selain mudah diterapkan, bahasa *python* memiliki beragam *library* atau pustaka yang mendukung otomatisasi pada jaringan [10].

E. *Python*

Pengembang pertama bahasa pemrograman *python* adalah Guido van Rossum pada tahun 1990 di CWI, Amsterdam. Versi terakhir yang dikeluarkan CWI adalah 1.2. *Python* dapat diperoleh dan dipergunakan secara bebas karena lisensi *Python* tidak bertentangan baik menurut definisi *Open Source* maupun *General Public License* (GPL). Hingga saat ini pengembangan *Python* terus dilakukan oleh kumpulan pemrogram secara terpusat dikoordinir oleh Guido dan *Python Software Foundation*. *Python Software Foundation. Python* dapat digunakan untuk berbagai keperluan pengembangan perangkat lunak dan dapat berjalan diberbagai macam sistem operasi karena sifatnya yang *multiplatform* sehingga mencegah *Python* dimiliki oleh perusahaan komersial [11].

Python adalah bahasa pemrograman yang kuat dan mudah dipelajari. *Python* memiliki struktur data tingkat tinggi yang efisien dan pendekatan yang sederhana namun efektif untuk pemrograman berorientasi objek. Sintaks *Python* yang elegan dan pengetikan dinamis, bersama dengan sifat interpretasinya, menjadikannya bahasa yang ideal untuk pembuatan skrip dan pengembangan aplikasi yang cepat di banyak area pada sebagian besar *platform* [12].

Penelitian mengenai otomatisasi menggunakan bahasa *python* menjadi pengantar ide yang akan dikembangkan banyak pihak. Penelitian tentang metode ringkas dalam mengkonfigurasi perangkat jaringan atau disebut otomatisasi akan mampu mengurangi waktu untuk konfigurasi peralatan dan memudahkan perawatan. Memanfaatkan *library* Netmiko dan Paramiko, infrastruktur skrip yang tepat mampu mewujudkan sistem baik disertai tingkat kerentanan yang minim [13].

F. Paramiko

Paramiko adalah pustaka yang mengimplementasikan protokol SSH, biasanya digunakan untuk mengelola sistem UNIX dari jarak jauh. Paramiko membungkus protokol dan memungkinkan abstraksi tingkat tinggi dan tingkat rendah [14].

Saat paramiko telah berhasil berjalan, maka proses kerjanya SSH *client* sebagai objek akan meminta *bind* pada *local port* sehingga ssh *server* dan ssh *client* saling terkoneksi. *Client* meminta *public key* dan *host key* milik *server*. *Server* bertanggung jawab untuk membatasi pengguna dengan mengizinkan data apa saja yang boleh digunakan seperti kata sandi, dan atau jenis saluran. *Client* dan *server* menyetujui algoritma enkripsi yang akan dipakai. Setelah itu *Client* akan membentuk *session key* yang didapat sesuai persetujuan keduanya dan dienkripsikan menggunakan public *key* milik *server*. *Server* akan men-decrypt *session key* yang didapat dari *client* lalu mengenkripsi ulang dengan public *key* milik *client*, dan mengirimkannya kembali ke *client* untuk verifikasi. Langkah terakhir user akan mengautentifikasi ke *server* melalui *session key* yang tersedia untuk melakukan pengaksesan data [15].

G. MQTT

MQTT adalah *publish* atau *subscribe protocol* berbasis topik yang menggunakan string karakter untuk mendukung topik hierarki. Protokol ini memfasilitasi langganan ke berbagai topik. Protokol MQTT bersifat terbuka yang dirancang untuk perangkat terbatas yang digunakan dalam aplikasi telemetri. MQTT tidak menentukan teknik perutean atau jaringan apa pun yang berarti diasumsikan bahwa jaringan yang mendasari menyediakan layanan transportasi data *point-to-point*, *session-oriented*, *auto-segmenting* dengan *in-order delivery* (seperti TCP / IP) dan menggunakan layanan ini untuk pertukaran pesan [16].

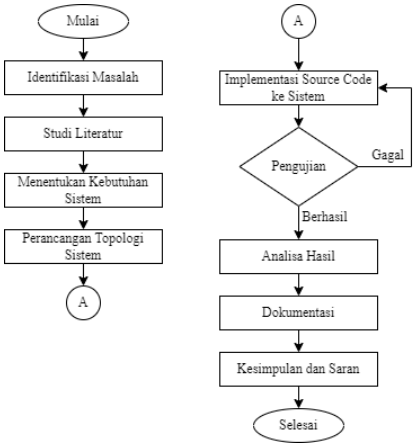
Implementasi Protokol MQTT untuk Sistem *Monitoring* Perangkat IoT. Penelitian menguji suatu sistem *monitoring* IoT memanfaatkan protokol MQTT yaitu *publish-subscribe* sangat berpengaruh dalam berjalannya proses transfer data yang dibutuhkan. Pada MQTT terdapat *library* paho yang dimanfaatkan *client* untuk men *subscribe* MQTT *broker* sebagai sumber data. Keunggulan lainnya dari MQTT salah satunya kebutuhan *resource* pada protokol MQTT lebih sedikit dibanding protokol UDP yang membuat sistem mampu berjalan pada keadaan *bandwith* yang rendah dan *latency* yang tinggi [17].

III. METODE PENELITIAN

Bab ini akan menjabarkan mengenai metode penelitian yang digunakan dalam penelitian Rancangan Sistem Otomatisasi *Packet Filtering* Berdasar Sinkronasi Data pada *IP Profile Database* Menggunakan *Python*.

A. Tahap Penelitian

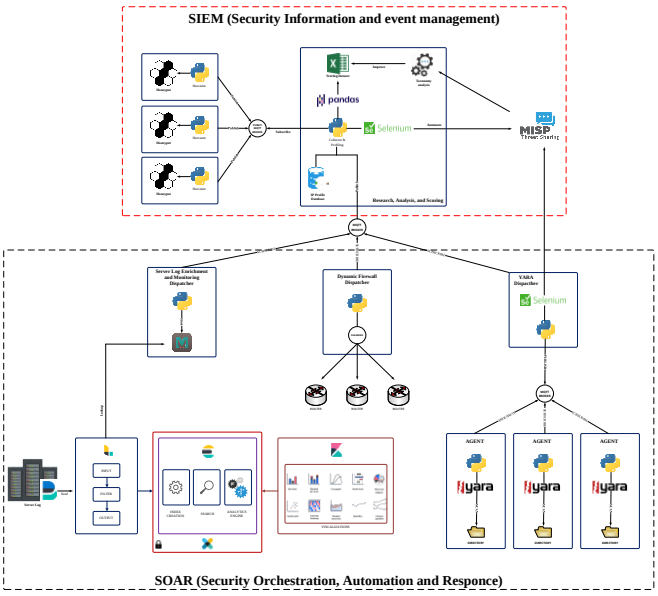
Penelitian ini melalui proses dan tahapan yang digambarkan pada Bagan Alir pada Gambar 1.



Gambar 1. Alur Penelitian

B. Arsitektur Sistem Keseluruhan

Penelitian *Threat Intelegence System* ini terdiri dari beberapa topik pembahasan yang saling terkait. Pada Gambar 2 digambarkan arsitektur topik penelitian atau *framework* keamanan informasi yang akan diimplementasikan dan diuji yang mengimplementasikan konsep SIEM (*Security Information and Event Management*) dan SOAR (*Security Orchestration, Automation and Responce*).



Gambar 2. Arsitektur Keseluruhan Sistem

C. Alat dan Bahan

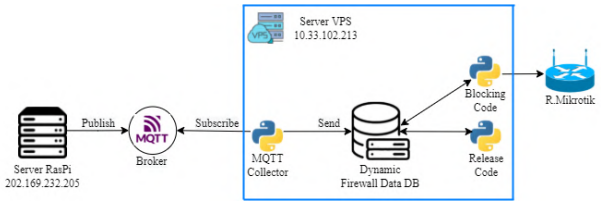
Penelitian ini menggunakan beberapa perangkat keras dan perangkat lunak sebagai penunjang proses simulasi penelitian yang dilakukan. Berikut peralatan yang digunakan

1) Perangkat Keras
- Satu (1) Komputer (Laptop)
- Satu (1) Router Mikrotik RB951Ui-2HnD

2) Perangkat Lunak
- Windows 10
- OpenVPN
- VPS UGM
- MongoDB Compass
- *Python*
- *Library* pymongo
- *Library* paramiko
- *Library* paho-mqtt *client*
- *Python* Idle

D. Arsitektur Sistem dan Fokus Penelitian

Proyek Akhir yang dikerjakan ini berfokus pada manajemen respon implementasi dari sistem SOAR untuk melakukan pengamanan gerbang antara jaringan lokal dengan jaringan internet. Sistem bernama *Dynamic Firewall* akan berjalan secara otomatis mengelola data yang akan di blokir. Arsitektur sistem pada *Dynamic Firewall* memiliki tahapan dalam berjalannya sistem yang dapat dilihat pada Gambar 3.



Gambar 3. Arsitektur Sistem Penelitian

Penjelasan pengaturan topologi dijabarkan menjadi 4 tahap.

1) *Setup* VPS dan Mikrotik RB951Ui-2HnD

Tahapan menjalankan sistem pada perangkat jaringan, diawali dengan installasi kebutuhan sistem pada VPS. *Virtual Private Server* yang digunakan menggunakan layanan VPS milik ugm sehingga jaringan yang terhubung bersama dengan router dalam jaringan ugm. Menginstall kebutuhan *library python* pada VPS seperti paramiko, *pymongo* dan paho MQTT *client*. Pada router mikrotik di atur konfigurasi IP serta memastikan router dapat berkomunikasi dengan VPS, selain itu dilakukan pengaturan penting yaitu mengatur *command firewall rules* sejak awal pengkonfigurasian. Setelah mengkonfigurasi router, tak lupa melakukan konfigurasi akses *Secure Shell* (SSH) protokol agar router bisa di *remote* atau di kontrol jarak jauh. Pengaturan SSH yang dilakukan ini agar router mikrotik dapat di otomatisasi oleh program *python* yang akan di hubungkan nanti. Jika pengaturan SSH berhasil maka router akan bisa di *remote* dan diakses melalui program dengan paramiko.

2) *Setup* MQTT *Collector*

Tahap pengambilan data selanjutnya dengan menjalankan *Collector* yang akan bertugas membangun koneksi dengan MQTT *Broker* yang terkoneksi dengan RasPi *Server* selaku *Publisher*. RasPi akan mengirimkan data berisi satu persatu file utuh berformat *json* yang telah melalui tahap *profiling*, fokus data yang akan dikumpulkan ada di parameter *average base score*. Melalui minimal nilai *average base score* yang ditentukan pada program *blocking code* akan menentukan data mana yang akan di blokir. Seluruh data yang masuk akan di simpan ke MongoDB secara terpusat.

3) *Setup Blocking Code*

Tahapan berikutnya, setelah data yang dibutuhkan dalam proses pemfilteran masuk ialah menjalankan perintah otomatisasi *packet filtering* atau *blocking code* menggunakan bahasa pemrograman *python*. Memanfaatkan *library* paramiko agar bisa melakukan kontrol jarak jauh pada *router* mikrotik yang digunakan, *library* ini masuk menggunakan protokol SSHv2 sebagai *server* maupun *client*. Selain itu, pengaturan perintah pada kode diinputkan *firewall rules* sesuai *command* yang dimiliki oleh *router* mikrotik. *Firewall Rules* tersebut akan berkorelasi dengan nilai *average base score* yang akhirnya terjadi proses pemblokiran. Saat program dijalankan perintah-perintah yang dituliskan akan otomatis tercatat didalam mongodb di *collection* blog_log_history.

4) *Setup Release Code*

Tahap keempat ini menjadi penentu kapan IP yang terblokir oleh sistem boleh dikeluarkan. Menggunakan bahasa permograman *python* yang memanfaatkan *library time* dan *datetime*, diatur agar waktu data masuk dalam *collection* blog_log_history dengan data yang akan di keluarkan berjarak sesuai dengan tenggat waktu yang ditentukan. Data yang berhasil keluar dari list pemblokiran akan di simpan ke MongoDB di *collection* log_release.

E. Penentuan Batas Nilai *Average Base Score*

Nilai *average base score* yang dimiliki oleh data *profiling* ditentukan berdasarkan pada jumlah kemunculan IoC yang sama pada Honeypot (penyerang melakukan *brute-force attack*), yang berarti bila semakin banyak IoC yang sama muncul dideteksi bahwa penyerang berusaha masuk terus menerus secara paksa. Perhitungan nilai *risk score* dilakukan oleh Arfan di penelitian "Perancangan Metode Profiling pada Honeypot Indicator of Compromise (IoC) berbasis Korelasi pada *Malware Information Sharing Platform* (MISP)" yang menghasilkan *average base score* yang akan digunakan sebagai acuan nilai batas pemblokiran berdasar Tabel 1 berikut.

Tabel 1. Risk Score Protokol Mysql

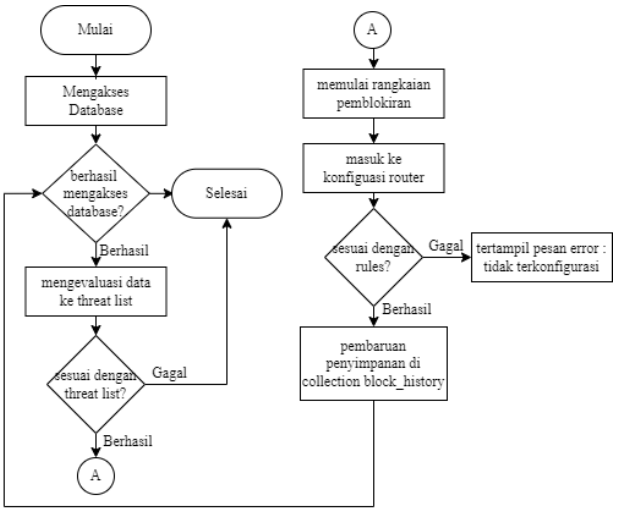
No	Protokol	Risk Score	Presentase
1	Mysql	15	25%
2	Mysql	15.45	22.06%
3	Mysql	17.25	17.65%
4	Mysql	21.3	16.18%
5	Mysql	100	19.12%

Average base score didapatkan dari hasil kali antara rerata *risk score* protocol mysql dengan banyaknya data serangan yang masuk. Data serangan masuk ke protokol *mysql* sebanyak 61.1% sehingga didapat batas nilai *average base score* sebagai acuan pemblokiran di kisaran nilai 20,6.

F. Cara Kerja Sistem

1) Skema Berjalannya Proses Pemblokiran

Diagram alir berjalannya *blocking code* dijabarkan pada Gambar 4. Penjelasan diagram alir dimulai dari pengambilan data *profiling* dari MongoDB di *collection* data_to_block, ketika data berhasil di dapat maka akan masuk ke proses seleksi kondisi, jika nilai *average base score* data sesuai ketentuan batas minimal maka akan diteruskan ke proses selanjutnya yaitu diteruskan ke *router* mikrotik melalui paramiko. Saat berhasil masuk ke *router* dan melewati proses pemblokiran maka alamat IP penyerang tidak punya hak akses di lalu lintas jaringan *router* dan masuk ke catatan pada MongoDB di *collection* block_log_history.



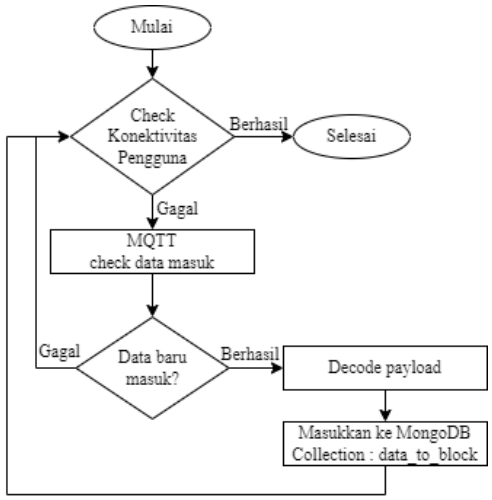
Gambar 4. Alur Sistem Pemblokiran

2) Cara Pengambilan Data

Mengimplementasikan kode yang saling terkait melalui tahapan berikut

a. Implementasi MQTT *Collector*

Program MQTT *Collector* menggunakan bahasa pemrograman *python* berfungsi untuk mengambil data yang dikirimkan oleh *server* RasPi melalui *protokol* MQTT *publisher*. Data akan disimpan pada *database* MongoDB mengunakan *library pymongo*. Program ini akan membangun koneksi untuk men-*subscribe* MQTT *Broker*. Proses berjalannya program *collector* saat menghimpun data dari *publisher* dijabarkan pada Gambar 5.



Gambar 5. Alur Proses Program MQTT-Collector

b. *Setting* Perangkat Jaringan

Memasukkan 2 *firewall rules* selaku *intial config* yang perlu disiapkan dari awal konfigurasi sebagai syarat program dapat berjalan semestinya. Berikut penjabaran *rules* yang di tuliskan

- *Rules* Satu

  Menuliskan rules ini kan membuat *address-list* yang sudah terdaftar dalam *drop-traffic* akan di *block*

  /ip firewall mangle add action=add-src-to-address-list address-list=drop_traffic chain=prerouting protocol=tcp

- *Rules* Dua

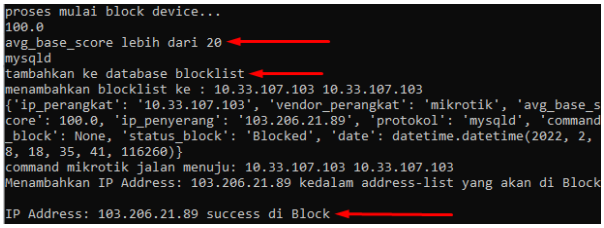
  Setelah berhasil masuk *drop-traffic* maka dipasang *firewall filter* ke *interface ether2* sehingga *traffic* yang masuk melalui *ether2* akan di *filter* sesuai daftar *address list*

  /ip firewall filter add action=drop chain=input src-address-list=drop_traffic comment=WORM in-interface=ether2

c. Implementasi *Blocking Code*

Saat *bloking code* berhasil dijalankan maka akan tertampil keterangan proses pemblokiran, letak dan keterangan alamat IP perangkat dimana data terblokir, serta data apa saja yang akan masuk pada tahap pemblokiran. Terminal tempat program berjalan pada Gambar 6 menampilkan keterangan hasil dari proses

pemblokiran. Keterangan bahwa alamat IP trekait terporses dalam *filtering* terblokir tertulis jelas.



Gambar 6. Terminal *Blocking-Code*

d. Implementasi *Release Code*

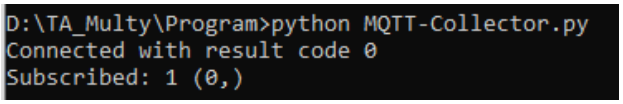
Menjalankan kode ini maka, pengecekan akan terus dilakukan pada data-data yang ada di *collection* block_log_history sehingga akan terus muncul keterangan data di jendela depan apakah data itu masih tersimpan atau bila sudah di *release* akan ditampilkan keterangan '*database* kosong'.

IV. HASIL DAN PEMBAHASAN

A. Hasil *Running* MQTT *Collector*

1) Terminal

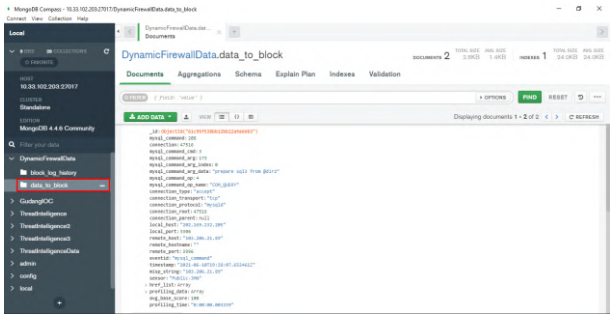
Saat MQTT *Collector* berhasil terhubung dengan *broker* akan menampilkan keterangan seperti pada Gambar 7. Saat data masuk, jendela MQTT-*Collector* akan menampilkan tiap-tiap file utuh json hasil *profiling* yang dikirm dari *IP Profile Database* secara berkala dan akan tercatat di *collection Dynamic Firewall Data database.*



Gambar 7. Hasil MQTT-*Collector* Terhubung *Broker*

2) MongoDatabase

Terbuat *collection* baru dengan nama data_to_block yang akan menyimpan masuknya data-data yang akan di proses pada *blocking code*. Data yang tersimpan disini tidak bersifat permanen untuk mencegah penumpukan data yang akan terus menerus bertambah. Tampilan data yang berhasil masuk dan tercatat pada *collection* terdapat pada Gambar 8.



Gambar 8. Tampilan Isi *Collection* data_to_block

B. Hasil *Input Firewall Rules*

Tampilan pada router ketika *firewall rules* berhasil di *input*kan seperti pada Gambar 9.

C. Hasil *Running Blocking Code*

1) MongoDatabase

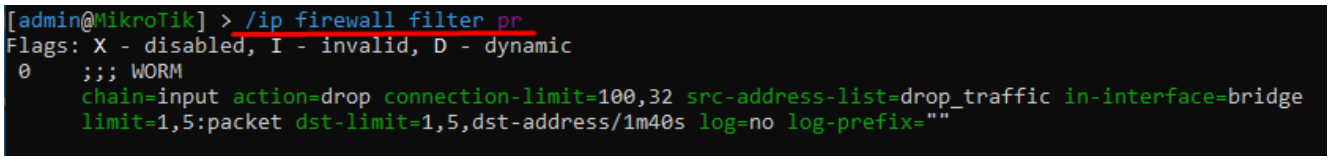
Data akan hilang dari *database* setelah berhasil masuk pada tahap pemblokiran di *router* sehingga isi dari *collection* data_to_block kosong. Data-data hasil pemblokiran tercatat dan tersimpan seperti pada tampilan Gambar 10 di block_log_history.

2) Terminal

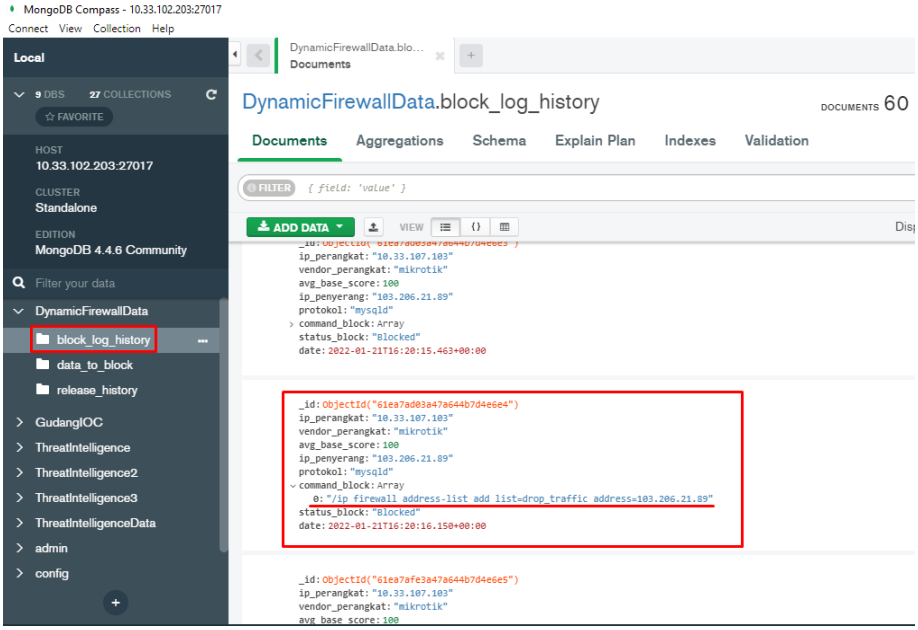
Rangkaian proses pemblokiran selesai dan berhasil, maka akan muncul keterangan pada terminal di Gambar 11 bahwa data penyerang masuk pada *log* sesuai format yang telah di atur pada *code*.

3) Router

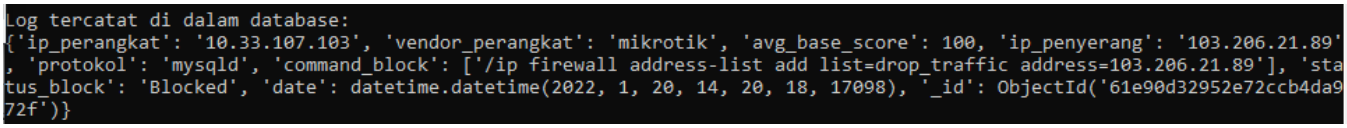
Saat *Blocking Code* berhasil masuk dan menjalankan perintah serta *rules* diterapkan. Maka untuk pengecekan apakah alamat IP penyerang masuk dalam catatan terblokir dituliskan perintah */ip firewall address print.* Ditampilan seperti pada Gambar 12.



Gambar 9. Hasil *Input Firewall Filter*

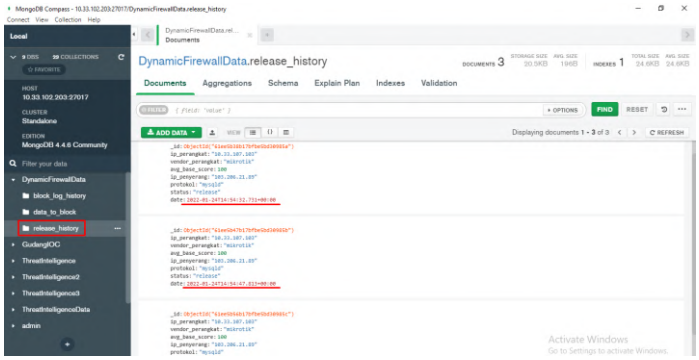


Gambar 10. Tampilan *Collection* block_log_history



Gambar 11. Tampilan *Log Format*

Gambar 12. Alamat IP Penyerang Tercatat



Gambar 13. Tampilan isi log_release

D. Hasil *Running Release Code*

1) MongoDatabase

Data-data hasil *blocking* di *collection* block_log_history yang siap di *release*, disororoti bagian waktu masukknya data akan menjadi parameter penting dalam proses pe-*release*-an. *Collection* log_release pada Gambar 13 akan berisi data hasil pengeluaran data dari block_log_history yang sudah tersimpan selama 30 hari atau lebih disana. tercatat tanggal keterangan data dikeluarkan

E. Hasil Pengujian

Program pemblokiran menyimpan data berdasar rerata *base score* lebih dari sama dengan 20. Pada pengamatan selama 30 menit, didapat data masuk pada *collection* data_to_block sebanyak 60 *document* yang langsung terhapus dan berpindah ke *collection* block_log_history saat *blocking code* dijalankan dengan jeda 45 detik karena memenuhi batas nilai rerata *base score*. Data yang berusia lebih dari dan sama dengan 30 hari berpindah pada *log release* saat program *release* dijalankan. Pada Tabel 2 terangkum hasil pengujian dari keempat tahap menjalankan sistem.

Tabel 2. Hasil Pengujian

No.	Proses	Pengujian	Hasil
1.	MQTT-*Collector* menghimpun data dengan men-*subscribe* topik "UGM_ProFeedAlert " kemudian menyimpannya di MongoDB.	Menjalankan program MQTT-*Collector* dan mengamati terminal program serta mongodb compass.	Berhasil
2.	Memastikan settingan ssh betul sehingga pramiko bisa mengontrol router.	Login SSH router mikrotik pada putty atau terminal.	Berhasil
3.	Data proofing yang tersimpan pada *collection* akan di seleksi average base score yang dimiliki kemudian masuk pada router untuk pemblokiran dan dicatat pada block history pada MongoDB.	Menjalankan program blocking code, mengamati terminal dan pembaruan data pada mongodb.	Berhasil
4.	Data akan dicek setiap hari selama 30 hari untuk siap di *release.*	Menjalankan program log *release* serta mengamati terminal dan pembaruan data pada *collection* log release.	Berhasil

V. KESIMPULAN DAN SARAN

A. Kesimpulan

Rancangan otomastisasi ini berhasil mengembangkan sistem pemblokiran alamat IP terduga sumber *malware* berdasar data *profiling*. Melalui program otomasi berbahasa python, pemblokiran data berjalan secara *real-time* melalui koneksi paramiko yang mengimplementasikan protokol ssh. Program otomatis mengakses router mikrotik dan mencatat alamat IP terduga, didapat efisiensi waktu dalam melakukan pengkonfigurasian. Pada pengamatan selama 30 menit, data masuk pada *database* sebanyak 60 document yang terdistribusi langsung saat batas rerata *base score* data terpenuhi ke log yang telah disediakan. Data pada *block list* akan dikontrol selama 30 hari dan setelahnya perbolehkan keluar sembari rekam jejaknya dicatat.

B. Saran

Beberapa saran yang dapat diterapkan dalam pengembangan penelitian berikutnya :

1) Melakukan uji fungsionalitas terhadap versi *router* mikrotik lainnya

2) Melakukan uji otomatisasi *packet filltering* pada *vendor* lainnya

3) Menambahkan *interface* yang menarik untuk tampilan hasil otomatisasi di terminal

REFERENSI

[1] S. Iovan and A. Iovan, "From Cyber Threats To Cyber-Crime". *Romanian Economic Business Review*, vol. 10(2), p. 425-434, 2016.

[2] M. Rizki, "Implementasi dan Analisis Perbandingan Performa IDS *Snort* dan IDS Suricata pada Raspberry Pi". Yogyakarta: Universitas Gadjah Mada, 2019.

[3] D. Irawan, "Keamanan Jaringan Komputer dengan Metode *Blocking Port* pada Laboratorium Komputer Program Diploma-III Sistem Informasi Universitas Muhammadiyah Metro". *Mikrotik: Jurnal Manajemen Informatika*, vol 5 (2), 2015.

[4] I.G.K.O. Mardiyana, "Keamanan Jaringan dengan *Firewall Filter* Berbasis Mikrotik pada Laboratorium Komputer STIKOM Bali". *Konferensi Nasional Sistem dan Informatika 2015, Denpasar, Indonesia. STMIK STIKOM Bali*.

[5] A. Muzakir and M. Ulfa, "Analisis Kinerja *Packet Filtering* Berbasis Mikrotik *Routerboard* Pada Sistem Keamanan Jaringan", *Jurnal SIMETRIS*, vol. 10 (1), 2015.

[6] T. Katic and P. Pale., "Optimization of Firewall Rules". 2007 29th *International Conference on Information Technology Interfaces*, p. 685-690, 2007

[7] F.A. Purwaningrum, E.A. Darmadi, A. Purwanto, "Optimalisasi Jaringan Menggunakan *Firewall". Jurnal IKRA-ITH Informatika*, vol 2 (3), 2018.

[8] B. Jose and S. Abraham, "Performance analysis of NoSQL and relational databases with MongoDB and MySQL". *Materials Today: Proceedings*, 24, p. 2036–2043, 2020.

[9] M. Ulinic and S. House, *Network automation at scale*. O'Reilly Media, Inc, 2017

[10] R.A. Wiryawan, "Pengembangan Aplikasi Otomatisasi Administrasi Jaringan Berbasis *Website* Menggunakan Bahasa Pemrograman *Python". SIMETRIS*, 741- 752, 2019

[11] D. Triasanti. *Konsep Dasar Python* ,Surabaya: Sulita Jaya, 2002.

[12] G. Rossum and the Python development team. *Python Tutorial*. *Python Software Foundation*. 2020.

[13] P. Mihăilă, T. Bălan, R. Curpen and F. Sandu, F. *Network Automation and Abstraction Using Python Programming Methods.* Macro 2015, vol 2 (1), p. 95-103, 2019.

[14] M. Zadka, *DevOps in Python*: *Infrastructure as Python*. Belmont, CA, USA: Apress Media, p.111, 2019.

[15] I.D. Cahyani, I.D, "Sistem Keamanan Enkripsi *Secure Shell* (SSH) untuk Keamanan Data", *Jurnal Universitas Pandanaran*, vol 8 (16), 2010.

[16] U. Hunkeler, H. L. Truong and A. Stanford-Clark., "MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks". *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*, Bangalore, India, 2008. p. 791-798, 2008.

[17] Z. Abilovani, W. Yahya and F. Bakhtiar, "Implementasi Protokol MQTT untuk Sistem Monitoring Perangkat IOT". *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer*, vol 2 (12), p. 7521-7527, 2018.

Implementasi dan Analisis Performa Sistem *Monitoring* Suhu dan Kelembaban Kondisi Ruang *Server* pada Jaringan Berbasis Lora

Siti Zubaidah Effendi[1], Unan Yusmaniar Oktiawati[1,*]
[1]Departemen Teknik Elektro dan Informatika, Sekolah Vokasi, Universitas Gadjah Mada;
siti.zubaidah.e@mail.ugm.ac.id
[*]Korespondensi: unan_yusmaniar@ugm.ac.id;

Abstract – A server room is a room used to store applications, data and network devices such as routers. The server room is the data center of a company or institution in which there are applications and databases that store all important and valuable information for the company or institution concerned, therefore the server room must always be in good temperature conditions because the devices work for 24 hours no stopping. If there is a significant increase in temperature, it can cause system performance to be disrupted or hardware damage occurs. The cooler in the server room is not optimal because the cooler is often constrained by frequent power outages. From these problems, a solution is needed to be able to monitor the system remotely so that changes in room temperature can be seen in real time. The Internet of Things (IoT) is the solution to this problem, by using LoRa, which then sends the data to the Telegram application that has been installed on the smartphone so that real-time temperature changes can be obtained.

Keywords : internet of thing, quality of service, temperature and humidity, raspberry pi, long range

Intisari – Ruang *server* adalah ruangan yang digunakan untuk menyimpan aplikasi, data dan perangkat jaringan seperti *router*. Ruang *server* merupakan pusat data suatu perusahaan atau lembaga yang di dalamnya terdapat aplikasi dan basis data yang menyimpan segala informasi penting dan bernilai bagi perusahaan atau lembaga yang bersangkutan, oleh karena itu ruang *server* harus selalu dalam kondisi suhu yang baik karena perangkat yang bekerja selama 24 jam tanpa henti. Apabila terjadi kenaikan suhu yang signifikan, dapat menyebabkan kinerja sistem menjadi terganggu atau terjadi kerusakan pada perangkat keras. Pendingin yang terdapat di ruang *server* dirasa belum maksimal karena pendingin tersebut sering kali terkendala dengan aliran listrik yang sering padam. Dari permasalahan tersebut diperlukan solusi untuk dapat memantau sistem dari jarak jauh sehingga dapat diketahui perubahan suhu ruang secara realtime. *Internet of Things* (*IoT*) menjadi solusi dari permasalahan ini, dengan menggunakan *LoRa* yang selanjutnya data akan dikirimkan ke aplikasi *Telegram* yang sudah dipasang pada *smartphone* sehingga dapat diperoleh perubahan suhu secara *real-time*.

Kata kunci *: internet of things, quality of service, temperature and humidity, raspberry pi, long range*

I. PENDAHULUAN

Internet of Things atau biasa dikenal dengan *IoT* merupakan salah satu perkembangan teknologi internet dan kemungkinan besar akan menjadi tren di masa depan. *Internet of Things* (*IoT*) memungkinkan pengguna untuk mengelola dan mengoptimalkan elektronik dan peralatan listrik yang menggunakan internet. Salah satu penerapan *Internet of Things* yaitu sistem *monitoring* atau *controlling* yang menggunakan sensor dan akuator pada sebuah lingkungan dengan kondisi tertentu [1].

Ruang *server* adalah ruangan yang digunakan untuk menyimpan aplikasi, data dan perangkat jaringan seperti *router* dan *hub*, sehingga ruang *server* merupakan aset penting bagi suatu perusahaan atau lembaga yang menerapkan teknologi informasi sebagai penunjang dalam kegiatan sehari-hari. Hal ini karena pada ruang *server* terdapat aplikasi dan basis data yang menyimpan segala informasi penting yang bernilai bagi perusahaan atau lembaga yang bersangkutan, oleh karena itu ruangan *server* harus selalu dalam kondisi baik. Selain itu perangkat yang berada di ruang *server* harus bekerja selama 24 jam tanpa henti, sehingga perlu ditunjang dengan *monitoring* untuk memastikan perangkat bekerja dengan baik. Karena sifatnya yang tak henti-hentinya, sebuah pusat data membutuhkan pemantauan yang sangat baik dan perlindungan data dari kerusakan yang disebabkan dari kecelakaan ke infrastruktur jaringan.

Ancaman lingkungan paling umum terhadap ruang *server* adalah suhu, kelembaban, kebocoran air, human error dan pemadaman listrik. Banyak ancaman yang muncul seperti suhu dan kelembaban yang mempersulit pemantauan lingkungan pada ruang *server*. Suhu ruang adalah faktor penunjang yang berpengaruh besar pada kinerja *server* dan perangkat jaringan lainnya karena jika suhu tidak mencapai rentang $18°C - 23°C$ maka kinerja kipas *server* akan bekerja ekstra untuk menstabilkan suhu *server*. Suhu yang terlalu tinggi mengakibatkan komponen pada perangkat seperti *harddisk* cepat rusak, sedangkan suhu yang terlalu rendah mengakibatkan udara mengalami pengembunan dan kelembaban menjadi tinggi sehingga terjadi hubungan arus pendek pada perangkat.

Berdasarkan permasalahan tersebut, penulis mengusulkan sebuah purwarupa sistem berbasis mikrokontroler yang dapat memantau suhu dan kelembaban. Sistem ini nantinya akan mengirimkan notifikasi pada *Telegram* ketika terjadi kenaikan atau penurunan suhu pada ruang *server*.

II. TINJAUAN PUSTAKA

Internet of Things atau yang dikenal dengan istilah *IoT* merupakan teknologi yang memungkinkan penggunanya untuk mengelola dan mengoptimalkan elektronik dan peralatan listrik yang menggunakan internet. *Internet of Things* adalah segala sesuatu atau perangkat elektronik yang dapat berinteraksi secara langsung dengan pengguna yang digunakan untuk kebutuhan *monitoring* atau mengendalikan perangkat tersebut melalui internet.

Sistem monitoring adalah suatu sistem yang melakukan proses pemantauan secara terus menerus [2]. Sistem monitoring dibutuhkan dalam proses pemantauan keadaan suatu objek yang diamati guna mendapatkan informasi yang tepat waktu. *Monitoring* infrastruktur jaringan merupakan suatu hal yang sangat penting karena perangkat jaringan baik itu *server*, *router*, maupun perangkat jaringan lainnya dapat tetap terjaga kinerjanya. Pada penelitian yang dilakukan oleh [3] dibuat sebuah aplikasi *monitoring* yang dirancang dengan

menggunakan *Bot Telegram* dan diintegrasikan dengan *server OpenNMS* dan *Router Mikrotik* yang bertujuan agar pengguna dapat memantau *server* maupun perangkat jaringan dengan mudah.

Perangkat pada ruang *server* memiliki fungsi yang penting sehingga tidak boleh mengalami gangguan. Beberapa penyebab kinerja perangkat di ruang *server* terganggu adalah *overheat* dan kelembaban ruangan *server*. Kelembaban yang tinggi pada ruangan akan mempengaruhi usia perangkat [4]. Pada tahun 2019 dilakukan sebuah penelitian dengan membuat sebuah rancang bangun pengatur suhu dan kelembaban ruang *server* berbasis *IoT* dengan *DHT 11* sebagai sensor suhu dan *Arduino Uno* sebagai mikrokontrolernya, dan *Ethernet Shield* agar dapat terhubung pada internet. Dari penelitian ini dihasilkan bahwa sensor dapat memberikan informasi keadaan suhu di dalam ruangan dengan baik dan mikrokontroler yang menerima informasi tersebut akan mengatur sistem pendingin jika suhu dalam ruangan terlalu rendah atau tinggi [5].

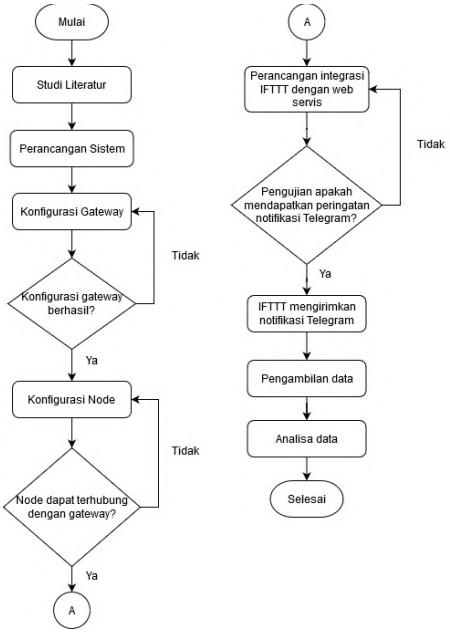
Pada tahun 2019 juga dilakukan penelitian dengan membuat rancang bangun sistem *monitoring* suhu dan kelembaban untuk mendekteksi perubahan suhu dan kelembaban pada ruangan dengan menggunakan *NodeMCU ESP8266* sebagai kontrol utamanya, *DHT 11* sebagai sensor pendeteksi suhu dan kelembaban, selanjutnya data akan dikirim ke aplikasi *Telegram* yang sudah ter-*install* pada *smartphone*. Pada penelitian ini juga ditambahkan sensor gas *MQ-2* untuk mendeteksi adanya asap, jika lebih besar dari 500 *ppm* maka *buzzer* akan berbunyi. Dari penelitian ini didapatkan hasil bahwa pengambilan data suhu *DHT 11* dengan Termometer hampir sempurna karena perbedaan hasil tidak lebih dari 2°C atau masih dalam batas normal sesuai dengan ketentuan yang berlaku 20°*C*-25°*C*. Dengan kesalahan pembacaan 1°*C*-2°*C*. Sedangkan pengukuran kelembaban pada ruangan *server* menggunakan *DHT 11* dengan *Hygrometer* sebagai pembanding mendapatkan hasil selisih 1%-6%. Data hasil pembacaan *DHT 11* dan *MQ-2* dikirimkan ke *Telegram* menggunakan *NodeMCU ESP866* yang dapat terhubung dengan *wi-fi* dengan jarak maksimal dalam ruang *server* dengan jarak 20 meter [6].

Salah satu perkembangan *IoT* adalah pengaplikasian *LoRa* (*Long Range*). Pada *IoT*, *LoRa* digunakan dalam komunikasi M2M (*Machine to Machine*), contohnya adalah untuk pengembangan *Smart City*, dengan adanya *LoRa* sensor-sensor dapat berinteraksi langsung dengan manusia di mana saja dan kapan saja. Salah satu keunggulan teknologi *LoRa* adalah penggunaan daya rendah dan memiliki jangkauan komunikasi yang luas yaitu lebih dari 2 km. Namun pengiriman data ini tidak dapat langsung menuju ke *server*, sehingga diperlukan *gateway* sebagai penghubung antar perangkat di *node sensors* dengan *server*. Pada tahun 2019 dilakukan sebuah penelitian dengan melakukan pengembangan sistem *gateway* agar dapat menghubungkan komunikasi atara *node sensors* dengan *server* menggunakan modul komunikasi *LoRa* dan protokol *MQTT*. Hasil pengujian kinerja *succesful rate gateway* dengan menggunakan variabel jarak, besar data, dan interval yang berbeda menunjukkan bahwa pada jarak 400 meter kinerja *gateway* dalam menerima dan meneruskan data ke pusat data lebih baik daripada jarak 200 meter [7].

Dalam penerapan *IoT* dibutuhkan suatu jaringan komunikasi yang sesuai dengan kebutuhan sistem. Salah satu protokol yang sesuai dengan penerapan konsep *IoT* adalah protokol *Message Queue Telemetry Transport* (*MQTT*) yang sering digunakan dalam berbagai sistem yang menggunakan konsep *IoT* salah satunya sistem *monitoring*. Pada tahun 2018 dilakukan sebuah penelitian oleh [8] yang mengimplementasikan konsep *Internet of Things* pada sistem *monitoring* banjir menggunakan protokol *MQTT*. Berdasarkan hasil pengujian, diperoleh kesimpulan bahwa protokol *MQTT* dapat digunakan pada sistem *monitoring* banjir. Tingkat akurasi yang didapat dari pengujian sistem ini adalah 97,801% dengan standar deviasi yang diperoleh sebesar ± 0.0309 cm.

III. METODE PENELITIAN

Tahapan penelitian yang dilakukan dalam proyek akhir ini yang pertama adalah dilakukan studi literatur dari beberapa jurnal dan buku, kemudian mulai dilakukan perancangan dan analisis sistem. Selanjutnya dilakukan persiapan untuk perangkat keras dan perangkat lunak yang dibutuhkan. Dilanjutkan dengan konfigurasi *gateway* kemudian konfigurasi pada *node*. Kemudian dilakukan konfigurasi agar *node* dan *gateway* dapat terhubung kemudian dilanjutkan perancangan integrasi antara *IFTTT* dengan *web service*. Gambar 1 merupakan diagram alir metode penelitian yang dilakukan oleh penulis.



Gambar 1. Bagan Alir Proyek Akhir

A. Alat dan Bahan

Dalam melaksanakan penelitian ini, dibutuhkan perangkat keras dan perangkat lunak guna menunjang penelitian yang akan dilakukan. Adapun perangkat tersebut diantaranya sebagai berikut:

1) Perangkat keras

- 2 *Unit Raspberry Pi 3 B+*
- 2 *Unit LoRa Dragino GPS/HAT*
- 1 Sensor Suhu *DHT11*
- Kabel *Jumper*
- *Android*

- *PC/Laptop*
- *Power Supply/Adaptor* 5V

2) Perangkat lunak

- *Raspbian*
- *Python*
- *PosgreSQL*
- *Telegram*

B. Pengujian dan Pengambilan Data

Pengujian *Quality of Service* (*QoS*) dilakukan untuk menilai kualitas pengiriman data dari *node* ke *gateway* hingga sampai ke pengguna untuk membuktikan fitur pemantauan bersifat *real-time*. Parameter yang diuji adalah *Delay*, *Packet Loss* dan *Packet Delivery*.

Nilai parameter-parameter *QoS* didapat dari 2 skenario yang akan dijalankan dengan 10 perulangan percobaan pada tiap skenario sehingga total terdapat 20 percobaan. Skenario berupa pengiriman data dengan interval waktu yang berbeda-beda pada jaringan yang sama. Skenario 1 dilakukan pengiriman data sebanyak 100 paket data dan antara satu paket data dengan paket data berikutnya diberikan jeda waktu selama 5 detik, sehingga dapat diasumsikan bahwa pengiriman data dilakukan setiap 5 detik. Skenario 2 dilakukan dengan mengirim paket data sebanyak 100 paket data dan diberikan jeda waktu selama 10 detik antara pengiriman satu paket data dengan paket data berikutnya sehingga dapat dikatakan bahwa dilakukan pengiriman paket data setiap 10 detik. Berikut Tabel 1 yang menunjukkan skenario pengujian.

Tabel 1. Skenario Pengujian

Skenario	**Eksperimen**
Skenario 1	Pengiriman data per 5 detik
Skenario 2	Pengiriman data per 10 detik

1) Pengambilan Nilai *Delay*

Nilai *delay* diperoleh dari perhitungan waktu paket data ketika diterima dikurangi dengan waktu paket data dikirimkan atau untuk lebih jelasnya dengan menggunakan persamaan berikut:

$$Delay = Waktu_{penerimaan\ paket} - Waktu_{pengiriman\ paket} \quad (1)$$

Waktu pengirim dan penerimaan paket data diketahui dari hasil rekaman trafik data yang telah dibuat dan ditampilkan pada *LoRa* yang kemudian diolah pada *Excel* hingga menjadi seperti pada Gambar 2.

No.	Percobaan 1		Percobaan 2		Percobaan 3		Percobaan 4	
	Gateway	Node	Gateway	Node	Gateway	Node	Gateway	Node
1	0.370886	Event EV_TXCOMPLETE	0.367949	Event EV_TXCOMPLETE	0.379923	Event EV_TXCOMPLETE	0.3616	Event EV_TXCOMPLETE
2	0.353567	Event EV_TXCOMPLETE	0.353715	Event EV_TXCOMPLETE	0.353381	Event EV_TXCOMPLETE	0.367393	Event EV_TXCOMPLETE
3	0.353697	Event EV_TXCOMPLETE	0.361344	Event EV_TXCOMPLETE	0.355041	Event EV_TXCOMPLETE	0.357209	Event EV_TXCOMPLETE
4	0.362658	Event EV_TXCOMPLETE	0.358837	Event EV_TXCOMPLETE	0.340441	Event EV_TXCOMPLETE	0.366125	Event EV_TXCOMPLETE
5	0.3599	Event EV_TXCOMPLETE	0.367185	Event EV_TXCOMPLETE	0.349184	Event EV_TXCOMPLETE	0.358834	Event EV_TXCOMPLETE

Gambar 2. Nilai *Delay*

2) Pengambilan Nilai *Packet Loss*

Packet Loss yang diambil pada pengujian ini merupakan paket data yang hilang ketika proses pengiriman data.

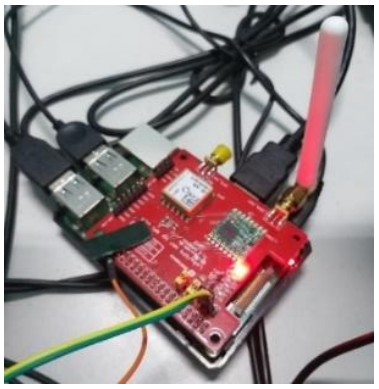
3) Pengambilan Nilai *Packet Delivery*

Packet Delivery yang diambil pada pengujian ini adalah paket data yang dikirimkan dari *node* ke *gateway* dalam 100 data yang dikirimkan.

IV. HASIL DAN PEMBAHASAN

Penelitian ini menghasilkan purwarupa sistem pemantau suhu dan kelembaban ruang *server* yang terdiri dari dua buah *node* yaitu *node sensors* dan *node gateway*. *Node sensors* berfungsi untuk membaca suhu dan kelembaban pada rak *server*. *Node gateway* berfungsi untuk menerima hasil pembacaan dari *node sensors* yang selanjutnya akan dikirimkan ke *server*. Gambar 3 adalah tampilan *node gateway* yang komponen-komponennya sudah dirakit dan siap untuk digunakan. Sedangkan Gambar 4 merupakan *node sensors* yang berfungsi untuk membaca suhu dan kelembaban ruangan dimana pada *node* tersebut terdapat sensor *DHT 11* untuk membaca suhu dan kelembaban. *Node sensors* juga akan mengirimkan data ke *node gateway* ketika suhu dan kelembaban ruangan melebihi dari batas yang telah ditetapkan.

Gambar 3. *Node Gateway*



Gambar 4. *Node Sensors*

Hasil pembacaan suhu dan kelembaban yang melebihi batas yang telah ditentukan akan dikirimkan melalui notifikasi pada aplikasi *Telegram.*

Gambar 5. Telegram *IFTTT* yang menunjukkan suhu dan kelembaban melebihi batas yang telah ditentukan

A. Hasil Pengujian Sistem

Penulis melakukan pengujian sensor *DHT11* dengan variasi jeda waktu pengiriman data. Variasi jeda tersebut sebesar 5 detik dan 10 detik pada setiap percobaan. Setiap variasi jeda tersebut dilakukan sebanyak 10 kali percobaan. Sistem pemantauan suhu dan kelembaban yang telah dibuat sudah terintegrasi dengan sistem notifikasi melalui *Telegram*. Jika suhu dan kelembaban melebihi batas yang sudah ditentukan, maka akan mengirimkan notifikasi melalui *Telegram* seperti pada Gambar 6.

Gambar 6. Notifikasi *Telegram*

B. Hasil Pengujian *Delay*

Pengujian *delay* dilakukan di Gedung Sekip Unit IV tepatnya di ruang 128. Pengujian dilakukan dengan 2 skenario yaitu diberikan jeda 5 detik dan 10 detik. Dalam mencari nilai *delay*, tidak dapat dilakukan dengan satu kali percobaan saja. Oleh karena itu penulis melakukan 10 kali percobaan untuk masing-masing skenario yaitu 10 percobaan untuk skenario jeda 5 detik dan 10 percobaan pada skenario 10 detik. Setiap percobaan dilakukan pengiriman data sebanyak 100 data, setelah selesai melakukan percobaan dicari nilai rata-ratanya. Hal tersebut dilakukan agar data yang didapat *valid*. Rata-rata nilai *delay* dari hasil percobaan pada setiap skenario termasuk dalam kategori sangat bagus karena nilainya kurang dari 150 *ms*. Tabel 2 menunjukkan indeks dan kategori *delay* yang menentukan kualitas layanan menurut *Telecommunication Internet Protocol Harmonization Over Network* (TIPHON).

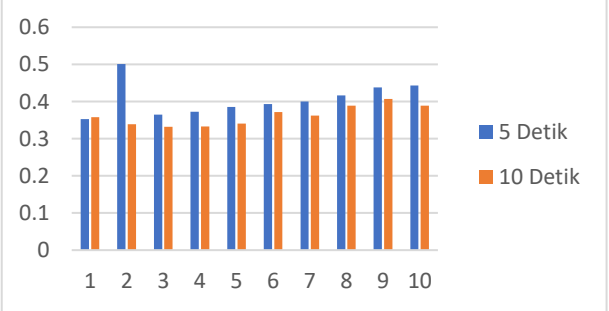
Tabel 2. Performa Jaringan Berdasarkan *Delay*

Kategori	***Delay***
Sangat Bagus	< 150ms
Bagus	150ms s/d 300ms
Sedang	300ms s/d 450ms
Jelek	>450 ms

Hasil pengukuran *delay* dapat dilihat pada Tabel 3 dan didapatkan nilai rata-rata setelah sepuluh kali percobaan, pada *delay* 5 detik didapat nilai rata-rata 0.4068275 dan pada *delay* 10 detik didapat nilai rata-rata 0.3621856.

Tabel 3. Hasil Pengukuran *Delay*

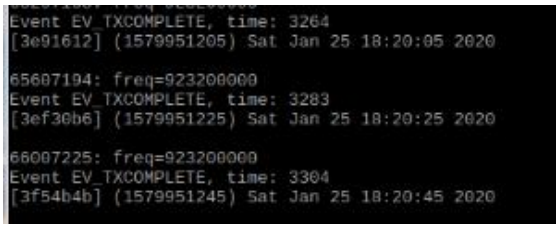
Percobaan Ke-n	***Delay* n (detik)**	
	5	**10**
1	0.353082	0.357532
2	0.500792	0.338959
3	0.364898	0.331779
4	0.372503	0.333313
5	0.385375	0.340870
6	0.393357	0.371652
7	0.400360	0.362293
8	0.416657	0.389247
9	0.437990	0.407397
10	0.443261	0.388814
Rata-Rata	0.4068275	0.3621856



Gambar 7. Grafik Pengujian *Delay*

C. Hasil Pengujian *Packet Loss*

Dalam pengambilan data *packet loss* dilakukan dua skenario, skenario pertama dilakukan pengiriman data dari *node* menuju *gateway* sebanyak 100 data dan antara pengiriman satu data dengan data berikutnya diberikan jeda waktu selama 5 detik. Skenario kedua juga dilakukan pengiriman data dari *node* menuju *gateway* sebanyak 100 data dan antara pengiriman satu data dengan data berikutnya diberikan jeda waktu selama 10 detik. Pada dua skenario tersebut dilakukan sebanyak 10 percobaan pada setiap skenario sehingga total terdapat 20 percobaan dengan 10 percobaan pada skenario jeda 5 detik dan 10 percobaan pada skenario jeda 10 detik. Data *packet loss* didapat ketika *node* mengirimkan data ke *gateway* hingga muncul *Event EV_TXCOMPLETE* seperti pada Gambar 8 berikut



Gambar 8. Hasil Pengiriman Data di *Node*

Setelah dilakukan masing-masing 10 percobaan di tiap skenario, *packet loss* yang didapat bernilai 0% pada setiap skenario atau dapat dikatakan bahwa tidak ada paket yang hilang selama pengiriman data berlangsung. Nilai 0% tersebut didapat dari persamaan berikut ini:

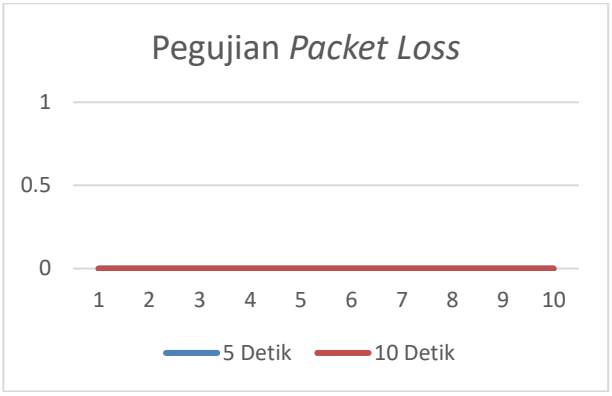
$$\textit{Packet Loss Rasio (PLR)}\ \% = \frac{\textit{Paket Data Hilang}}{\textit{Paket Data Dikirim}}\ x\ 100\%$$
$$\textit{Packet Loss Rasio (PLR)}\ \% = \frac{0}{100}\ x\ 100\%$$
$$= 0\% \quad (2)$$

Hal ini dapat dilihat pada Tabel 4 berikut:

Tabel 4. Hasil Pengukuran *Packet Loss*

Percobaan Ke-n	***Packet Loss* Skenario n Detik**	
	5	**10**
1	0%	0%
2	0%	0%
3	0%	0%
4	0%	0%
5	0%	0%
6	0%	0%
7	0%	0%
8	0%	0%
9	0%	0%
10	0%	0%

Dari Tabel 4 dapat digambarkan grafik seperti pada Gambar 9.



Gambar 9. Grafik Pengujian *Packet Loss*

Setelah dilakukan percobaan pada masing-masing skenario dan menghasilkan nilai *packet loss* 0% maka dapat disimpulkan bahwa *packet loss* untuk pengiriman data dari *node* menuju *gateway* termasuk kategori sangat bagus menurut *TIPHON*. Hal ini dapat dilihat pada Tabel 5 terdapat empat kategori penurunan performasi jaringan berdasarkan nilai *packet loss* menurut *TIPHON*.

Tabel 5. Performa Jaringan Berdasarkan *Packet Loss*

Kategori	***Packet Loss***
Sangat Bagus	0%
Bagus	3%
Sedang	15%
Jelek	>25%

D. Hasil Pengujian *Packet Delivery*

Packet Delivery yang diambil pada pengujian ini adalah paket data yang dikirimkan oleh *node sensors* dan berhasil sampai ke *node gateway* dalam satu kali pengiriman. Sama halnya dengan pengujian *packet loss*, dilakukan 2 skenario percobaan dengan masing-masing skenario dilakukan percobaan sebanyak 10 kali. Dalam satu kali percobaan dikirimkan paket data sebanyak 100 data. Pembeda antara skenario satu dan skenario dua adalah interval waktu pengiriman data. Skenario satu diberikan jeda waktu pengiriman paket data antara satu paket data dengan paket data berikutnya selama 5 detik sedangkan skenario dua diberikan jeda waktu pengiriman selama 10 detik. Nilai *packet delivery* didapatkan dari persamaan:

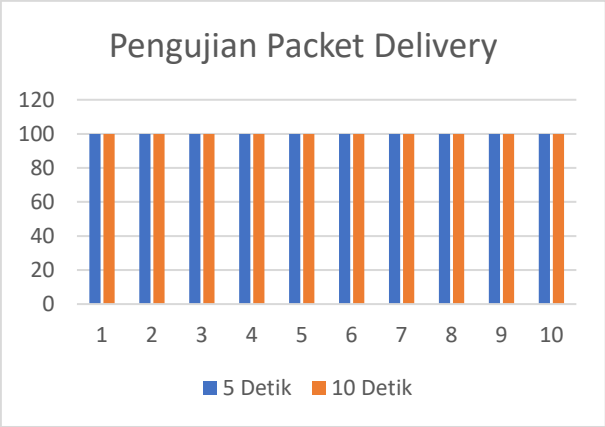
$$\textit{Packet Loss Rasio (PLR)}\ \% = \frac{\textit{Paket Data Hilang}}{\textit{Paket Data Dikirim}}\ x\ 100\%$$
$$\textit{Packet Loss Rasio (PLR)}\ \% = \frac{100}{100}\ x\ 100\%$$
$$= 100\% \quad (3)$$

Hasil dari percobaan kedua skenario tersebut dapat dilihat pada Tabel 6 berikut:

Tabel 6. Hasil Pengukuran *Packet Delivery*

Percobaan Ke-n	***Packet Delivery* Skenario n Detik**	
	5	**10**
1	100%	100%
2	100%	100%
3	100%	100%
4	100%	100%
5	100%	100%
6	100%	100%
7	100%	100%
8	100%	100%
9	100%	100%
10	100%	100%

Dari Tabel 6 di atas kemudian dapat digambarkan grafik seperti pada Gambar 10 berikut:



Gambar 10. Grafik Pengujian *Packet Delivery*

Dari hasil pengujian yang telah dilakukan didapat nilai *packet delivery* sebesar 100% atau dapat disimpulkan bahwa tidak ada paket yang hilang selama pengiriman data. Menurut *TIPHON* performasi jaringan berdasarkan nilai dari *packet delivery* dari dua skenario yang telah dijalankan dapat dikategorikan sebagai sangat bagus, hal ini dapat dilihat pada Tabel 7.

Tabel 7. Performa Jaringan Berdasarkan *Packet Delivery*

Kategori	***Packet Delivery***
Sangat Bagus	100%
Bagus	97%
Sedang	85%
Jelek	>75%

V. KESIMPULAN

Berdasarkan analisis pengujian yang telah dilakukan, dapat diambil kesimpulan sebagai berikut:

- Sistem yang dibuat berupa alat pendeteksi suhu dan kelembaban pada ruang *server* dengan menggunakan sensor *DHT 11*, ketika sensor mendeteksi adanya perubahan suhu yang terjadi, *Raspberry Pi* dan *LoRa* bekerja dengan mengirimkan notifikasi berupa pesan pada aplikasi *Telegram*.
- Sistem dapat mengintegrasikan sensor suhu dan kelembaban dengan mikrokontroller *Raspberry Pi* menggunakan *LoRa* secara *real time* sehingga ketika terjadi perubahan suhu *LoRa node* mengirimkan paket kepada *LoRa gateway* yang selanjutnya akan diteruskan ke *server* sampai pada akhirnya terdapat pesan dari *Telegram* berupa notifikasi.
- Performa kinerja *LoRa* pada parameter *packet loss* dapat dikatakan sangat bagus dengan persentase *packet loss* penelitian 0%.
- Kinerja dari sistem yang telah diuji terdapat kendala jika dilakukan pengujian terhadap jarak antara *node sensors* dan *gateway*, kendala tersebut adalah *gateway* mampu menerima paket data yang dikirimkan oleh *node* pada satu hingga tiga percobaan pertama namun pada percobaan berikutnya *gateway* tidak menerima paket data yang dikirimkan.

Berikut adalah saran yang dapat digunakan untuk mengembangkan penelitian selanjutnya:

- Menerapkan protokol komunikasi nirkabel lain. Diharapkan kedepannya dapat dilakukan komparasi kinerja berbagai protokol yang beragam.
- Jumlah *gateway* yang digunakan pada penelitian ini hanyalah satu. Diharapkan untuk kedepannya dapat menggunakan jumlah *gateway* lebih dari satu.
- Menerapkan serangan untuk mengetahui efeknya terhadap kinerja *LoRa*.

DAFTAR PUSTAKA

[1] H. Rochman, R. Primananda and H. Nurwasito, "Sistem Kendali Berbasis Mikrokontroler Menggunakan Protokol MQTT pada Smarthome," *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer*, 2017.

[2] Mudjahidin and N. D. P. Putra, "Rancang Bangun Sistem Informasi Monitoring Perkembangan Proyek Berbasis Web Studi Kasus di Dinas Bina Marga dan Pemantusan," *Jurnal teknik Industri*, 2012.

[3] R. J. Putra, N. P. Sastra and D. M. Wiharta, "Pengembangan Komunikasi Multikanal untuk Monitoring Infrastruktur Jaringan Berbasis Bot Telegram," *Jurnal SPEKTRUM*, 2018.

[4] S. Taftazanie, A. B. Prasetijo and E. D. Widianto, "Aplikasi Pemantau Perangkat Jaringan Berbasis Web Menggunakan Protokol SNMP dan Notifikasi SMS," *Jurnal Teknologi dan Sistem Komputer*, 2017.

[5] M. I. Z., J. Endri and Sarjana, "Rancang Bangun Pengatur Suhu dan Kelembaban Ruang Server Berbasis IoT," *Prosiding SENIATI*, 2019.

[6] G. Santoso, S. Kristiyana, S. Hani and A. M. Mujahidin, "Rancang Bangun Sistem Monitoring Suhu dan Kelembaban pada Ruang Server Berbasis IoT (Internet of Things)," *Jurnal Teknologi Technoscientia*, 2019.

[7] H. Arijuddin, A. Bhawiyuga and K. Amron, "Pengembangan Sistem Perantara Pengiriman Data Menggunakan Modul Komunikasi LoRa dan Protokol MQTT Pada Wireless Sensor Network," *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer*, 2019.

[8] C. P. R. &. A. K. Hasiholan, "Implementasi Konsep Internet of Things pada Sistem Monitoring Banjir menggunakan Protokol MQTT," *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer*, 2018.

Perancangan Sistem Monitoring Performa Aplikasi Menggunakan Opentelemetry dan Grafana Stack

Guntoro Yudhy Kusuma[1], Unan Yusmaniar Oktiawati[1,*]
[1]Departemen Teknik Elektro dan Informatika, Sekolah Vokasi, Universitas Gadjah Mada;
guntoro.y.k@mail.ugm.ac.id
[*]Korespondensi: unan_yusmaniar@ugm.ac.id;

Abstract – The increasingly massive use of digital technology requires that the application architecture be designed to have high availability and reliability. This is because when an application cannot be accessed, it will cause no small loss to the organization. Therefore, the development and operation teams must be able to detect when their system is not working well. For that, we need a system that can monitor application performance. In this research, a system is developed to collect telemetry data, namely metrics and traces from an online donation backend application based on the REST API. OpenTelemetry produces telemetry as an open-source telemetry instrumentation tool. Then the telemetry data is collected by the OpenTelemetry Collector which is then stored on the backend of each telemetry. Metrics are sent to Prometheus and traces are sent to Jaeger. The data metrics collected are throughput, request latency, and error rate which are visualized using the Grafana dashboard. The test results show that the monitoring system can collect real-time metrics data with an average delay of 13,8 seconds. The system can also detect when an anomaly occurs in the app and sends notifications via Slack. In addition, the trace data collected can be used to simplify the debugging process when an error occurs in the application. However, the implementation of OpenTelemetry in a REST API-based backend application to monitor metrics and traces has a negative impact on the performance of the application itself, which can reduce the number of request throughput with an average decrease of 23.32% and increase request latency with an average increase of 22.80%.

Keywords - monitoring, APM, system observability, OpenTelemetry, Grafana

Intisari – Semakin masifnya penggunaan teknologi digital mengharuskan arsitektur aplikasi didesain agar memiliki ketersediaan dan keandalan yang tinggi. Hal ini karena ketika sebuah aplikasi tidak dapat diakses, akan menyebabkan kerugian yang tidak sedikit bagi organisasi. Oleh karena itu tim *developer* maupun *operation* harus bisa mendeteksi ketika sistem mereka sedang tidak baik-baik saja. Untuk itulah diperlukan sebuah sistem yang dapat memonitor performa aplikasi. Pada penelitian kali ini dikembangkan sebuah sistem yang dapat mengumpulkan data telemetri yaitu *metrics* dan *traces* dari sebuah aplikasi *backend* donasi *online* yang berbasis *REST API*. Telemetri tersebut dihasilkan oleh *OpenTelemetry* sebagai alat instrumentasi telemetri yang *open-source*. Kemudian data telemetri tersebut dikumpulkan oleh *OpenTelemetry Collector* yang selanjutnya disimpan pada *backend* masing-masing telemetri. *Metrics* dikirimkan menuju Prometheus dan *traces* dikirimkan ke *Jaeger*. Data *metrics* yang dikumpulkan adalah *throughput, request latency,* dan *error rate* yang divisualisasikan menggunakan *Grafana dashboard*. Hasil pengujian menunjukkan bahwa sistem monitoring dapat mengumpulkan data *metrics* secara *realtime* dengan waktu tunda rata-rata 13,8 detik. Sistem juga dapat mendeteksi ketika terjadi anomali pada aplikasi dan mengirimkan pemberitahuan melalui *Slack*. Selain itu, data *traces* yang dikumpulkan dapat digunakan untuk mempermudah proses *debugging* ketika terjadi kesalahan pada aplikasi. Namun, implementasi *OpenTelemetry* dalam aplikasi *backend* berbasis *REST API* untuk memonitor metrics dan *traces* memberikan dampak negatif pada performa aplikasi itu sendiri yaitu dapat menurunkan jumlah *request throughput* dengan penurunan rata-rata 23,32% dan menaikkan *request latency* dengan kenaikan rata-rata 22,80%.

Kata kunci – *monitoring, APM, system observability, OpenTelemetry, Grafana*

I. PENDAHULUAN

Semakin masifnya penggunaan teknologi digital mengharuskan arsitektur aplikasi didesain agar memiliki ketersediaan dan keandalan yang tinggi. Hal ini karena ketika sebuah aplikasi tidak dapat diakses, akan menyebabkan kerugian yang tidak sedikit bagi organisasi. Sebagai contoh, *Amazon* menemukan setiap 100 milidetik latensi membuat mereka kehilangan 1% dalam penjualan. *Google* menemukan tambahan 5% dalam waktu pembuatan halaman hasil pencarian menurunkan lalu lintas sebesar 20%. Sebuah pialang saham dapat kehilangan pendapatan sebesar *$4* per milidetik jika platform perdagangan elektronik mereka tertinggal 5 milidetik dari pesaingnya [1].

Untuk memastikan bahwa sistem aplikasi memberikan kualitas layanan yang diharapkan, sangat penting untuk memiliki informasi terkini tentang sistem untuk mendeteksi masalah dan menyelesaikannya secara efektif [2]. Informasi yang diperlukan tersebut berupa data telemetri *metrics*, *traces*, dan *logs* yang dikumpulkan dari berbagai komponen pada sebuah sistem seperti infrastrukutur, sistem operasi, *database*, dan aplikasi [3].

Pada penelitian kali ini dikembangkan sebuah sistem *monitoring* yang akan mengumpulkan telemetri berupa *metrics* dan *traces* dari sebuah backend sistem donasi online yang berbasis *REST API*. Telemetri tersebut dihasilkan oleh *OpenTelemetry* sebagai alat instrumentasi telemetri yang *open-source* dan kemudian dikumpulkan oleh *OpenTelemetry Collector*. Selanjutnya, telemetri berupa *metrics* dikirimkan menuju alat *monitoring* dan *alerting Prometheus* yang kemudian dapat divisualisasikan menggunakan *Grafana* secara *realtime*. Telemetri berupa *traces* akan diteruskan oleh *OpenTelemetry Collector* menuju ke *Jaeger* yang diintegrasikan pada *Grafana*. Sistem *monitoring* menggunakan data *metrics* yang dikumpulkan untuk mendeteksi terjadinya anomali pada aplikasi kemudian memberikan peringatan melalui *Slack*. Kemudian, data *traces*

yang dikumpulkan dapat digunakan untuk mempermudah proses *debugging* ketika terjadi kesalahan pada aplikasi.

II. DASAR TEORI

A. *Observability* dan *Monitoring*

Menurut JJ Tang pada *blogpost* di situs *devops.com*, dalam industri TI secara khusus, *observability* dapat didefinisikan sebagai penggunaan *logs*, *metrics*, dan *traces* untuk memahami keadaan sistem perangkat lunak yang kompleks. Sedangkan *monitoring* adalah proses pengumpulan data dari sebuah sistem. Di industri TI, *monitoring* adalah bagaimana tim mengumpulkan *logs*, *metrics*, dan *traces* dan memindahkannya ke tempat di mana mereka dapat dianalisis [4].

Di sini dapat dipahami bahwa *observability* dan *monitoring* adalah dua hal yang berbeda tetapi saling berkaitan. *Observability* berfokus pada proses menginterpretasi dan memahami data, sedangkan *monitoring* hanyalah proses pengumpulan data tersebut. Dengan adanya *monitoring* kita mendapatkan peringatan ketika terjadi sesuatu yang salah pada sistem. Kemudian dengan *observability* kita dapat mengetahui mengapa sesuatu tersebut bisa salah dan bagaimana untuk memperbaikinya.

B. *Application Performance Management (APM)*

*Application Perfromance Managemen*t (*APM*) adalah disiplin operasi inti TI yang bertujuan untuk mencapai tingkat kinerja yang memadai selama operasi. APM terdiri dari metode, teknik, dan alat untuk i) terus memantau keadaan sistem aplikasi dan penggunaannya, serta untuk ii) mendeteksi, mendiagnosis, dan menyelesaikan masalah terkait kinerja menggunakan data yang dipantau. Perlu ditekankan bahwa dalam konteks *APM*, gagasan kinerja mencakup seperangkat properti non-fungsional yang komprehensif (misalnya, ketepatan waktu, penggunaan sumber daya, keandalan, ketersediaan, dan keamanan) dan bahkan mungkin aspek fungsional [2].

C. *OpenTelemetry*

OpenTelemetry adalah seperangkat *API*, *SDK*, perkakas, dan integrasi yang dirancang untuk pembuatan dan pengelolaan data telemetri seperti *logs*, *metrics*, dan *traces*. Proyek ini menyediakan implementasi *vendor-agnostic* yang dapat dikonfigurasi untuk mengirim data telemetri ke *backend* pilihan seperti *Jaeger* dan *Prometheus*. Data telemetri dibutuhkan untuk memperkuat observabilitas sebuah sistem. Secara tradisional, data telemetri disediakan oleh vendor-vendor komersial maupun *open-source*. Namun, terdapat kurangnya standardisasi dari berbagai vendor tersebut sehingga menyebabkan kurangnya portabilitas data dan kesulitan dalam memelihara instrumentasi. Oleh karena itu, *OpenTelemetry* hadir untuk menyelesaikan masalah tersebut dengan menyediakan satu solusi yang *vendor-agnostic*.
Beberapa hal yang diberikan oleh *OpenTelemetry*:

- Pustaka instrumentasi vendor-agnostik tunggal per bahasa pemrograman dengan dukungan untuk instrumentasi otomatis dan manual.
- Biner kolektor tunggal (*OpenTelemetry Collector*) yang dapat digunakan dalam berbagai cara termasuk sebagai agen atau *gateway*.
- Implementasi ujung ke ujung untuk menghasilkan, memancarkan, mengumpulkan, memproses, dan mengekspor data telemetri.
- Kontrol penuh terhadap data kita dengan kemampuan untuk mengirimkannya ke berbagai tujuan secara paralel melalui konfigurasi.
- Konvensi semantik standar terbuka untuk memastikan pengumpulan data vendor-agnostik.
- Kemampuan untuk mendukung berbagai format propagasi konteks secara paralel untuk membantu migrasi seiring berkembangnya standar
- Dengan dukungan terhadap beragam protokol komersial maupun *open-source* membuat *OpenTelemetry* mudah untuk diadopsi [5].

D. *Prometheus*

Prometheus adalah sebuah alat *monitoring* dan *alerting* bersifat *open source* yang pada awalnya dibangun untuk *SoundCloud*. Sejak dimulai pada tahun 2012, banyak perusahaan dan organisasi telah mengadopsi *Prometheus*, dan proyek ini memiliki komunitas pengembang dan pengguna yang sangat aktif. Sekarang *Prometheus* merupakan proyek *open-source* mandiri dan dikelola secara independen dari perusahaan mana pun. Untuk menekankan hal ini, dan untuk memperjelas struktur tata kelola proyek, *Prometheus* bergabung dengan *Cloud Native Computing Foundation* pada tahun 2016 sebagai proyek yang di-*hosting* kedua setelah *Kubernetes*.

Prometheus mengumpulkan dan menyimpan metriknya sebagai data deret waktu (*time series data*), yaitu informasi metrik disimpan dengan stempel waktu saat direkam, bersama pasangan nilai kunci opsional yang disebut label. Fitur-fitur utama *Prometheus* adalah:

- Sebuah data model multidimensi dengan data deret waktu (*time series data*) yang diidentifikasi menggunakan nama metrik dan pasangan *key/value*.
- *PromQL*, bahasa *query* yang fleksibel untuk memanfaatkan data model multidimensi tersebut.
- Tidak bergantung pada penyimpanan terdistribusi, *node server* tunggal bersifat otonom.
- Pengumpulan data *time series* dilakukan dengan model *pull* melalui protokol *HTTP*.
- Dukungan model *push* untuk pengumpulan *data time series* melalui *intermediary gateway*.
- Target yang akan diambil datanya diatur melalui konfigurasi statis atau menggunakan *service discovery*.
- Dukungan berbagai mode grafik dan *dashboard* [6].

E. *Jaeger*

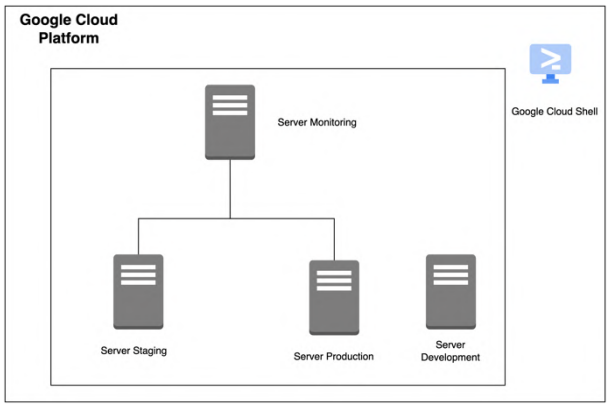
Jaeger yang merupakan *backend* penelusuran (*traces*) terdistribusi yang bersifat *open-source*. *Jaeger* dapat digunakan untuk memantau dan melakukan *troubleshooting* pada sistem terdistribusi. *Jaeger* ini juga mendukung data telemetri *traces* yang dihasilkan oleh *OpenTelemetry API* [7].

F. *Grafana Stack*

Grafana adalah *stack* observabilitas lengkap yang memungkinkan untuk memantau dan menganalisis *metrics*, *logs*, dan *traces*. *Grafana* memungkinkan untuk melakukan *query*, memvisualisasikan, membuat peringatan, dan memahami data telemetri di manapun data tersebut disimpan. *Grafana* mendukung berbagai sumber data seperti *Prometheus*, *Graphite*, *InfluxDB*, *ElasticSearch*, *MySQL*, *PostreSQL*, *dll* [8].

III. METODOLOGI

A. Perancangan *Server*



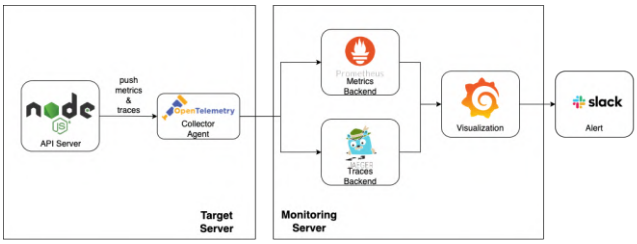
Gambar 1. Perancangan *Server*

Pada Gambar 1 merupakan perancangan *server* yang digunakan untuk lingkungan pengujian pada penelitian ini. Terdapat empat buah *Virtual Machine* dengan sistem operasi *Ubuntu 20.04* yang terpasang pada *Google Cloud Platform*. Masing-masing *VM* tersebut adalah satu *VM* sebagai *server monitoring*, dua *VM* sebagai *server staging* dan *production* yang akan menjadi target *monitoring*, dan satu *VM* sebagai *server development* yang digunakan untuk memasang aplikasi tanpa implementasi *OpenTelemetry* di dalamnya.

B. Perancangan Sistem Monitoring

Sistem *monitoring* terdiri dari dua komponen yaitu *Server Monitoring* dan Target *Monitoring*. Untuk target *monitoring* sendiri terdapat dua *server* yaitu *server staging* dan *server production*. Pada masing-masing target server terdapat beberapa *backend* sistem donasi *online* berbasis *REST API* yang dibangun menggunakan *NodeJS* yang sudah diintegrasikan dengan *OpenTelemetry API*. Selain itu, pada masing-masing target *server* juga terdapat *OpenTelemetry Collector* sebagai agen yang akan mengumpukan *metrics* dan *traces* kemudian mengirimkannya ke *Prometheus* dan *Jaeger*.

Kemudian pada *server monitoring* terpasang *server Prometheus Server* yang bertugas untuk menyimpan dan mengolah data telemetri berupa *metrics*, *Jaeger Server* yang bertugas untuk menyimpan dan mengolah data telemetri berupa *traces*, dan *Grafana Server* yang bertugas untuk visualisasi data. Terakhir terdapat *Slack* yang menjadi saluran peringatan ketika sistem *monitoring* mendeteksi adanya anomali pada data-data telemetri.



Gambar 2. Rancangan Sistem *Monitoring*

C. Skenario Pengujian

Pengujian yang digunakan dalam penelitian ini terdapat 4 metode. Metode yang digunakan adalah pengujian fungsionalitas *monitoring* dan metrics, fungsi *monitoring error rate*, pendeteksian anomali, serta pengujian performa.

1) Pengujian fungsionalitas *monitoring* data *metrics*

a. Pengujian fungsi monitoring *throughput* dan *request latency*

Pengujian ini bertujuan untuk menguji apakah sistem *monitoring* dapat mengumpulkan data *metric* berupa *throughput* dan *request latency* dan mengukur *delay* pengirimannya. Pada pengujian ini dilakukan total 10 kali percobaan *load test* terhadap *server REST API* pada *endpoint GET /v1/campaigns*. Percobaan dilakukan pada *server staging* dan *production* di mana setiap *server* mendapat bagian 5 percobaan. *Load test* dilakukan menggunakan alat bantu *Vegeta* yang dijalankan melalui *Google Cloud Shell* untuk memberikan simulasi pemanggilaan *endpoint* tersebut selama 3 menit dengan *request rate* sebesar 10 *rps*, 15 *rps*, 20 *rps*, 25 *rps*, dan 30 *rps*.

Perintah untuk menjalankan *load test* adalah sebagai berikut:

- *Production*
  echo "GET http://34.128.121.13:5000/v1/campaigns" | vegeta attack -duration=3m -rate=10 | vegeta report
- *Staging*
  echo "GET http://34.128.100.99:5000/v1/campaigns" | vegeta attack -duration=3m -rate=10 | vegeta report

b. Pengujian fungsi *monitoring error rate*

Pada pengujian ini dilakukan empat kali *load test* pada *server REST API staging* dan *production*. *Load test* dilakukan dengan cara melakukan pemanggilan *endpoint GET /v1/campaigns* untuk simulasi *request* berhasil sebanyak 10 *rps* selama 5 menit. Kemudian melakukan pemanggilan

endpoint GET /v1/error untuk simulasi *request* gagal karena *Internal Server Error* sebanyak 5 *rps* selama 5 menit.

2) Pengujian fungsionalitas *monitoring data traces*

Pengujian ini dilakukan dengan cara mematikan *server database* pada *server production* dan *staging* kemudian dilakukan uji coba *login* menggunakan *Postman* ke *REST API production* dan *staging*. Dikarenakan *server database* mati sehingga pengguna yang melakukan *login* akan mendapatkan respons *Internal Server Error*. Pengujian dikatakan berhasil jika sistem *monitoring* dapat menunjukkan bahwa penyebab pengguna tidak bisa melakukan *login* adalah aplikasi gagal terhubung ke *server database*.

3) Pengujian Deteksi Anomali

a. Error Rate

Pengujian ini bertujuan untuk menguji kemampuan sistem dalam mendeteksi *error rate* lebih dari 80%. Nilai *error rate* merupakan perbandingan dari jumlah *request* yang mendapakan *response* 500 dibandingkan dengan total seluruh *request*. Pada pengujian ini dilakukan pemanggilan sebanyak 10 *rps* ke *endpoint GET /v1/campaigns* pada *REST API production* dan *staging* selama 10 menit. Setelah 2 menit berjalan maka *server database* pada masing-masing *environment* akan dimatikan sehingga akan mendapatkan *response* 500 *Internal Server Error*. Hal ini menyebabkan *error rate* menjadi semakin meningkat. Pengujian dikatakan berhasil apabila sistem dapat mengirimkan notifikasi peringatan menuju *Slack* ketika *error rate* bernilai lebih dari 80% selama minimal 30 detik.

b. Throughput

Pengujian ini bertujuan untuk menguji kemampuan sistem dalam mendeteksi adanya penurunan rata-rata *throughput* dalam satu menit terakhir lebih dari 50% dibandingkan dengan rata-rata *throughput* 5 menit kebelakang. Pada pengujian ini dilakukan *load test* ke *endpoint GET /v1/campaigns* pada *REST API production* dan *staging* dengan ketentuan sebagai berikut:

- 10 *rps* selama 5 menit
- 4 *rps* selama 2 menit
- 10 *rps* selama 5 menit

Pengujian dikatakan berhasil apabila sistem dapat mengirimkan notifikasi peringatan menuju *Slack* ketika *throughput* turun lebih dari 50% yang disebabkan oleh perubahan jumlah *rps* dari 10 *rps* menjadi 4 *rps*. Selain itu, sistem juga harus mampu memberikan pemberitahuan apabila jumlah *rps* kembali normal yang disebabkan oleh jumlah rps kembali menjadi 10 *rps* selama 5 menit.

c. Request Latency

Pada pengujian ini dilakukan pemanggilan sebanyak 20 *rps* ke *endpoint GET /v1/campaigns* pada *REST API production* dan *staging* selama 2 menit. Pengujian dikatakan berhasil apabila sistem dapat mengirimkan notifikasi peringatan menuju Slack ketika *request latency* bernilai lebih dari 80 milidetik.

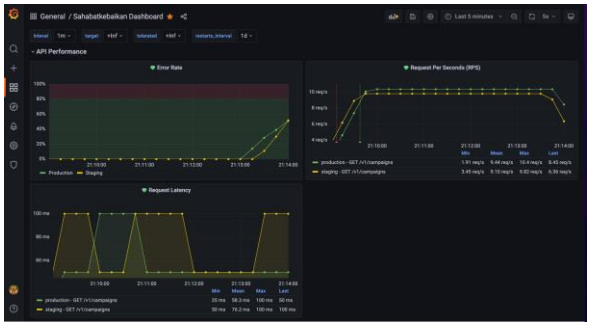
4) Pengujian Performa Aplikasi

Pengujian ini bertujuan untuk mengetahui bagaimana pengaruh implementasi monitoring *metrics* dan *traces* menggunakan *OpenTelemetry* terhadap performa dari aplikasi itu sendiri. Pada topologi akan ditambahkan satu buah *development server* yang berisi *backend* sistem donasi berbasis *REST API* tanpa implementasi *OpenTelemetry* di dalamnya. Pengujian *load test* dilakukan sebanyak 5 kali untuk *server production* dan *development*. Pengujian pertama dilakukan dengan melakukan pemanggilan ke *endpoint GET /v1/campaigns* menggunakan satu buah *user* (*thread*), pengujian kedua menggunakan dua buah *user* (*thread*), dan seterusnya hingga pengujian kelima menggunakan lima buah *user* (*thread*). Setiap pengujian akan berlangsung selama 5 menit.

IV. HASIL DAN PEMBAHASAN

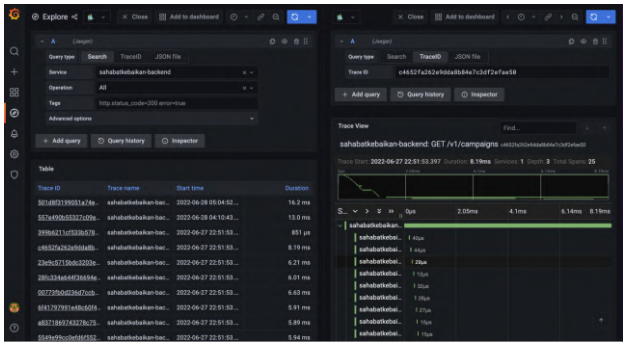
A. Tampilan Antar Muka

Berikut ini adalah tampilan antarmuka dashboard *Grafana* sistem *monitoring* yang telah dibuat. Dalam *dashboard* ini terdapat panel grafik *Error Rate* untuk memantau persentase jumlah *request error* per *endpoint*, panel *Request Per Second (RPS)* untuk memantau jumlah *request* per *endpoint* setiap detiknya, dan panel *Request Latency* untuk memantau *latency* setiap *request* per *endpoint*.



Gambar 3. Tampilan *dashboard Grafana*

Kemudian berikut ini adalah tampilan antarmuka *Jaeger* yang diintegrasikan pada *Grafana* melalui menu *Explore*. Menu ini dapat digunakan untuk membaca data-data *traces* yang telah dikumpulkan.



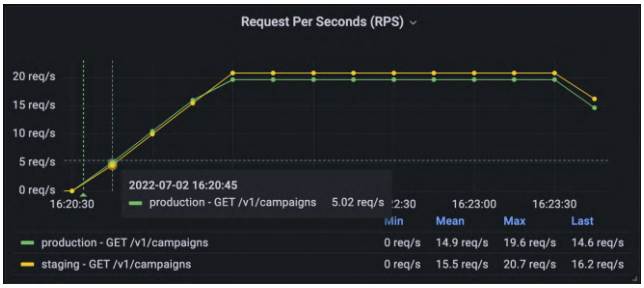
Gambar 4. Tampilan *Jaeger* di *Grafana*

B. Hasil Pengujian Sistem

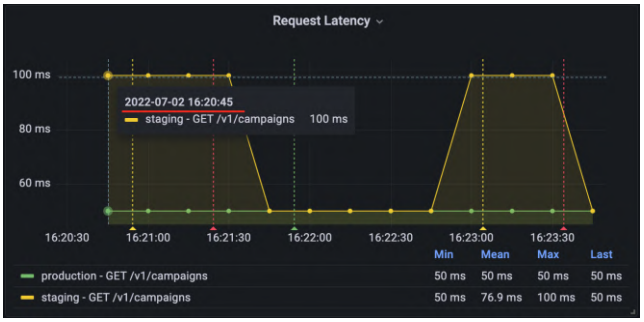
1) Pengujian *Monitoring* Data Metrics

a. Pengujian *Monitoring Throughput* dan *Request Latency*

Selama proses *load test* berlangsung, dilakukan pemantuan pada *dashboard* Grafana untuk panel *Request Per Seconds (RPS)* dan *Request Latency*. Melalui panel ini juga dilihat kapan *metrics* pertama kali diterima untuk masing-masing pengujian. Gambar 3 dan Gambar 4 adalah contoh tampilan dari pemantauan melalui *dashboard Grafana*.



Gambar 5. Contoh tampilan pemantauan panel *Request Per Seconds (RPS)*



Gambar 6. Contoh pemantauan panel *Request Latency*

Dari hasil pengujian *load test* dan pemantauan pada *dashboard Grafana* serta memperhatikan laporan *load test* dari *Vegeta*, diperoleh data sebagai berikut:

Tabel 1. Hasil pengujian *monitoring* data *metrics*

No	*Env*	Rate (rps)	Waktu Mulai *Test*	Waktu Metric Terbaca	Delay (s)
1	*Staging*	10	02/07/2022 15:52:51	02/07/2022 15:53:15	24
2	*Production*	10	02/07/2022 15:52:52	02/07/2022 15:53:00	8
3	*Staging*	15	02/07/2022 16:06:23	02/07/2022 16:06:30	7
4	*Production*	15	02/07/2022 16:06:23	02/07/2022 16:06:30	7
5	*Staging*	20	02/07/2022 16:20:27	02/07/2022 16:20:45	18
6	*Production*	20	02/07/2022 16:20:27	02/07/2022 16:20:45	18
7	*Staging*	25	02/07/2022 16:38:25	02/07/2022 16:38:45	20
8	*Production*	25	02/07/2022 16:38:25	02/07/2022 16:38:45	20
9	*Staging*	30	02/07/2022 16:45:37	02/07/2022 16:45:45	8
10	*Production*	30	02/07/2022 16:45:38	02/07/2022 16:45:45	8
			Rata-rata		13,8

Dari Tabel 1 hasil pengujian *throughput* di atas dapat dilihat bahwa sistem *monitoring* berhasil mengumpulkan data *metrics* berupa *throughput* dan *request latency* dan menampilkannya ke *dashboard Grafana* secara *realtime* dengan *delay* rata-rata sebesar 13,8 detik.

b. Pengujian *Monitoring Error Rate*

Selama proses *load test* berlangsung, dilakukan pemantuan pada dashboard *Grafana* untuk panel *Error Rate*. Berikut ini adalah hasil dari pemantauan melalui *dashboard Grafana*.

Gambar 7. Hasil pemantauan panel *Error Rate*

Dari hasil pengujian *load test* dan pemantauan pada *dashboard Grafana*, diperoleh data sebagai berikut:

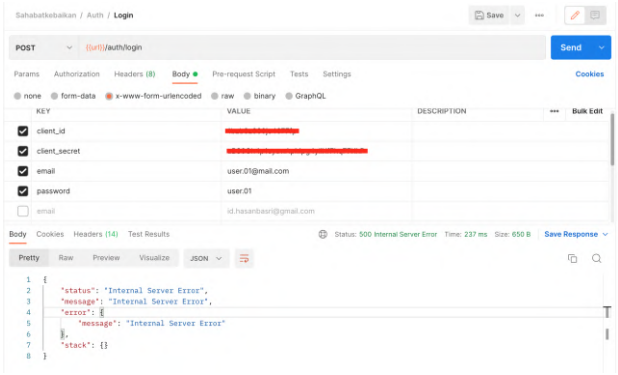
Tabel 2. Hasil pengujian *Error Rate*

No	*Env*	*Endpoint*	Durasi	*Rate*	Status Perubahan
1	*Staging*	GET /v1/campaigns	5 menit	10 *rps*	Berhasil terbaca
	Staging	GET /v1/error	5 menit	5 *rps*	
2	*Production*	GET /v1/campaigns	5 menit	10 *rps*	Berhasil terbaca
	Production	GET /v1/error	5 menit	5 *rps*	

Dari hasil tersebut dapat disimpulkan bahwa sistem *monitoring* dapat mengumpulkan data *metrics* berupa *error rate* dan mampu menampilkannya secara *realtime* pada *dashboard Grafana*.

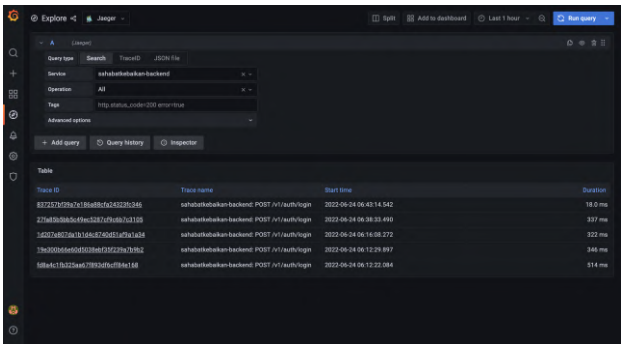
c. Pengujian *Monitoring* Data *Traces*

Pengguna melakukan *request* ke *endpoint POST /auth/login* dan mendapatkan respons *500 Internal Server Error* seperti pada gambar di bawah ini.



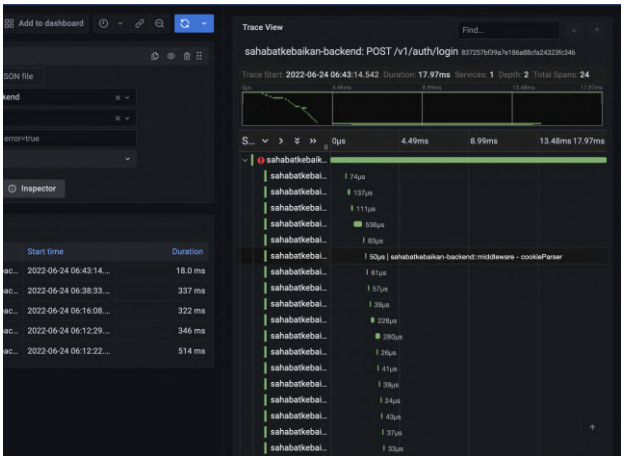
Gambar 8. Pemanggilan *Login* pada *API production*

Untuk mengetahui penyebab dari *error* tersebut adalah dengan mengecek *traces request* melalui menu *Explore Grafana* dan memilih *datasource Jaeger* seperti ditunjukkan pada gambar berikut.



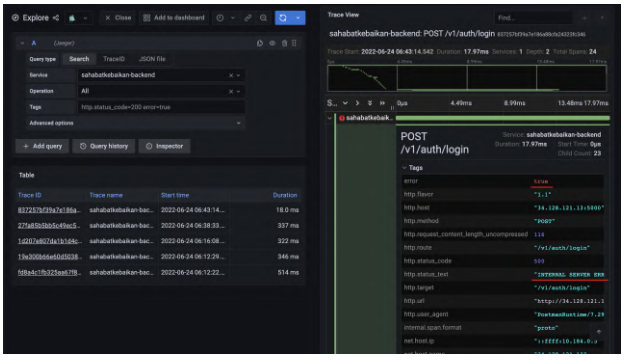
Gambar 9. Melihat daftar traces pada pengujian API *production*

Pada Gambar 9 menampilkan daftar *traces request* yang terjadi dalam satu jam terakhir. Kemudian dipilih *trace* yang memiliki *start time* sama dengan saat pelakukan pengujian yaitu *trace* yang paling terakhir dengan *Id 837257bf39a7e186a88cfa24323fc346*. Ketika diklik pada pada *trace* tersebut maka muncul detail *trace* berupa *spans* yang menunjukkan *trace* tersebut melewati fungsi apa saja seperti yang ditunjukkan pada Gambar 10 di bawah ini.



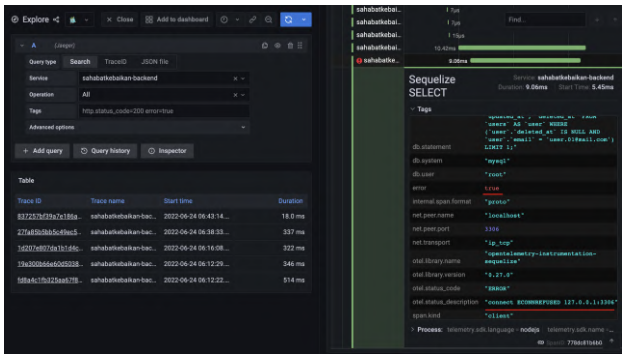
Gambar 10. Melihat detail *trace* pada pengujian *API production*

Gambar di bawah ini merupakan detail dari *span* paling luar pada *request* dengan *trace Id 837257bf39a7e186a88cfa24323fc346*. Di dalamnya terdapat berbagai *tags* yang menjadi informasi penting tentang *trace* ini. *Tag error* bernilai *true* artinya *trace* ini mengembalikan *error*. Kemudian terdapat *tags http.status_code* yang menunjukkan kode *error*-nya yaitu 500, dan *tags http.status_text* yang menunjukkan teks *error*-nya yaitu *Internal Server Error*.



Gambar 11. Melihat *tags* pada detail *trace* pada pengujian *production*

Setelah ditelurusi lebih lanjut pada span-span di dalam *trace* ini, diketahui bahwa penyebab *error* tersebut adalah pada *library Sequelize* yang mengalami kegagalan dalam melakukan *query SELECT*. Kegagalan ini dikarenakan tidak dapat terhubung ke *server database*.



Gambar 12. Melihat penyebab *error* pada *span Sequelize* pada pengujian *production*

Dengan demikian sistem *monitoring* berhasil menemukan sumber masalah penyebab pengguna tidak dapat *login* ke aplikasi *production* yaitu aplikasi gagal terhubung ke *server database*.

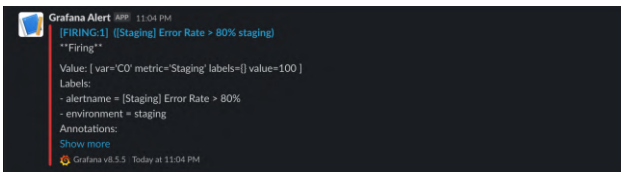
3) Pengujian Deteksi Anomali

a. Deteksi Anomali *Error Rate*

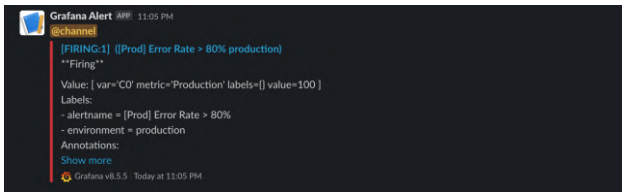
Pada pengujian ini dilakukan pemanggilan sebanyak 10 *rps* ke *endpoint GET /v1/campaigns* pada *REST API production* dan *staging* selama 10 menit. Setelah 2 menit berjalan maka *server database* akan dimatikan sehingga akan mendapatkan *response* 500 *Internal Server Error*. Hal ini menyebabkan *error rate* menjadi semakin meningkat seperti yang ditunjukkan oleh Gambar 13 pada pukul 23:03:00. Ketika *error rate* mencapai lebih dari 80% selama 30 detik maka akan memicu *alert* yang dikirimkan menuju *Slack*.

Gambar 13. Melihat panel *Error Rate* pada pengujian deteksi anomali

Berikut ini adalah tangkapan layar notifikasi yang dikirimkan menuju *Slack* baik untuk *server production* dan *staging*.



Gambar 14. Peringatan anomali *error rate* pada *API staging* terkirim ke *Slack*



Gambar 15. Peringatan anomali *error rate* pada *API production* terkirim ke *Slack*

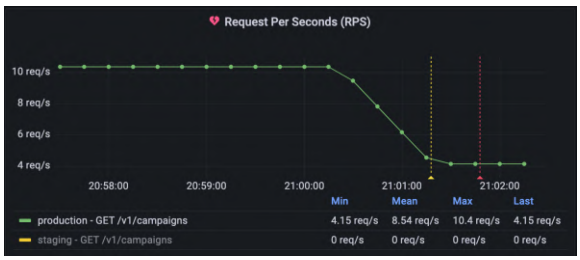
Sekitar 2 menit kemudian *server database* untuk masing-masing *server* dinyalakan kembali. Hal ini membuat pemanggilan *endpoint* mendapatkan respons 200 sehingga *error rate* perlahan menurun seperti yang ditunjukkan oleh Gambar 13 pada pukul 23:05:45. Setelah *error rate* berada di bawah 80 selama 30 detik maka kondisi akan menjadi normal kembali. Kemudian sistem akan mengirimkan notifikasi menuju *Slack* bahwa keadaan sudah kembali normal.

Gambar 16. Pemberitahuan anomali *error rate API staging* sudah kembali normal terkirim ke *Slack*

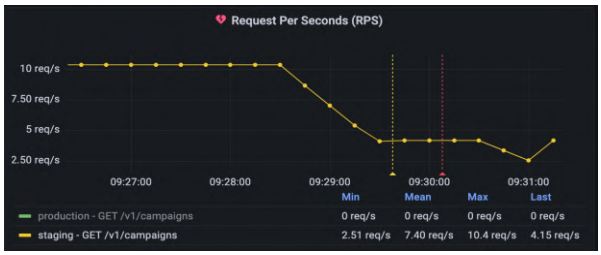
Gambar 17. Pemberitahuan anomali *error rate API production* sudah kembali normal terkirim ke *Slack*

b. Deteksi Anomali *Throughput*

Pada pengujian ini dilakukan pemanggilan ke *endpoint GET /v1/campaigns* pada *REST API production* dan *staging* sebanyak 10 *rps* selama 5 menit. Setelah itu langsung dilanjutkan dengan pemanggilan sebanyak 4 *rps* selama 2 menit. Hal ini mengakibatkan jumlah *rps* turun sebesar 60% seperti yang ditunjukkan pada gambar berikut ini.

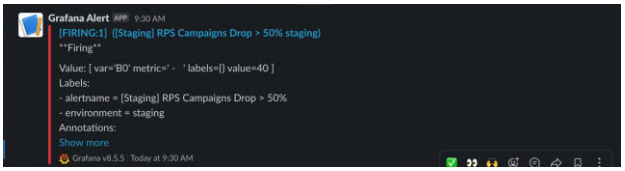


Gambar 18. Melihat panel *Request Per Seconds (RPS)* pada pengujian deteksi anomali *request drop API production*



Gambar 19. Melihat panel *Request Per Seconds (RPS)* pada pengujian deteksi anomali *request drop API staging*

Setelah keadaan ini berlangsung selama 30 detik sistem mengirimkan peringatan menuju *Slack* seperti yang ditunjukkan pada gambar berikut.

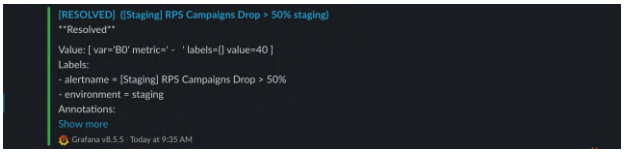


Gambar 20. Peringatan anomali *request drop* pada *API staging* terkirim ke *Slack*

Gambar 21. Peringatan anomali *request drop* pada *API production* terkirim ke *Slack*

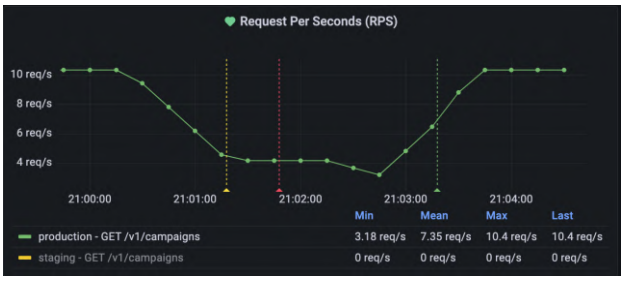
Setelah pemanggilan sebanyak 4 *rps* selama 2 menit kemudian dilanjutkan pemanggilan sebanyak 10 *rps* selama 5 menit. Hal ini menyebabkan keadaan menjadi normal kembali dan sistem berhasil mengirimkan pemberitahuan menuju *Slack*.

Gambar 22. Pemberitahuan anomali *request drop API staging* sudah kembali normal terkirim ke *Slack*

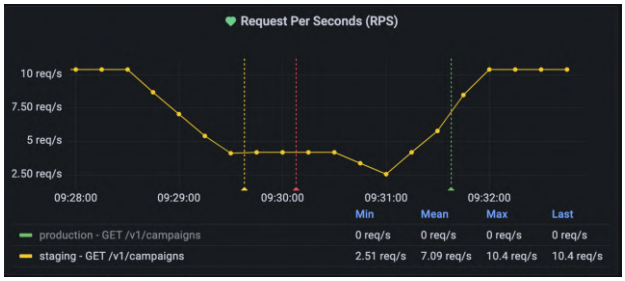


Gambar 23. Pemberitahuan anomali *request drop API staging* sudah kembali normal terkirim ke *Slack*

Ketika dilihat secara keseluruhan dalam panel *Request Per Seconds (RPS)* maka tampilannya seperti gambar berikut.



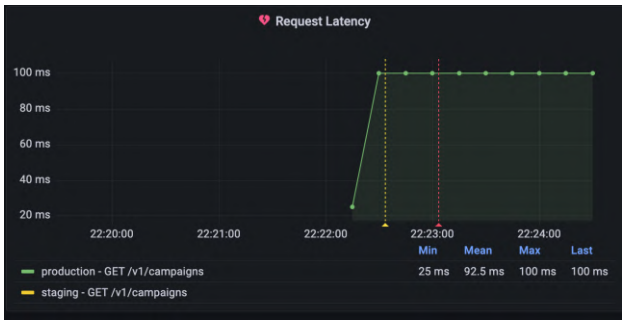
Gambar 24. Tampilan grafik keseluruhan pada pengujian *RPS Drop production*



Gambar 25. Tampilan grafik keseluruhan pada pengujian *RPS Drop staging*

c. *Request Latency*

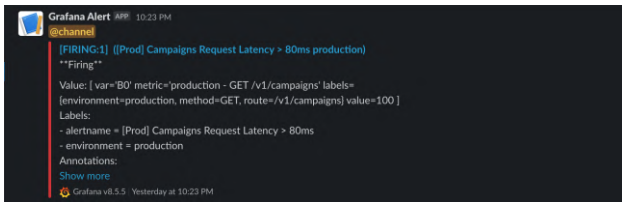
Pada pengujian ini dilakukan pemanggilan sebanyak 20 *rps* ke *endpoint GET /v1/campaigns* pada *REST API production* dan *staging* selama 2 menit. *Request latency* rata-rata sebesar 100 milidetik baik pada *server production* dan *staging* seperti yang ditunjukkan pada grafik berikut ini.



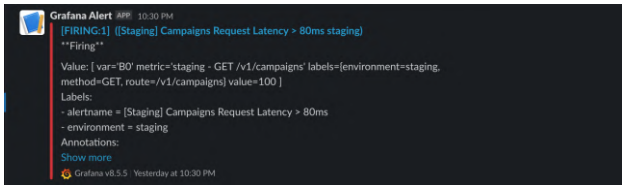
Gambar 26. Melihat panel *Request Latency* pada pengujian deteksi anomali latensi > 80 *ms* pada *API production*

Gambar 27. Melihat panel *Request Latency* pada pengujian deteksi anomali latensi > 80 *ms* pada *API staging*

Sistem *monitoring* berhasil memberikan peringatan melalui *Slack* seperti yang ditunjukkan pada gambar berikut.



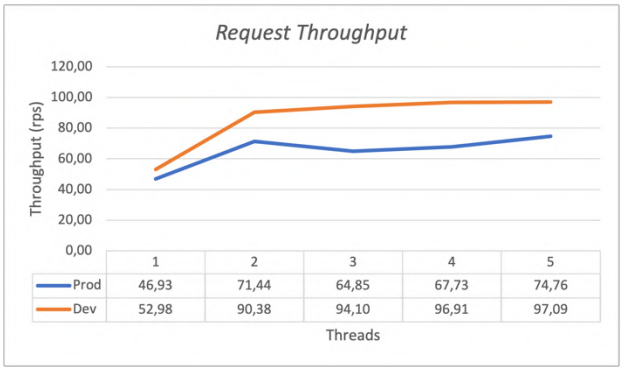
Gambar 28. Peringatan anomali latensi >80 *ms* pada *API production* terkirim ke *Slack*



Gambar 29. Peringatan anomali latensi >80 *ms* pada *API staging* terkirim ke *Slack*

4) Pengujian Performa Aplikasi

Gambar 30. menunjukkan *request throughput* dari total 10 pengujian yang dilakukan pada *server* aplikasi *production* dan *development*.



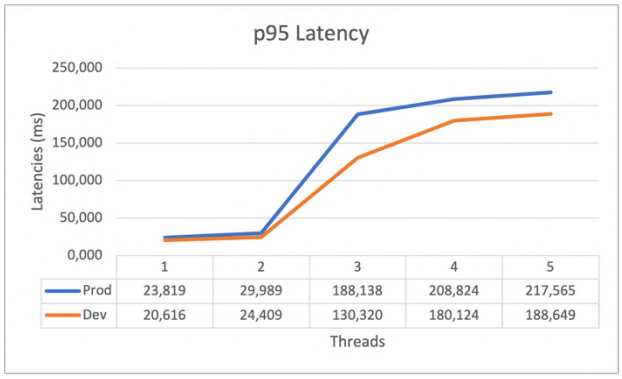
Gambar 30. Hasil *throughput* pada uji performa aplikasi

Seperti yang terlihat pada Gambar 30 implementasi *OpenTelemetry* untuk *monitoring metrics* dan *traces* pada *server production* memberikan dampak negatif pada performa aplikasi. Dari 5 kali pengujian dengan setiap pengujian menggunakan jumlah *virtual user (threads)* yang berbeda semuanya menunjukkan jumlah *request throughput* pada *server production* lebih sedikit daripada *server development*. Berikut ini adalah presentase penurunan jumlah *request throughput server production* dibandingkan dengan *server development*.

Tabel 3. Penurunan *request throughput* pada *server production*

Thread	**Penurunan (%)**
1	11,42
2	20,96
3	31,08
4	30,11
5	23,00
Rata-rata	23,31

Gambar 31 menunjukkan hasil *request latency* dari total 10 pengujian yang dilakukan pada *server* aplikasi *production* dan *development*.



Gambar 31. Hasil *Request Latency* pada uji performa aplikasi

Dari gambar tersebut diketahui bahwa latensi pada *server production* selalu lebih tinggi dibandingkan dengan *server development*. Hal ini menunjukkan bahwa implementasi *OpenTelemetry* untuk *monitoring metrics* dan *traces* pada *server production* juga memberikan pengaruh terhadap performa aplikasi itu sendiri dalam hal *request latency*. Berikut ini adalah presentase kenaikan latensi *server production* dibandingkan dengan *server development*.

Tabel 4. Kenaikan latensi pada *server production*

Thread	**Kenaikan (%)**
1	15,54
2	22,86
3	44,37
4	15,93
5	15,33
Rata-rata	22,80

Dari data yang ditunjukkan oleh Gambar 30, Gambar 31, Tabel 3, dan Tabel 4 dapat diambil kesimpulan bahwa implementasi *OpenTelemetry* untuk *monitoring metrics* dan *traces* memberikan pengaruh negatif pada performa aplikasi

yaitu dapat menurunkan jumlah *request throughput* dengan penurunan rata-rata 23,32% dan menaikkan *request latency* dengan kenaikan rata-rata 22,80%. Hal ini dapat diterima karena dengan adanya implementasi *OpenTelemetry* pada aplikasi artinya terdapat tambahan pekerjaan yang berjalan di setiap *request* yang terjadi yaitu pekerjaan untuk mengirimkan data *metrics* dan *traces* menuju ke *server monitoring*. Perlu adanya penelitian lebih lanjut untuk melakukan optimasi pengiriman agar penurunan performa yang disebabkan oleh implementasi *OpenTelemetry* menjadi semakin kecil.

V. SIMPULAN

Berdasarkan hasil penelitian yang diperoleh dari penelitian "Perancangan Sistem *Monitoring* Performa Aplikasi Menggunakan *OpenTelemetry* dan *Grafana Stack*", dapat ditarik kesimpulan sebagai berikut:

- *OpenTelemetry* dapat digunakan untuk menginstrumentasi dan mengumpulkan data telemetri berupa *metrics* dan *traces* yang dapat diproses dan ditampilkan melalui *dashboard Grafana*.
- Sistem *monitoring* mampu memantau *data metrics* dua aplikasi berbasis *REST API* secara *realtime* dengan *delay* rata-rata 13,8 detik.
- Melalui *Grafana Alert Manager*, sistem *monitoring* mampu memberikan peringatan melalui *Slack* ketika terjadi anomali pada data *metrics* yang dikumpulkan.
- Sistem *monitoring* dapat digunakan sebagai alat *debugging* yang efektif untuk menemukan akar masalah yang terjadi pada aplikasi berbasis *REST API* dengan menggunakan data *traces* yang disimpan pada *Jaeger*.
- Implementasi *OpenTelemetry* dalam aplikasi *backend* berbasis *REST API* untuk memonitor *metrics* dan *traces* memberikan dampak negatif pada performa aplikasi itu sendiri yaitu dapat menurunkan jumlah *request throughput* dengan penurunan rata-rata 23,32% dan menaikkan *request latency* dengan kenaikan rata-rata 22,80%.

REFERENSI

[1] Y. Einav. "Amazon Found Every 100ms of Latency Cost them 1% in Sales". [Online]. Available: https://www.gigaspaces.com/blog/amazon-found-every-100ms-of-latency-cost-them-1-in-sales (accessed January 31, 2022).

[2] A. Van Hoorn and S. Siegl. "Application Performance Management: Measuring and Optimizing the Digital Customer Experience". *SIGS DATACOM GmbH. Troisdrof*. 2019.

[3] G. Kim, J. Humble, P. Debois and J. Willis. *The Devops Handbook How to Create World Class Agility, Reliability, and Security in Technology Organizations*. 2nd ed. Page 360. 2021

[4] J.J. Tang. "Observability Vs. Monitoring: What's the Difference?". [Online]. Available: https://devops.com/observability-vs-monitoring-whats-the-difference/ (accessed January 31, 2022).

[5] OpenTelemetry. [Online]. Available: https://opentelemetry.io/docs/concepts(accessed January 31, 2022).

[6] Prometheus. [Online]. Available: https://prometheus.io/docs/introduction/overview/(accessed January 31, 2022).

[7] Jaeger Overview. [Online]. Available: https://www.jaegertracing.io/docs/1.35/ (accessed January 31, 2022).

[8] Grafana. [Online]. Available: https://grafana.com/docs/grafana/latest/basics/ (accessed January 31, 2022).

Implementasi Arsitektur *Serverless Internet of Things* pada *Monitoring Cold Chain*

Aisyah Mulyani[1], Unan Yusmaniar Oktiawati [1,*]
[1]Departemen Teknik Elektro dan Informatika, Sekolah Vokasi, Universitas Gadjah Mada;
aisyahmulyani@mail.ugm.ac.id
*Korespondensi: unan_yusmaniar@ugm.ac.id

Abstract – With the current technological developments, the need for monitoring in cold chain logistics as well as the problem of limitations on IoT devices such as data storage and computing capabilities can be handled with the help of cloud technology. Cloud has a concept called serverless. In this study, researchers tried to examine serverless services that can be integrated with IoT devices. The IoT devices used consist of nodes and gateways. In this final project research, testing and analysis of gateway devices and serverless architecture are carried out. NodeMCU which is part of the gateway device and AWS IoT as a serverless service will be integrated in this research. The serverless architecture system that was built also has a function to store data and send notifications to the user. Meanwhile, data monitoring will be carried out through Grafana. The results of AWS IoT performance testing when integrated with IoT gateway devices through QoS testing according to the TIPHON standard are categorized as satisfactory and the implemented system functions as expected.

Keywords : Internet of Things, Cloud Computing, Serverless, Amazon Web Services

Intisari – Dengan adanya perkembangan teknologi saat ini, kebutuhan *monitoring* pada *cold chain logistic* serta masalah keterbatasan pada perangkat IoT seperti kemampuan penyimpanan data dan komputasi dapat ditangani dengan bantuan teknologi *cloud*. *Cloud* memiliki konsep yang dinamakan dengan *serverless*. Pada penelitian ini peneliti mencoba untuk meneliti layanan *serverless* yang dapat diintegrasikan dengan perangkat IoT. Perangkat IoT yang digunakan terdiri dari node dan gateway. Pada penelitian proyek akhir ini dilakukan pengujian dan analis dari perangkat gateway dan arsitektur *serverless*. NodeMCU yang merupakan bagian dari perangkat gateway dan AWS IoT sebagai layanan *serverless* akan diintegrasikan pada penelitian ini. Sistem arsitektur *serverless* yang dibangun juga memiliki fungsi untuk menyimpan data dan mengirimkan notifikasi kepada user. Sedangkan *monitoring* data akan dilakukan melalui Grafana. Hasil pengujian performa AWS IoT saat diintegrasikan dengan perangkat IoT gateway melalui pengujian QoS menurut standar TIPHON berkategorikan memuaskan dan sistem yang telah diimplementasikan berfungsi sesuai yang diharapkan.

Kata kunci : ***Internet of Things, Cloud Computing, Serverless, Amazon Web Services***

I. PENDAHULUAN

Pertumbuhan penduduk yang meningkat menyebabkan meningkatkan kebutuhan makanan. Salah satu produk makanan yang membutuhkan jaminan mutu yang baik adalah produk *perishable* yaitu produk yang rentan busuk atau rusak seperti buah – buahan, sayuran, daging dan ikan. Dalam pendistribusian produk *perishable* atau yang biasa disebut dengan *cold chain* logistic, terdapat dua kegiatan yaitu kegiatan logistik dan pengendalian serta pemantauan suhu. Suhu dari produk *perishable* perlu dijaga dengan baik agar kualitas makanan tetap terjamin. Salah satu cara untuk memastikan suhu tetap rendah adalah dengan melakukan *monitoring* suhu.

Pada penelitian ini dibuat sebuah arsitektur *cloud* yang berfungsi untuk diintegrasikan dengan perangkat IoT sehingga dapat melakukan pemantauan atau *monitoring* suhu secara *real-time*. Perangkat tersebut terdiri dari node dan gateway. Node akan berfungsi sebagai alat pengukur suhu yang diletakkan disetiap *box* pengiriman sedangkan gateway berfungsi sebagai router yang akan mendistribusikan data suhu ke *cloud* untuk dapat di*monitoring* oleh pengguna dimanapun berada. Oleh karena itu pembuatan arsitektur *cloud* untuk dapat menerima, menyimpan dan serta mengirimkan notifikasi mengenai kondisi tertentu pada *box* pengiriman sangat dibutuhkan.

Dalam mengintegrasikan perangkat IoT dan *cloud*, penelitian ini akan memanfaatkan layanan arsitektur *serverless* pada Amazon Web Services yaitu AWS IoT. AWS IoT akan berperan sebagai penghubung antara perangkat dan arsitektur *cloud*. AWS IoT akan menerima data dan melanjutkan data tersebut untuk disimpan maupun diolah untuk fungsi lainnya seperti notifikasi dan pemantauan.

Selain AWS IoT, layanan *serverless* yang akan digunakan pada penelitian ini adalah AWS Lambda. Fungsi dari Lambda pada penelitian ini adalah melakukan *query* ke *database* setiap beberapa menit lalu mengirimkan notifikasi melalui AWS SNS (*Simple Notification Service*) ketika terjadi kondisi tidak normal pada suhu.

Berdasarkan penjelasan di atas, penulis akan mencoba untuk melakukan penelitian mengenai performa AWS IoT sebagai layanan *serverless* yang di integrasikan dengan perangkat IoT gateway serta bagaimana layanan *serverless* tersebut dapat menerima, menyimpan dan menampilkan data secara real-time serta mengirimkan notifikasi ke pengguna. Serta untuk memudahkan pemantauan suhu, pada penelitian ini akan mengimplementasikan penggunaan Grafana untuk memvisualisasikan data yang telah disimpan pada *database* di AWS Timestream.

II. TEORI PENDUKUNG

A. *Cold chain*

Cold chain adalah rantai pengiriman yang terdiri dari serangkaian aktivitas penyimpanan dan distribusi yang tidak terputus dengan mempertahankan kisaran suhu yang ditentukan. Salah satu cara untuk memastikan pendistribusian produk *perishable* pada *cold chain logistic* berjalan dengan baik, dilakukan pemantauan besaran suhu. Pemantauan besaran suhu dengan menggunakan cara manual

adalah dengan melakukan pengecekan ke dalam *tiap cold storage* secara berkala. Kekurangannya adalah menghabiskan banyak waktu terutama apabila terdapat banyak *cold storage* dalam satu pengiriman. Oleh karena itu diperlukan sebuah penerapan teknologi yang mampu membangun sebuah sistem *monitoring cold chain* yang lebih efisien dan real-time, yakni pemanfaatan *Internet of Things* (IoT).

B. *Internet of Things*

Internet of Things (IoT) adalah sebuah konsep di mana suatu objek yang memiliki kemampuan untuk mentransfer data melalui jaringan tanpa memerlukan adanya interaksi dari manusia ke manusia atau dari manusia ke komputer. Internet of Things (IoT) adalah struktur di mana objek, orang disediakan dengan identitas eksklusif dan kemampuan untuk pindah data melalui jaringan tanpa memerlukan dua arah antara manusia ke manusia yaitu sumber ke tujuan atau interaksi manusia ke komputer [1].

Salah satu pemanfaaatan IoT yang berhubungan dengan sistem monitoring *cold chain* telah dilakukan oleh [2] mengenai "*On the Application of Internet of Things (IOT) in Cold chain Logistics Management*". Penelitian tersebut menjelaskan mengenai manfaat penerapan IoT dalam *cold chain logistics* seperti meminimalisir *error*, melindungi asset dan kargo, serta menjamin kualitas dari produk *perishable*.

C. *Serverless*

Severless Architecture atau arsitektur tanpa *server* merupakan langkah untuk membangun dan menjalankan berbagai aplikasi dan layanan tanpa perlu mengelola infrastruktur. *Server* masih menjalankan suatu aplikasi, tetapi semua manajemen *server* dilakukan oleh penyedia layanan. Sehingga tidak lagi perlu menyediakan atau mengelola *server* untuk menjalankan aplikasi, *database*, dan sistem penyimpanan. Dengan komputasi tanpa *server*, tugas manajemen infrastruktur seperti penyediaan kapasitas dan patching ditangani oleh penyedia layanan, sehingga hanya dapat fokus pada penulisan kode yang melayani pelanggan. Dengan demikian hal ini dapat mengurangi biaya pengeluaran tambahan dan juga dapat menghemat waktu dan energi bagi *developer* [3].

Arsitektur *serverless* bukan menandakan bahwa pada arsitektur tersebut pengguna tidak menggunakan *server* sama sekali. Yang dimaksud dengan *serverless* dalam hal ini adalah dapat berkurangnya interaksi pengguna dalam melakukan *managing* atau *deploying server*. Apabila sebelumnya pengguna perlu melakukan SSH kedalam untuk mengakses *server* maka dalam konsep *serverless* pengguna dapat melakukan hal tersebut melalui konsol web. Selain memiliki sifat minimal *deployment* atau *simple*, arsitektur *serverless* memiliki beberapa sifat atau kelebihan seperti *auto-scaling, cost saving dan time saving.*

D. AWS IoT

AWS IoT adalah layanan *cloud* terkelola yang memungkinkan perangkat yang terhubung dengan mudah dan aman berinteraksi dengan aplikasi *cloud* dan perangkat lain. AWS IoT mendukung protokol HTTP, WebSockets, dan MQTT, protokol komunikasi ringan yang dirancang khusus untuk mentolerir koneksi terputus-putus, meminimalkan jejak kode pada perangkat, dan mengurangi kebutuhan bandwidth jaringan. Karena mendukung protokol MQTT, AWS IoT dapat berperan sebagai broker dan melakukan subscribe dan publish. AWS IoT juga dapat dikolaborasikan dengan berbagai layanan lainnya pada AWS seperti SNS, *database*, IoT Analytics dan lain – lain. Selain itu AWS IoT juga menyediakan *Certificate Authority* (CA) sehingga hanya perangkat yang yang diberikan otorisasi saja yang dapat terhubung dengan AWS IoT.

E. Protokol Komunikasi MQTT

Protokol komunikasi merupakan seperangkat aturan yang digunakan untuk melakukan komunikasi agar dua belah pihak yang berkomunikasi dapat saling memahami. Protokol komunikasi IoT dibagi menjadi dua yaitu *IoT data protocol* dan *Network protocol for IoT*. Dalam penelitian ini akan dibahas mengenai *IoT data protocol* yaitu protokol yang digunakan perangkat IoT untuk berkomunikasi dengan pengguna melalui internet. Jenis *IoT data protocol* yang digunakan dalam penelitian ini adalah MQTT.

Message Queue Telemetry Transport (MQTT) adalah sebuah protokol komunikasi data *machine to machine* (M2M) yang berada pada layer aplikasi, MQTT bersifat *lightweight message* artinya MQTT berkomunikasi dengan mengirimkan data pesan yang memiliki header berukuran kecil yaitu hanya sebesar 2 bytes untuk setiap jenis data, sehingga dapat bekerja di dalam lingkungan yang terbatas sumber dayanya seperti kecilnya bandwidth dan terbatasnya sumber daya listrik, selain itu protokol ini juga menjamin terkiriminya semua pesan walaupun koneksi terputus sementara, protokol MQTT menggunakan metode publish/subscribe untuk metode komunikasinya [4].

F. Grafana

Grafana adalah perangkat lunak open-source yang digunakan untuk mengolah data menjadi informasi yang ditampilkan dalam format grafik. Grafana menyediakan banyak fitur yang powerful dalam melakukan *monitoring* dan analisa sehingga menjadi terkenal dan banyak digunakan di perusahaan sebagai platform *monitoring*. Grafana dapat dijalankan pada sistem operasi Gnu / Linux, Windows dan MacOS. Umumnya grafana mendapatkan metric atau data yang diolah dengan melakukan *query* ke *database*. Grafana kompatibel dengan banyak *database* opensource maupun berbayar seperti MySQL, Influx DB, AWS Timestream dan lain – lain.

G. QoS

QoS didesain untuk membantu *end user (client)* menjadi lebih produktif dengan memastikan bahwa user mendapatkan performansi yang handal dari aplikasi-aplikasi berbasis jaringan. QoS mengacu pada kemampuan jaringan untuk menyediakan layanan yang lebih baik pada trafik jaringan tertentu melalui teknologi yang berbeda-beda [5].

Terdapat beberapa parameter QoS seperti throughput, *delay*, jitter, *packet* loss dan *packet delivery*. Namun dalam penelitian ini hanya akan digunakan 2 parameter yaitu *packet delivery* dan *delay*. *Packet delivery* diartikan sebagai jumlah kedatangan paket yang berhasil sampai ke penerima, menurut standar TIPHON (*Telecommunications and Internet Protocol Harmonization Over Network*) kategori *packet delivery* dibagi ke dalam empat kategori. Tabel 1 menunjukkan indeks kategori *packet delivery*.

Tabel 1. Kategori *Packet delivery*

Kategori	Presentase	Indeks
Sangat Bagus	100%	4
Bagus	97%	3
Sedang	85%	2
Jelek	75%	1

Sedangkan *delay* merupakan waktu penundaan paket saat melakukan proses pengiriman dari satu titik ke titik lain yang dituju. *Delay* menggunakan satuan milidetik (ms). Kategori *delay* menurut standar TIPHON tertera pada Tabel 2 kategori *delay*.

Tabel 2. Kategori *Delay*

Kategori	Nilai *Delay*	Indeks
Sangat Bagus	<150 ms	4
Bagus	150 s/d 300 ms	3
Sedang	300 s/d 450 ms	2
Jelek	>450 ms	1

Hasil dari perhitungan *packet delivery* dan *delay* akan dikalkulasikan sesuai kategori QoS, seperti terletak pada Tabel 3. Kategori QoS Versi TIPHON.

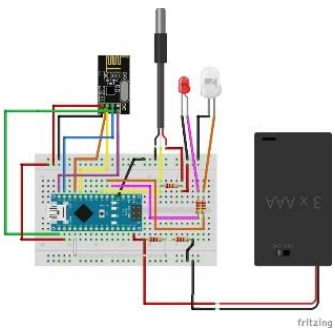
Tabel 3. Kategori QoS

Kategori	Presentase	Indeks
3,8 – 4	95 – 100%	Sangat Memuaskan
3 – 3,75	75 – 95,75%	Memuaskan
2 – 2,99	50 – 74,75%	Kurang Memuaskan
1 – 1,99	25 – 49,75%	Buruk

III. METODOLOGI PENELITIAN

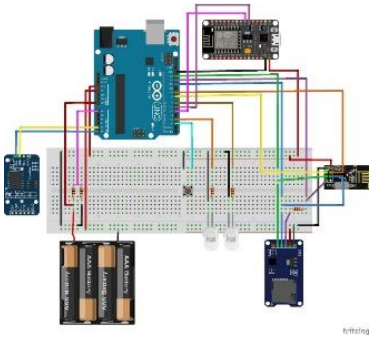
A. Perancangan Perangkat dan Alur Sistem

Perancangan perangkat terdiri dari 2 perancangan yaitu perancangan node dan perancangan gateway. Perancangan node yang terdiri dari 3 komponen utama yaitu Arduino Nano sebagai microcontroller, NRF24L01 sebagai modul komunikasi serta modul DSB18 sebagai sensor suhu sebagaimana disajikan pada Gambar 1 berikut.



Gambar 1. Perancangan Node

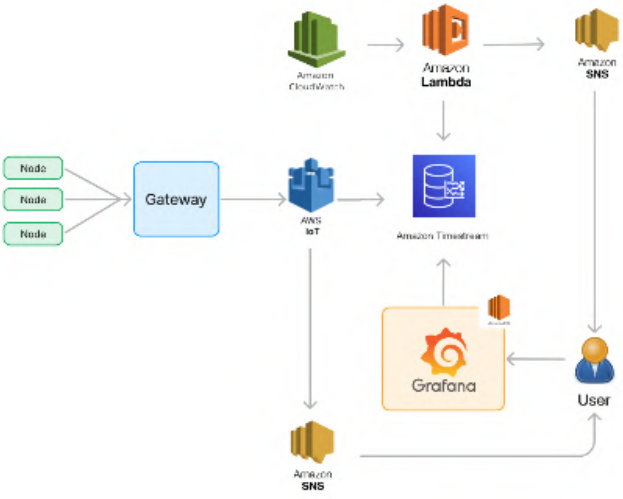
Sedangkan untuk rangkaian gateway, perangkat terdiri dari lebih banyak bagian seperti pada Gambar 2 berikut.



Gambar 2. Perancangan Gateway

Gateway terdiri dari dua microcontroller yaitu Arduino UNO dan NodeMCU ESP8266. Gateway juga disertai beberapa modul tambahan untuk menerima data dari node. NodeMCU akan menerima data yang sebelumnya telah dikumpulkan Arduino UNO menggunakan modul NRF24L01. Data dikirimkan melalui pin serial D7 dan D6. Pengiriman data tersebut dilakukan dengan memanfaatkan *library* Arduino Json. Setelah NodeMCU menerima data dari Arduino UNO, data tersebut akan digabungkan ke dalam variable menggunakan fungsi sprint membentuk sebuah pesan JSON untuk dikirimkan menggunakan protokol MQTT. Kemudian data yang diterima oleh AWS IoT Core akan diteruskan ke *database* Timestream.

Untuk *monitoring* data, *database* akan dihubungkan dengan Grafana *server*. Grafana akan menambahkan Timestream *Database* sebagai *data source*, menambahkan info ARN Timestream *Database* yang digunakan ke dalam konfigurasi sehingga Grafana dapat melakukan *query* dan menampilkan visualisasi dari data tersebut. Sedangkan untuk notifikasi, AWS Lambda akan melakukan *query* ke *database* setiap beberapa menit yang dijadwalkan menggunakan AWS CloudWatch EventBridge. Notifikasi akan dikirimkan ke email user yang telah melakukan subscription ke AWS SNS. Gambar 3 berikut akan menunjukkan ilustrasi dari alur transmisi data dan kerja sistem.



Gambar 3. Arsitektur *cloud*

B. Skenario Pengujian

1) Pengujian Performa AWS IoT

Pengujian performa AWS IoT berguna untuk mengetahui performa yang dimiliki AWS IoT sebagai layanan *serverless* untuk pengintegrasian *cloud* dengan perangkat IoT.

a. *Packet delivery*

Pengujian *packet delivery* dilakukan dengan dua tahap. Tahap pertama adalah memastikan tumpukan data yang diterima oleh AWS IoT sesuai dengan yang dikirimkan NodeMCU. Pengujian akan dilakukan secara berulang dengan mencoba mengirimkan informasi node dengan ID terkecil ke ID terbesar. Hal ini merupakan hal yang penting dikarenakan akan berpengaruh pada catatan waktu kapan informasi tersebut disimpan. Setelah informasi 3 node dengan ID 101, 103 dan 104 berhasil diterima dengan urutan yang sesuai tahap selanjutnya adalah melakukan pengujian delivery. Percobaan dilakukan dengan mengirimkan data dari masing – masing node secara bergantian sebanyak 30 pesan.

b. *Delay*

Pengujian *delay* dilakukan dengan 3 kali percobaan dengan masing – masing percobaan mengirimkan 10 data dari salah satu node. Selanjutnya bandingkan waktu pengiriman data pada NodeMCU dan penerimaan data pada AWS IoT.

c. Pengujian Waktu Penyimpanan

Pengujianini dilakukan untuk mengetahui waktu yang dibutuhkan AWS IoT menyimpan data ke *database* Timestream dengan membandingkan waktu penerimaan data pada AWS IoT dengan waktu yang tercatat pada *database*.

2) Pengujian Fungsional

Pengujian fungsionalitas bertujuan menguji keberhasilan sistem dalam menerima, menyimpan dan menampilkan data secara realtime serta mengirimkan notifikasi ke pengguna. Beberapa pengujian fungsional yang dilakukan pada penelitian ini antara lain:

a. AWS IoT *Rules* dapat memicu SNS mengirimkan notifikasi pada saat node terputus

b. CloudWatch dapat mengatur kerja Lambda untuk mendeteksi suhu lebih dari 20°C dan mentrigger SNS mengirimkan notifikasi ke email user yang telah melakukan subscribe.

3) Pengujian *Monitoring* Grafana

Pengujian ini dilakukan untuk menguji apakah implementasi dan rancangan dari sistem *monitoring* yang telah dilakukan berjalan dengan baik, Grafana dapat memvisualisasikan data ke dalam bentuk grafik serta Grafana dapat mengkonversi grafik ke dalam dokumen berformat PDF.

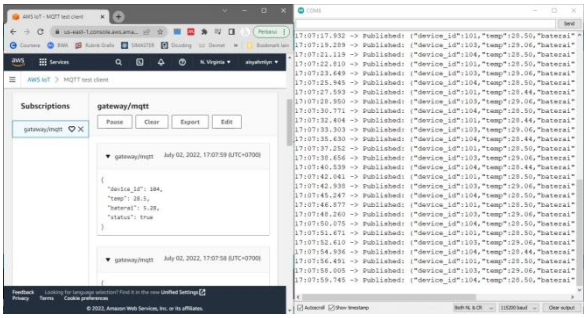
IV. HASIL DAN PEMBAHASAN

A. Hasil Pengujian Performa AWS IoT

1) Hasil Pengujian *Packet delivery*

Pada tahap pertama pengujia yaitu urutan data di diterima di AWS IoT. Melalui pengujian ini terbukti bahwa AWS IoT mampu menerima data yang dikirimkan NodeMCU menggunakan protokol MQTT. Urutan data yang diterima AWS IoT sesuai dengan yang dikirimkan perangkat gateway.

Gambar 4 merupakan salah satu bukti hasil pengujian. Pada pengujian tersebut, NodeMCU seperti yang terlihat pada serial monitor berhasil meneruskan pesan atau informasi node secara urut dengan melakukan published. Kemudian pada halaman AWS IoT, menggunakan MQTT test client, AWS IoT melakukan subscribe terhadap topik yang digunakan, lalu pesan baru akan muncul setiap kali perangkat mengirimkan pesan.



Gambar 4. Pengujian urutan data

Packet delivery merupakan *packet* yang berhasil sampai ke penerima. Penerima dalam hal ini adalah AWS IoT, namun AWS IoT dalam website testingnya hanya dapat menyimpan informasi data sebanyak 10 data. Oleh karena itu jumlah data yang diterima dilihat melalui hasil eksport data di Grafana. Dari hasil eksport data tersebut dapat diketahuni bahwa semua data yang dikirimkan dapat diterima oleh AWS IoT, masing – masing data node yg dikirimkan lengkap sejumlah 30 data.

2) Hasil Pengujian *Delay*

Untuk mendapatkan detail waktu diterimanya pesan pada AWS IoT, data perlu di eksport. Hasil file eksport akan berformat json dimana waktu disimpan ke dalam format timestamp. Format timestamp tersebut kemudian dikonversi ke waktu yang dapat dimengerti manusia. Selanjutnya waktu tersebut dan waktu pengiriman perangkat akan dibandingkan untuk mendapatkan nilai *delay* dan *delay* rata – rata. Tabel 4 menunjukkan hasil pengujian *delay*.

Tabel 4. Hasil Pengujian *Delay*

Pesan ke - n	Pengujian *Delay* Pengiriman data ke – n (ms)		
	1	2	3
1	226	244	469
2	274	251	241
3	263	447	233
4	199	224	283
5	217	213	257
6	219	223	257
7	213	238	273
8	209	263	222
9	234	239	267
10	240	141	233
Delay Rata-rata	229,4	248,3	273,5

Tabel 5. Hasil Pengujian Waktu Penyimpanan Data

Pesan ke - n	Pengujian waktu Penyimpanan data ke – n (ms)		
	1	2	3
1	462	485	485
2	468	471	492
3	452	284	489
4	448	580	469
5	521	428	561
6	471	474	495
7	530	477	480
8	466	478	493
9	455	587	592
10	452	470	503
Waktu Rata-rata	472,5	473,4	505,9

Melalui pengujian ini dapat terlihat bahwa *delay* terbesar selama pengiriman berada pada 469 ms sedangkan *delay* terkecil mencapai 141 ms. Sedangkan nilai *delay* rata – rata untuk ketiga percobaan tersebut adalah 250 ms. Penilaian mengenai hasil pengujian ini apabila mengacu pada standar TIPHON akan berindeks 3 yang berarti berkategori "Bagus".

Hasil pengujian QoS untuk delay dan packet delivery dalam penelitian ini akan dikalkulasikan sesuai standarisasi TIPHON. Seperti yang telah disebutkan sebelumnya indeks untuk pengujian packet delivery adalah 4 sedangkan untuk delay adalah 3. Bila dikalkulasikan nilai Quality of Services dari parameter delay dan packet delivery, maka pengujian QoS dalam pengiriman data dari perangkat gateway NodeMCU menuju AWS IoT bernilai 3,5 atau masuk dalam kategori "Memuaskan".

3) Hasil Pengujian Waktu Penyimpanan Data

Selain menguji *delay* pengiriman data, setelah mengetahui waktu penerimaan AWS IoT, dilakukan pula pengujian proses penyimpanan data (Tabel 5). Dengan pengujian ini, dapat diketahui berapa lama waktu yang dibutuhkan AWS IoT untuk menyimpan data yang diterima ke dalam *database*. Pengujian ini dilakukan dengan membandingkan nilai pada AWS IoT dan yang tercatat pada *database* Timestream. Meskipun terdapat perbedaan waktu dikarenakan region pada AWS yang digunakan berada di North Virginia yang menggunakan format waktu UTC, pengujian ini akan dilakukan dengan membandingkan nilai menit dan detik yang tercatat. Berikut merupakan hasil pengujian penyimpanan data pada AWS IoT.

Melalui hasil pengujian ini dapat terlihat bahwa waktu terlama AWS IoT untuk menyimpan data adalah 592 ms sedangkan waktu tersingkatnya yaitu 284 ms. Sedangkan nilai waktu rata – rata untuk ketiga percobaan tersebut adalah 483.9 ms.

B. Hasil Pengujian Fungsional

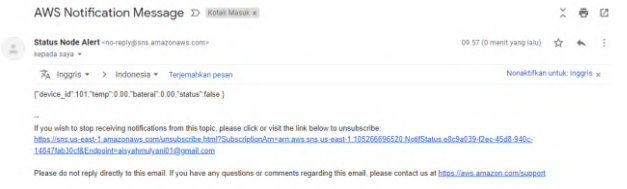
1) Hasil Pengujian AWS IoT *Rules*

Pengujian fungsional yang pertama adalah AWS IoT *Rules*. Setelah pada pengujian bagian sebelumnya AWS IoT telah berhasil menerima data dari perangkat gateway secara urut, pengujian selanjutnya adalah pengujian AWS IoT *Rules* yang telah dikonfigurasi dengan dua fungsi yaitu fungsi penyimpanan dan notifikasi. *Rules* yang dibuat akan membuat perangkat secara tidak langsung terhubung dengan layanan lain seperti *database* untuk penyimpanan dan AWS SNS untuk notifikasi. Pengujian mengenai penyimpanan sebelumnya telah dilakukan maka pada bagian ini akan dibahas mengenai fungsi notifikasi.

Berikut merupakan hasil dari pengujian notifikasi node yang tidak aktif. Node yang tidak aktif tersebut memiliki status *false*. Ketika AWS IoT mendapati pesan dengan kategori tersebut maka AWS IoT akan memicu AWS SNS untuk mengirimkan notifikasi berupa email. Notifikasi yang dikirimkan berupa informasi ID node yang tidak aktif seperti dapat dilihat pada Gambar 5 dan 6.

```
09:56:52.423 -> Waiting for NTP time sync: .
09:56:52.943 -> PubSubClient connecting to: a2ea5t93w76b6a-ats.iot.us-east-1.amazonaws.c
09:57:01.808 -> Published: {"device_id":104,"temp":28.25,"baterai":5.41,"status":true }
09:57:03.950 -> Published: {"device_id":104,"temp":28.25,"baterai":5.40,"status":true }
09:57:05.989 -> Published: {"device_id":101,"temp":28.31,"baterai":4.99,"status":true }
09:57:06.981 -> Published: {"device_id":103,"temp":28.81,"baterai":4.24,"status":true }
09:57:08.784 -> Published: {"device_id":104,"temp":28.25,"baterai":5.39,"status":true }
09:57:13.898 -> Published: {"device_id":101,"temp":0.00,"baterai":0.00,"status":false }
09:57:15.222 -> Published: {"device_id":103,"temp":28.81,"baterai":4.24,"status":true }
09:57:16.646 -> Published: {"device_id":104,"temp":28.31,"baterai":5.39,"status":true }
09:57:18.763 -> Published: {"device_id":101,"temp":28.25,"baterai":4.84,"status":true }
09:57:19.530 -> Published: {"device_id":103,"temp":28.81,"baterai":4.24,"status":true }
```

Gambar 5. Pengujian notifikasi melalui AWS IoT



Gambar 6. Hasil notifikasi yang diterima

2) Hasil Pengujian Lambda dan CloudWatch

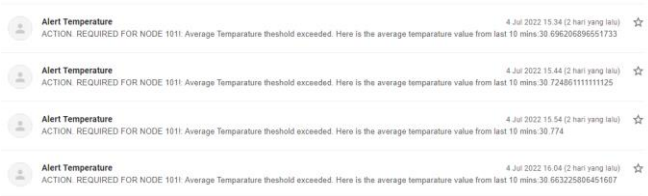
Pada penelitian ini Lambda digunakan untuk membuat sebuah fungsi agar dapat melakukan *query* ke *database* untuk melihat rata – rata besar suhu dalam 2, 5 atau 10 menit terakhir dan mengecek apakah nilai tersebut normal atau tidak. Satu fungsi Lambda yang dibuat pada penelitian ini digunakan untuk mengecek satu Node.

Fungsi Lamba berikut ini diatur kerjanya oleh Cloud Watch EventBridge sehingga penjadwalan kerja fungsi Lambda dilakukan di Cloud Watch EventBridge. Pengujian Lambda dan CloudWatch dilakukan untuk membuktikan bahwa fungsi yang telah dijalankan di Lambda dapat berfungsi dengan baik serta membuktikan bahwa CloudWatch mampu mengatur waktu kerja Lambda (Tabel 6).

Tabel 6. Hasil Pengujian Lambda dan CloudWatch

Fungsi yang diuji	Perulangan	Hasil
Dapat mengirimkan notifikasi saat suhu lebih dari 20°C	2 menit	Berhasil
	5 menit	Berhasil
	10 menit	Berhasil

Berikut ini merupakan salah satu hasil pemberitahuan email yang diterima ketika Lambda mendeteksi suhu yang tidak normal. Notifikasi tersebut dikirimkan setiap 10 menit sekali sebagaimana pengaturan yang dilakukan pada CloudWatch EventBridge (Gambar 7).



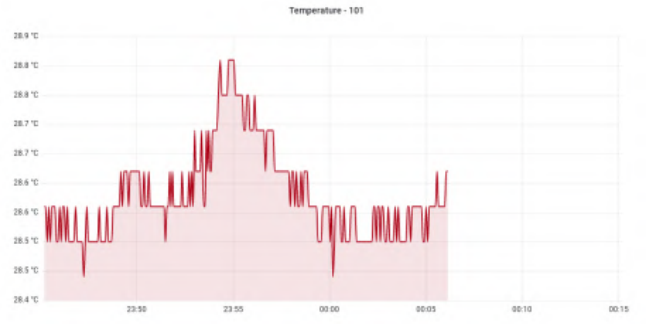
Gambar 7. Email pengujian Lambda dan *Cloud*Watch

C. Hasil Pengujian *Monitoring* Menggunakan Grafana

Monitoring dilakukan untuk memantau nilai suhu dan juga baterai dari perangkat node sehingga dashboard dari website *monitoring* dibuat untuk menampilkan visualisasi dalam bentuk grafik timeseries dan gauge. Grafik timeseries digunakan untuk melihat naik atau turunnya besaran suhu sedangkan gauge digunakan untuk melihat besaran suhu dan tegangan baterai terkini. Hasil konfigurasi dashboard *monitoring* dapat dilihat pada Gambar 8. Untuk grafik timeseries, sumbu x memberikan informasi mengenai waktu sedangkan sumbu y memberikan informasi mengenai besaran suhu yang tercatat.

Gambar 8. Dashboard *monitoring* cold-chain

Selain melakukan *monitoring*, apabila suatu pengiriman cold-chain logistic telah selesai, pengguna dapat melakukan dokumentasi dengan melakukan eksport data ke dalam format CSV. Dalam file CSV tersebut akan berisikan log data suhu atau baterai. Selain itu, agar bentuk terakhir grafik selama pengiriman dapat disimpan atau terdokumtasi dengan baik, pengguna dapat melakukan eksport grafik ke dalam file berformat PDF (Gambar 9).



Gambar 9. Hasil Pengujian Grafana Reporter

V. KESIMPULAN

Dari pengujian yang telah dilakukan serta hasil yang didapatkan maka melalui penelitian ini dapat disimpulkan bahwa:

- Pengintegrasian perangkat gateway dan *cloud* dengan memanfaatkan layanan *serverless* yaitu AWS IoT berhasil dilakukan dan menghasilkan perfoma yang baik dibuktikan dengan hasil pengujian QoS berstandar TIPHON yang berkategorikan memuaskan.
- Sistem arsitektur *serverless* yang dirancang melalui uji fungsionalitas yang dilakukan telah terbukti berhasil menjalankan semua fungsinya yaitu menerima, menyimpan, menampilkan data serta mengirimkan notifikasi ke pengguna melalui email.
- Data yang disimpan pada sistem arsitektur *serverless* yang dirancang dapat di*monitoring* menggunakan Grafana.

DAFTAR PUSTAKA

[1] A. W. Burange and H. D. Misalkar, "Review of Internet of Things in development of smart cities with data management & privacy," 2015 International Conference on Advances in Computer Engineering and Applications, 2015, pp. 189-195, doi: 10.1109/ICACEA.2015.7164693.

[2] Q. Huang and X. Yan, "On the Application of Internet of Things (IOT) in Cold Chain Logistics Management," in Proceedings of the 2018 International Symposium on Communication Engineering & Computer Science (CECS 2018), Hohhot, China, 2018. doi: 10.2991/cecs-18.2018.80.

[3] D. Oktaviani, F. S. Papilaya, and P. F. Tanaem, "Perancangan Aplikasi E-Menu Restaurant dengan Menggunakan Cloud Computing dan Serverless Architecture Lambda," Explore. jurnal. sistem. inf. dan. telematika, vol. 12, no. 1, p. 1, Apr. 2021, doi: 10.36448/jsit.v12i1.1887.

[4] H. A. Rochman, R. Primananda, and H. Nurwasito, "Sistem Kendali Berbasis Mikrokontroler Menggunakan Protokol MQTT pada Smarthome," Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, vol. 1, no. 6, pp. 445–455, 2017.

[5] Aprianto Budiman, M. Ficky Duskarnaen, and Hamidillah Ajie, "ANALISIS QUALITY OF SERVICE (QOS) PADA JARINGAN INTERNET SMK NEGERI 7 JAKARTA," pinter, vol. 4, no. 2, pp. 32–36, Dec. 2020, doi: 10.21009/pinter.4.2.6.