

**ANALISIS QUALITY OF SERVICE (QOS) MENGGUNAKAN
ROUTING STATIS DAN DINAMIS PADA JARINGAN
INTERNET OF THINGS (IOT)**

SKRIPSI

GALILEO GULAMPATI MARAPRATAMA PURBA

201402041



**PROGRAM STUDI S-1 TEKNOLOGI INFORMASI
FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI
UNIVERSITAS SUMATERA UTARA
MEDAN
2025**

ANALISIS QUALITY OF SERVICE (QOS) MENGGUNAKAN ROUTING
STATIS DAN DINAMIS PADA JARINGAN
INTERNET OF THINGS (IOT)

SKRIPSI

Diajukan untuk melengkapi dan memenuhi syarat memperoleh ijazah Sarjana
Teknologi Informasi

GALILEO GULAMPATI MARAPRATAMA PURBA

201402041



PROGRAM STUDI S-1 TEKNOLOGI INFORMASI

FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI

UNIVERSITAS SUMATERA UTARA

MEDAN

2025

PERSETUJUAN

Judul : ANALISIS QUALITY OF SERVICE (QOS)
MENGGUNAKAN ROUTING STATIS DAN DINAMIS
PADA JARINGAN INTERNET OF THINGS (IOT)

Kategori : SKRIPSI

Nama : GALILEO GULAMPATI MARAPRATAMA PURBA

Nomor Induk Mahasiswa : 201402041

Program Studi : SARJANA (S1) TEKNOLOGI INFORMASI

Fakultas : ILMU KOMPUTER DAN TEKNOLOGI INFORMASI
UNIVERSITAS SUMATERA UTARA

Medan, 06 Januari 2025

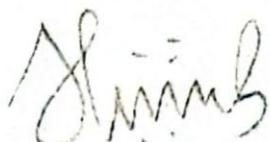
Komisi Pembimbing :

Pembimbing 2



Seniman S.Kom., M.Kom
198705252014041001

Pembimbing 1



Dr. Niskarto Zendrato S.Kom., M.Kom
198909192018051001

Diketahui/disetujui oleh :

Program Studi S-1 Teknologi Informasi

Ketua



Dedy Anzaldi S.T., M.Kom.
197908312009121002

PERNYATAAN

ANALISIS QUALITY OF SERVICE (QOS) MENGGUNAKAN ROUTING STATIS DAN
DINAMIS PADA JARINGAN INTERNET OF THINGS (IOT)

SKRIPSI

Saya mengakui bahwa skripsi ini adalah hasil karya saya sendiri, kecuali beberapa kutipan dan ringkasan yang masing – masing telah disebutkan sumbernya.

Medan, 06 Januari 2025

Galileo Gulampati Marapratama Purba
201402041

KATA PENGANTAR

Puji syukur kami panjatkan kehadirat Allah SWT kerena masih berkenan memberikan rahmat dan karunia-Nya kepada penulis sehingga penulis dapat menyelesaikan penyusunan skripsi ini sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer di Prorgam Studi S1 Teknologi Informasi, Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Suamtera Utara. Shalawat dan salam penulis hadiahkan khusus untuk Nabi besar Muhammad SAW. Semoga peneliti maupun pembaca mendapatkan syafa'atnya di hari akhir kelak.

Dalam penyusunan karya tulis ini, begitu banyak rintangan yang peneliti temui. Namun berkat ridha Allah SWT dan bantuan dari berbagai pihak, peneliti akhirnya dapat menyelesaikan skripsi ini. Maka dari itu peneliti mengucapkan banyak terima kasih kepada pihak-pihak yang turut serta membantu dalam pembuatan dan kelancaran skripsi ini, yaitu :

1. Keluarga penulis yaitu ibunda Gita Ismayawati, ayahanda Ir. Ademara Purba, dan kakanda Maya Sofhia, S.Kom., M.Kom., serta keluarga dan saudara-saudari lain yang telah memberikan doa, motivasi, dan dukungan penuh atas segala yang peneliti lakukan untuk menyelesaikan skripsi.
2. Bapak Dr. Muryanto Amin, S.Sos, M.Si., yang merupakan Rektor Universitas Sumatera Utara.
3. Ibu Dr. Maya Silvi Lydia, B.Sc., M.Sc. selaku Dekan Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Sumatera Utara
4. Bapak Dedy Arisandi, S.T., M.Kom. selaku Ketua Program Studi S1 Teknologi Informasi Universitas Sumatera Utara.
5. Bapak Ivan Jaya, S.Si., M.Kom. selaku Sekretaris Program Studi S1 Teknologi Informasi Universitas Sumatera Utara.
6. Bapak Dr. Niskarto Zendrato S.Kom., M.Kom selaku Dosen Pembimbing I penulis yang telah memberikan banyak masukan, bantuan, bimbingan serta meluangkan waktunya kepada penulis.
7. Bapak Seniman, S.Kom, M.Kom. selaku Dosen Pembimbing II penulis yang telah memberikan banyak masukan, bantuan, bimbingan serta meminjamkan alat dalam membantu proses penelitian penulis.

8. Bapak Ainul Hizriadi S.Kom., M.Sc. selaku Dosen penguji I penulis yang telah memberikan banyak masukan dan saran kepada penulis.
9. Rossy Nurhasanah S.Kom., M.Kom. selaku Dosen penguji II penulis yang telah memberikan banyak masukan dan saran kepada penulis.
10. Ibu Dr. Erna Budhiarti Nababan M.IT. selaku Dosen Pembimbing Akademik penulis yang telah membantu penulis selama masa perkuliahan terkait perkembangan akademik penulis.
11. Seluruh Bapak/Ibu Dosen dan staff pegawai Program Studi Teknologi Informasi yang telah membimbing dan membantu penulis selama masa perkuliahan penulis yang tidak dapat penulis sebutkan satu-persatu.
12. Indah Zahrani Lubis selaku seseorang yang selalu mendukung penulis dan menemani penulis selama masa perkuliahan, masa penulisan skripsi, dan ketika penulis membutuhkan saran.
13. Teman-teman penulis di laboratorium Internet of Things yaitu Gideon Tulus Hatta Yuda Siahaan, Farhan Yehanda, Mustafa Kamal Nasution, Muhammad Ridho Abidin Damanik, Rheza Yudhistira Ramadhan, dan Chalil Al Vareel yang selalu memberikan semangat, motivasi, saran, dan masukan yang membantu penulis dalam pembuatan skripsi.
14. Teman-teman penulis selama masa kuliah yaitu Raihan Alifya Lubis, Hizkia Winter Simaka Purba, Franda Christiano Siahaan, Muhammad Ridwan Rizki, Mikhael Aditha Sembiring Meliala, dan Amelia Angelita Silalahi yang telah bersama penulis selama melewati masa perkuliahan.
15. Rekan-rekan angkatan 2020 program studi Teknologi Informasi Universitas Sumatera Utara yang tidak dapat disebutkan satu-persatu namanya, yang telah memberikan semangat sehingga penulis dapat menyelesaikan skripsi ini.
16. Beberapa senior yang memberikan bantuan dan masukan kepada penulis yaitu bang Muhammad Saddam Zikri Dalimunthe, bang Nanda Ambiya, bang Fachri Adrian, bang Davin Guntur Habibi, dan bang Meizar Fachriza.
17. Seluruh pihak lain yang telah terlibat dalam pembuatan skripsi ini yang tidak bisa disebutkan satu-persatu.

Penulis menyadari skripsi ini masih jauh dari kata sempurna sehingga saran dan kritik yang bersifat membangun pun sangat diharapkan.

Medan, 06 Januari 2025



Penulis



ANALISIS QUALITY OF SERVICE (QOS) MENGGUNAKAN ROUTING STATIS DAN DINAMIS PADA JARINGAN INTERNET OF THINGS (IOT)

ABSTRAK

Internet of Things (*IoT*) adalah salah satu cabang teknologi yang sedang berkembang saat ini. *IoT* telah digunakan di berbagai hal seperti untuk bidang industri, bidang medis, perkembangan rumah cerdas (*smart home*), pengumpulan dan dukungan observasi serta pengambilan data lapangan, hingga pada kendaraan seperti mobil otonom ataupun *drone*. Dengan semakin banyaknya perangkat *IoT* dan pemanfaatannya di Indonesia. Maka, perangkat *IoT* yang ada harus dipastikan bekerja dengan seoptimal mungkin. Salah satu upaya untuk mengoptimalkan kerja perangkat *IoT* adalah menggunakan protokol *routing* jaringan yang baik. Dengan menggunakan protokol *routing* yang baik pada perangkat *IoT* tersebut. Maka, transfer data antar perangkat *IoT* tersebut akan meningkat kualitasnya sehingga keseluruhan sistem *IoT* tersebut dapat berjalan dengan sebaik-baiknya. Untuk menentukan protokol *routing* yang terbaik untuk perangkat *IoT* dapat dilakukan analisis protokol *routing* berdasarkan parameter *Quality of Service (QoS)* yaitu *throughput*, *packet loss*, *delay*, *latency*, dan *jitter*. Protokol *routing* yang diujikan untuk penelitian ini adalah protokol *routing* statis, *routing* dinamis *Routing Information Protocol (RIP)*, dan *routing* dinamis *Open Shortest Path First (OSPF)*. Jumlah perangkat *IoT* yang digunakan adalah 45 buah perangkat *node* virtual berbasis arsitektur *aarch64* dan 5 buah perangkat *node* fisik berbasis arsitektur *esp32*. Hasil penelitian menunjukkan bahwa protokol *routing* statis memiliki nilai *throughput* terbesar dan sangat baik menangani *delay* dan *jitter* pada jaringan. Protokol *routing* dinamis *RIP* memiliki nilai *latency* terendah, dan protokol *routing* dinamis *OSPF* memiliki kemampuan terbaik untuk menangani *packet loss* pada jaringan *IoT*.

Kata kunci : *IoT (Internet of Things)*, Protokol *Routing*, *QoS (Quality of Service)*

***QUALITY OF SERVICE (QOS) ANALYSIS USING STATIC AND DYNAMIC
ROUTING ON INTERNET OF THINGS (IOT) NETWORKS***

ABSTRACT

Internet of Things (*IoT*) is one of the growing subjects of technology today. *IoT* has been used in various ways such as for the industrial field, medical field, *smart home* development, collection and support of observations and field data retrieval, even to vehicles such as autonomous cars or *drones*. With the increasing number of *IoT* devices and their utilization in Indonesia. So, the existing *IoT* devices must be ensured to work as optimally as possible. One of the efforts to optimize the work of *IoT* devices is to use a good network *routing* protocol. By using a good routing protocol on the *IoT* device. Then, the data transfer between *IoT* devices have increased quality so that the entire *IoT* system can run at its finest. To determine the best routing protocol for *IoT* devices, routing protocol analysis can be done based on Quality of Service (QoS) parameters, which is throughput, packet loss, delay, latency, and jitter parameters. The routing protocols tested for this research are static routing protocol, dynamic routing named Routing Information Protocol (RIP), and dynamic routing named Open Shortest Path First (OSPF). The number of *IoT* devices used is 45 virtual node devices based on aarch64 architecture and 5 physical node devices based on esp32 architecture. The results show that the static routing protocol has the largest throughput value and is very good at handling network delay and jitter, RIP dynamic routing protocol has the lowest latency values, and OSPF dynamic routing protocol has the best ability to handle packet loss on *IoT* networks.

Keywords: *IoT (Internet of Things)*, *Routing Protocol* , *QoS (Quality of Service)*

DAFTAR ISI

PERSETUJUAN.....	iii
PERNYATAAN	iv
KATA PENGANTAR	v
ABSTRAK	viii
ABSTRACT	ix
DAFTAR ISI.....	x
DAFTAR TABEL.....	xiii
DAFTAR GAMBAR.....	xiv
BAB 1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah.....	4
1.3 Tujuan Penelitian	5
1.4 Batasan Masalah Penelitian	5
1.5 Manfaat Penelitian	5
1.6 Metodologi Penelitian.....	6
BAB 2 LANDASAN TEORI	7
2.1 Quality of Service (QoS).....	7
2.1.1 Throughput	7
2.1.2 Delay.....	8
2.1.3 Packet Loss.....	8
2.1.4 Latency	8
2.1.5 Jitter	8
2.2 Protokol <i>Routing</i> Jaringan	9
2.3 Routing Statis.....	9
2.4 Routing Dinamis	9
2.4.1 Routing Information Protocol (RIP).....	10
2.4.2 Open Shortest Path First (OSPF).....	10
2.5 Internet of Things (IoT)	10
2.6 Mikrokontroller ESP32.....	12
2.7 Message Queuing Telemetry Transport (MQTT)	13

2.8	Graphical Network Simulator 3 (GNS3)	13
2.9	Quick Emulator (QEMU)	13
2.10	Wireshark	14
2.11	Penelitian Terdahulu	14
2.12	Perbedaan Penelitian	19
BAB 3 ANALISIS DAN PERANCANGAN SISTEM	20	
3.1	Data Yang Digunakan	20
3.2	Arsitektur Umum Sistem	20
3.2.1	Persiapan Awal	22
3.2.2	Penyiapan Hardware dan Software	23
3.2.3	Penentuan Alamat IP Address	26
3.2.4	Deteksi Paket Data	28
3.2.5	Analisis Paket Data	29
3.2.6	Keluaran Akhir	30
BAB 4 IMPLEMENTASI DAN PENGUJIAN	31	
4.1	Implementasi Sistem	31
4.1.1	Perangkat Keras dan Perangkat Lunak	31
4.1.2	Alat	31
4.2	Implementasi Program	32
4.2.1	Implementasi Program di Komputer Host	32
4.2.2	Implementasi Program di Virtual Machine	32
4.2.3	Implementasi Program di Mikrokontroller ESP32	40
4.3	Implementasi Pengambilan Data	47
4.3.1	Pengaktifan Protokol Routing Pilihan	47
4.3.2	Pengaktifan Software Wireshark	49
4.3.3	Pengaktifan Node Publisher dan Subscriber	50
4.3.4	Penghitungan Waktu	51
4.4	Analisis <i>QoS Protokol Routing</i> Berdasarkan Hasil Deteksi Wireshark	51
4.4.1	Analisis QoS Berdasarkan Throughput	52
4.4.2	Analisis QoS Berdasarkan Packet Loss	54
4.4.3	Analisis QoS Berdasarkan Delay	56
4.4.4	Analisis QoS Berdasarkan Latency	63

4.4.5 Analisis QoS Berdasarkan Jitter	66
4.4.6 Faktor yang Memengaruhi Hasil Analisis QoS	74
4.4.7 Hasil Akhir Keseluruhan Analisis QoS	76
4.4.8 Ringkasan Akhir Untuk Keseluruhan Hasil Analisis QoS	81
BAB 5 KESIMPULAN DAN SARAN	84
5.1 Kesimpulan	84
5.2 Saran	84
DAFTAR PUSTAKA	85



DAFTAR TABEL

Tabel 2.1 Penelitian Terdahulu	14
Tabel 3.1 Tabel IP Address Rancangan Sistem	26
Tabel 3.2 Perlakuan Setelah Deteksi Paket Data Selesai Dilaksanakan	28
Tabel 3.3 Format Data dan Topik MQTT yang Digunakan Setiap Node	29
Tabel 4.1 Statistik Hasil Pengujian Throughput jalur R1A-R1B	52
Tabel 4.2 Statistik Hasil Pengujian Throughput Jalur R1B-R4	53
Tabel 4.3 Statistik Hasil Pengujian Packet Loss Jalur R1A-R1B	55
Tabel 4.4 Statistik Hasil Pengujian Packet Loss Jalur R1B-R4	55
Tabel 4.5 Statistik Data Waktu Terima dan Kirim Paket Data Jalur R1A-R1B	57
Tabel 4.6 Statistik Data Waktu Terima dan Kirim Paket Data Jalur R1B-R4	60
Tabel 4.7 Statistik Hasil Hitung Rerata Delay di Jalur R1A-R1B	64
Tabel 4.8 Statistik Hasil Hitung Rerata Delay di Jalur R1B-R4	65
Tabel 4.9 Statistik Data Waktu Delay Paket Data Jalur R1A-R1B	67
Tabel 4.10 Statistik Data Waktu Delay Paket Data Jalur R1B-R4	71
Tabel 4.11 Hasil Analisis Berdasarkan Throughput	76
Tabel 4.12 Hasil Analisis Berdasarkan Packet Loss	77
Tabel 4.13 Hasil Analisis Berdasarkan Delay	78
Tabel 4.14 Hasil Analisis Berdasarkan Latency	79
Tabel 4.15 Hasil Analisis Berdasarkan Jitter	80
Tabel 4.16 Ringkasan Hasil Analisis Dengan Peringkat Angka	81

DAFTAR GAMBAR

Gambar 2.1 Implementasi IoT Pada Rumah Cerdas (Smart Home)	11
Gambar 2.2 Implementasi IoT Pada Pertanian	11
Gambar 2.3 Mikrokontroller ESP32	12
Gambar 3.1 Arsitektur Umum	21
Gambar 3.2 Bentuk Topologi Jaringan	22
Gambar 3.3 Spesifikasi inti prosessor dan RAM untuk GNS3 VM	23
Gambar 3.4 Tambahan RAM memanfaatkan swap file untuk GNS3 VM	23
Gambar 3.5 Spesifikasi inti prosessor dan RAM untuk MikroTik CHR	24
Gambar 3.6 Spesifikasi RAM untuk QEMU	24
Gambar 3.7 Spesifikasi inti prosessor untuk QEMU	25
Gambar 3.8 Rangkaian detail sambungan kabel pada ESP32-1 dan ESP32-2	25
Gambar 4.1 Memasukkan Mikrotik CHR ke Bidang Kerja GNS3	33
Gambar 4.2 Hasil Konfigurasi Dilihat Melalui Software Winbox	33
Gambar 4.3 Memindahkan Folder DQIB ke GNS3 VM	34
Gambar 4.4 Memindahkan Folder DQIB ke Node Virtual GNS3	35
Gambar 4.5 Ekstraksi isi Folder DQIB di Node Virtual GNS3	35
Gambar 4.6 Baris Perintah untuk Menjalankan Program QEMU	36
Gambar 4.7 QEMU Berjalan dengan Arsitektur Prosessor ARM64/Aarch64	36
Gambar 4.8 Hasil Penyusunan Mikrokontroller dan Sensor di Breadboard	41
Gambar 4.9 Menghubungkan rangkaian sensor ke komputer host	41
Gambar 4.10 Menghubungkan ESP32 ke komputer host	45
Gambar 4.11 Protokol Routing Statis Aktif	48
Gambar 4.12 Protokol Routing Dinamis RIP Aktif	48
Gambar 4.13 Protokol Routing Dinamis OSPF Aktif	48
Gambar 4.14 Mengaktifkan Wireshark di Salah Satu Jalur Data	49
Gambar 4.15 Logo Kaca Pembesar Menandai Aktifnya Wireshark di Jalur Data Terpilih	49
Gambar 4.16 Mengaktifkan Node MQTT Subscriber di GNS3	50
Gambar 4.17 Mengaktifkan Node MQTT Publisher di GNS3	50
Gambar 4.18 Mengaktifkan Node MQTT Publisher ESP32 Dengan Sensor	50
Gambar 4.19 Mengaktifkan Node MQTT Publisher ESP32 Tanpa Sensor	50
Gambar 4.20 Hasil Deteksi Software Wireshark di Salah Satu Jalur Data	51
Gambar 4.21 Hasil Penggunaan Fitur Wireshark Untuk Perhitungan Throughput	52
Gambar 4.22 Hasil Penggunaan Fitur Wireshark Untuk Perhitungan Packet Loss	54
Gambar 4.23 Hasil Pelacakan Paket Untuk Perhitungan Delay pada Node aarch-xx	57
Gambar 4.24 Hasil Pelacakan Paket Untuk Perhitungan Delay pada Node esp-x	57
Gambar 4.25 Router 1B Terhubung ke Seluruh Jalur Jaringan	75

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Internet of Things (IoT) telah menjadi sebuah cabang teknologi yang sedang berkembang saat ini. *IoT* telah digunakan di berbagai hal seperti untuk bidang industri, bidang medis, perkembangan rumah cerdas (*smart home*), pengumpulan dan dukungan observasi serta pengambilan data lapangan, hingga pada kendaraan seperti mobil otonom ataupun *drone*. Menurut Kementerian Komunikasi dan Informatika (Kemenkominfo), jumlah perangkat *IoT* pada tahun 2022 di Indonesia diperkirakan sebanyak 400 juta perangkat serta diprediksi akan meningkat ke angka 678 juta perangkat pada tahun 2025 (Ayu, 2023).

Dengan semakin banyaknya perangkat *IoT* dan pemanfaatannya di Indonesia. Maka, perangkat *IoT* yang ada harus dipastikan bekerja dengan seoptimal mungkin. Salah satu upaya untuk mengoptimalkan kerja perangkat *IoT* adalah menggunakan protokol *routing* jaringan yang baik. Protokol *routing* adalah lapisan jaringan yang bertugas untuk membawa data melewati jaringan yang berbeda dengan cara memilih jalur yang terbaik untuk dilalui data dari titik pengirim ke titik tujuan (Mudhoep et al., 2021). Dengan menggunakan protokol *routing* yang baik pada perangkat *IoT* tersebut. Maka, transfer data antar perangkat *IoT* tersebut akan meningkat kualitasnya sehingga keseluruhan sistem *IoT* tersebut dapat berjalan dengan sebaik-baiknya.

Terdapat beberapa indikator yang digunakan untuk mengukur keoptimalan protokol *routing* pada perangkat *IoT*. Beberapa parameter yang umum digunakan untuk menentukan kualitas dari protokol *routing* jaringan adalah *Throughput*, *Delay*, *Packet loss*, *Latency*, dan *Jitter*. Untuk melakukan pengujian protokol *routing* di perangkat *IoT* dapat dilakukan dengan menggunakan *software* simulasi jaringan seperti *GNS3 Simulator*.

Analisis pengaruh *routing* jaringan terhadap perangkat internet ataupun perangkat *IoT* telah dilakukan oleh beberapa penelitian dengan perbedaan terletak pada protokol jaringan yang digunakan, variabel pengujian, dan teknik pengujian yang digunakan. Pada penelitian yang berjudul “Performance of Reactive Routing Protocols DSR and AODV in Vehicular Ad-Hoc

Networks Based on Quality of Service (Qos) Metrics” (Cañar et al., 2020), penelitian ini Melakukan analisis pada protokol *IoT* berbasis VANET (*Vehicular Ad-Hoc Networks*) tipe *Reactive Routing Protocols* yaitu *AODV* (*Ad-Hoc Demand Distance Vector*) dan *DSR* (*Dynamic Source Routing*) dengan variabel yang diujikan adalah *Throughput*, *Delay*, dan *Jitter*. Lalu, pengujian dilaksanakan dengan NS-3 Simulator. Hasil penelitiannya disebutkan bahwa Protokol *DSR* memiliki hasil yang lebih baik pada jumlah perangkat *IoT* yang lebih rendah, sebaliknya *AODV* memiliki hasil yang lebih baik pada jumlah perangkat *IoT* yang lebih tinggi.

Selain variabel-variabel yang telah disebutkan di penelitian sebelumnya. Variabel lain seperti *Latency* dan *Jitter* terdapat pada penelitian dengan judul “Implementasi Ad-Hoc Protocol Pada Tandem Multihop Wireless Network” (Agussalim et al., 2023), penelitian ini Melakukan perbandingan antar protocol *routing* pada *IoT* tipe *Ad-Hoc* yaitu *OLSR* (*Optimized Link State Routing*) & *BATMAN* (*Better Approach to Mobile Ad-hoc Networking*), variabel yang dipakai adalah *Delivery Probability*, *Average Latency*, dan *Jitter* dengan perhatian khusus pada hasil penelitiannya terletak pada besar data yang dikirim dalam satu waktu. Lalu, pengujian dilaksanakan dengan simulator *Mininet-WiFi*. Hasil penelitiannya menunjukkan bahwa Protokol *OLSR* & *BATMAN* terlihat stabil dan berperforma serupa satu sama lain ketika data yang dikirimkan sebesar 1 MB. Namun, ketika data yang dikirimkan sebesar 2 MB, performa protokol *OLSR* lebih cepat namun kurang stabil dan performa protokol *BATMAN* lebih lambat namun lebih stabil.

Penelitian selanjutnya yang salah satu variabelnya yang digunakan yaitu *packet loss* terdapat pada penelitian dengan judul “Implementasi Mobile Ad-Hoc Network Pada Daerah Pasca Bencana Dengan Protokol DSR” (Segara et al., 2022), penelitian ini melakukan implementasi prototipe *IoT* menggunakan protokol *routing* *IoT* tipe *Ad-Hoc* yaitu *DSR* (*Dynamic Source Routing*) untuk area rawan bencana, variabel yang digunakan adalah *End-to-End Delay*, *Packet Loss*, serta *Packet Delivery Ratio* (*PDR*). Lalu, pengujian dilaksanakan dengan NS-2 Simulator. Hasil penelitiannya menunjukkan bahwa protokol *routing* *DSR* memberikan peningkatan signifikan pada *PDR* jaringan dan menurunkan Tingkat *delay* dan *packet loss* ke angka yang cukup rendah seiring dengan peningkatan jumlah perangkat *IoT*.

Penelitian-penelitian yang telah disebutkan memilih menggunakan protokol *routing* yang bermacam-macam. Namun, penulis berencana menggunakan protokol *routing* yang lebih umum digunakan pada berbagai perangkat jaringan, salah satunya adalah protokol *routing* statis

dan *OSPF*. Penelitian yang memanfaatkan protokol *routing* statis dilakukan oleh (Lutfi et al., 2021). Penelitian ini membahas tentang implementasi protokol *routing* Statis pada *IoT* memanfaatkan LoRa dan *Websocket* sebagai *cloud* data. Penelitiannya dilakukan dengan *mini computer* Rasberry Pi, *microcontroller* Arduino Nano, dan Modul LoRa SX1278 dengan variabel yang diujikan yaitu *Successful Rate*, *Round Trip Time*, dan *Delay*. Hasil penelitian ini menghasilkan sistem yang telah dibangun dapat berfungsi dengan baik, variabel *successful rate* dengan angka diatas 90%, semua *node* memiliki *round trip time* dibawah 1 detik, dan *delay* dengan angka yang sangat rendah.

Di sisi lain. Terdapat juga penelitian yang memanfaatkan protokol *routing OSPF* pada perangkat *IoT*. Penelitian oleh (Luis García-Navas et al., 2021) melakukan pengujian dan komparasi 3 protokol *routing* yaitu *Routing Information Protocol (RIP)*, *Open Shortest Path First (OSPF)* dan *Enhanced Interior Gateway Routing Protocol (EIGRP)* di *software Cisco Packet Tracer* pada beberapa skenario dengan skenario pertama yaitu di perangkat *IoT* dan komputer tradisional. Lalu, skenario kedua pada perangkat *IoT* dan komputer tradisional serta perangkat yang menggunakan *Internet Protocol Television (IPTV)*. Variabel yang diujikan yaitu *Round Trip Time (RTT)* dengan hasil penelitian yaitu untuk skenario hanya perangkat *IoT*, protokol *routing RIP* berperforma terbaik dan *EIGRP* berperforma terburuk. Untuk skenario dengan *IPTV*, protokol *routing EIRGP* berperforma terbaik dan *RIP* berperforma terburuk. Protokol *routing OSPF* memiliki performa yang seimbang di 2 kondisi yang diujikan.

Selain itu, penelitian oleh (Kabir et al., 2021) juga melakukan komparasi 3 protokol *routing* yaitu *Routing Information Protocol (RIP)*, *Open Shortest Path First (OSPF)*, dan *Enhanced Interior Gateway Routing Protocol (EIGRP)* di *software simulasi Cisco Packet Tracer* dengan penekanan menggunakan topologi *full* dan *half mesh network* antar *router* pada *WSN (Wireless Sensor Network)* dengan data yang diujikan di jaringan yaitu *voice packet*, *HTTP packet*, dan *video Packet*. Variabel yang digunakan adalah *end-to-end latency*, *ping time*, *throughput* dengan hasil penelitian yaitu protokol *routing OSPF* memiliki performa terbaik dibandingkan protokol *routing* lain dan protokol *routing RIP* memiliki performa terburuk berdasarkan variabel yang diujikan.

Terdapat juga penelitian yang memanfaatkan protokol *routing OSPF*. Namun, hanya melakukan penelitiannya di perangkat bukan berbasis *IoT* seperti pada penelitian oleh (Mudhoep et al., 2021) yang menggunakan protokol *routing OSPF* dan *BGP* dengan protokol

redndansi *VRRP*, *HSRP*, dan *GLBP* pada perangkat komputer tradisional umumnya. Sehingga, penulis berniat untuk menggunakan protokol *OSPF* untuk penelitian *routing* pada *IoT* ini. Selain dengan protocol *OSPF*, penulis juga berencana menggunakan protokol *routing* lain seperti protokol *routing* statis yang sudah dilakukan oleh (Lutfi et al., 2021) sebagai protokol pembanding dan protokol *routing* *RIP* (Routing Information Protocol) sebagai protokol lain yang ingin diujikan oleh penulis.

Berdasarkan latar belakang dan penelitian terdahulu yang telah dibahas sebelumnya, penulis mengajukan penelitian untuk melakukan analisis pengaruh beberapa protokol *routing* jaringan seperti protokol *routing* statis, *RIP*, dan *OSPF* pada perangkat *IoT* untuk mencari protokol yang paling baik digunakan pada perangkat *IoT* berdasarkan parameter *Throughput*, *Delay*, *Packet loss*, *Latency*, dan *Jitter*. Penelitian ini akan dijalankan menggunakan *software* simulasi *GNS3* simulator. Dengan diketahuinya jenis protokol yang terbaik untuk perangkat berbasis *IoT* berdasarkan protokol yang akan diujikan. Maka protokol tersebut dapat diimplementasikan pada perangkat *IoT* untuk meningkatkan kualitas transfer data pada perangkat *IoT* tersebut. Penelitian ini diberi judul “**Analisis Quality of Service (QoS) Menggunakan Routing Statis dan Dinamis Pada Jaringan Internet of Things (IoT)**”

1.2 Rumusan Masalah

Peningkatan penggunaan perangkat *IoT* di berbagai bidang menyebabkan timbulnya urgensi untuk melakukan analisis berbagai jenis protokol *routing* yang paling baik untuk penggunaan di perangkat *IoT*. Beberapa protokol *routing* yang ada memberikan performa *transfer* data yang berbeda untuk tipe data dan protokol komunikasi tertentu. Sehingga diperlukan protokol yang lebih efisien dan efektif untuk *transfer* data perangkat *IoT*. Dengan semakin baiknya protokol *routing* yang digunakan, hal tersebut dapat meningkatkan kualitas *transfer* data antara perangkat *IoT* dengan pusat sistem dari perangkat *IoT* tersebut. Selain itu, dengan meningkatnya kualitas *transfer* data yang dilakukan, hal ini akan meningkatkan akurasi data yang di *trasnfer* dan kualitas respon perangkat *IoT* dalam menghadapi perubahan konfigurasi dalam sistem di jaringan *IoT* tersebut. Oleh karena itu, analisis pengaruh protokol *routing* terhadap perangkat *IoT* berdasarkan parameter *delay*, *jitter*, *latency*, *throughput*, dan *packet loss*

diperlukan untuk menganalisis protokol *routing* yang terbaik untuk digunakan di perangkat berbasis *IoT*.

1.3 Tujuan Penelitian

Tujuan dari penelitian ini adalah menerapkan protokol *routing* statis, *Routing Information Protocol (RIP)*, dan *Open Shortest Path First (OSPF)* pada perangkat *IoT* untuk dilakukan analisis *QoS* pada ketiga protokol tersebut apabila dijalankan pada platform jaringan berbasis perangkat *IoT*.

1.4 Batasan Masalah Penelitian

Batasan masalah pada penelitian ini antara lain :

1. Penelitian ini diimplementasikan pada *software* simulasi jaringan yaitu *GNS3* simulator.
2. Titik (*Node*) *IoT* yang digunakan adalah *node* virtual yang dipasangkan ke dalam *software* simulasi jaringan *GNS3 Simulator* dan *node* fisik berupa perangkat asli *IoT* yaitu perangkat *microcontroller ESP32*.
3. Protokol *routing* yang digunakan adalah protokol *routing* statis, *routing* dinamis *Routing Information Protocol (RIP)*, dan *routing* dinamis *Open Shortest Path First (OSPF)*.
4. Protokol jaringan akan diimplementasikan satu-persatu dan juga dilakukan analisis *QoS* satu-persatu per masing-masing konfigurasi protokol *routing*. Hal ini dilakukan agar setiap konfigurasi tidak saling tumpang tindih dan memengaruhi satu sama lain.
5. Penelitian ini menggunakan protokol *Message Queuing Telemetry Transport (MQTT)* sebagai protokol yang digunakan di perangkat *IoT* untuk proses kirim dan terima data.
6. Penelitian ini membatasi jumlah maksimal titik (*node*) *IoT* yang disimulasikan sebanyak 50 titik, baik berupa *node* fisik ataupun virtual.

1.5 Manfaat Penelitian

Manfaat dapat diperoleh dari penelitian yang dilakukan antara lain :

1. Mengetahui protokol *routing* jaringan yang paling efektif dan efisien dari protokol yang diujikan untuk penggunaan pada perangkat berbasis *IoT*.

2. Menjadi opsi alternatif untuk penentuan aplikasi *routing* jaringan pada jaringan berbasis *IoT* dibandingkan menggunakan protokol *routing* jenis lainnya.

1.6 Metodologi Penelitian

Penelitian ini akan dilakukan berdasarkan tahapan-tahapan berikut ini :

- 1) Studi Pustaka, hal ini dilakukan dengan mencari referensi dari berbagai sumber tepercaya dan melakukan peninjauan pustaka melalui buku-buku, jurnal, *e-book*, artikel ilmiah maupun situs internet yang berhubungan dengan *Internet-of-Things*, *Routing* jaringan, dan rancang bangun dari pengaplikasian *routing* jaringan pada perangkat *IoT*.
- 2) Analisis Kebutuhan Peralatan, hal ini dilakukan dengan melakukan peninjauan peralatan yang dibutuhkan, perangkat *software* yang dibutuhkan, hingga rancangan terkait penelitian yang akan dilakukan.
- 3) Perancangan atau *Design*, Pada tahap ini, penulis merancang pengaplikasian dari protokol *routing* pada perangkat *IoT* berdasarkan analisis yang sudah dilakukan, perancangan akan dilakukan secara simulasi menggunakan *software* simulasi jaringan dengan variabel yang sudah ditetapkan.
- 4) Implementasi, Pada tahap ini, penulis melakukan implementasi dari protokol *routing* pada *IoT* menggunakan *software* simluasi jaringan
- 5) Pengujian (*Testing*), tahap ini akan dilakukan dengan cara menjalankan hasil pengaplikasian dari protokol *routing* pada perangkat *IoT* secara simulasi dan dilakukan pengambilan data terkait efektivitas dari setiap protokol *routing* yang diaplikasikan berdasarkan variabel yang sudah ditentukan. Pengambilan data dilakukan menggunakan peralatan analisis jaringan seperti wireshark dan lainnya.
- 6) Analisis Hasil Pengujian. Setelah pengujian, dilakukan analisis terhadap data dari setiap hasil pengujian untuk melihat tingkat efektivitas dan efisiensi dari setiap protokol *routing* yang diujikan dan ditentukan apakah protokol *routing* yang diujikan memiliki kemampuan yang paling optimal atau tidak berdasarkan variabel yang sudah ditentukan.
- 7) Dokumentasi. Tahapan-tahapan proses yang dilakukan pada penelitian ini akan dituangkan dalam sebuah kesimpulan akhir yang dibuat penulis dalam bentuk skripsi.

BAB 2

LANDASAN TEORI

2.1 Quality of Service (QoS)

Quality of Service (QoS) atau kualitas layanan dalam bahasa Indonesia mengacu pada teknologi apapun yang mampu mengelola alur data di suatu jalur jaringan untuk mengurangi *packet loss* (hilangnya paket data), *latency*, dan *jitter* pada jaringan. *QoS* mengatur sumber daya jaringan dengan menetapkan sistem prioritas untuk tipe data tertentu pada sistem jaringan (Sukmandhani, 2020).

Pendapat lain yang dikemukakan oleh (Saputra et al., 2023) menyebutkan bahwa *QoS* (*Quality of Service*) merupakan kemampuan sistem jaringan untuk menyediakan layanan yang dapat menangani *jitter* dan *delay*. Di saat yang sama juga menyediakan *bandwidth* dengan baik.

2.1.1 Throughput

Menurut (Rifki Wardana & Santoso, 2023), *Throughput* adalah kemampuan asli dari suatu jaringan dalam proses kirim dan terima data. Dapat dianggap juga sebagai kecepatan data sebenarnya dari jaringan tersebut.

Pendapat lain oleh (Marfoq et al., 2020) menyebutkan bahwa *Throughput* adalah jumlah putaran komunikasi data di jaringan per satuan waktu.

Throughput dapat dicari dengan bantuan *software wireshark* atau menggunakan rumus *Throughput* menurut (Cañar et al., 2020) yang ditunjukkan pada rumus 2.1

$$Th = \frac{bytes * 8}{Ts * 1000} \quad (2.1)$$

Penjelasan :

Th = *Throughput*

bytes = Jumlah *bytes* di setiap paket data

Ts = Lama waktu simulasi

2.1.2 Delay

Menurut (Segara et al., 2022), *Delay* adalah waktu yang dibutuhkan paket data untuk melakukan perutean jaringan dari sumber ke tujuan. Waktu penundaan paket didasarkan pada rasio pengiriman paket. Ketika jarak dari titik sumber ke titik tujuan semakin jauh, maka dapat terjadi pembuangan paket data atau *packet drop* yang meningkat angkanya. Menurut (Segara et al., 2022) pula, rumus *Delay* ditunjukkan pada rumus 2.2.

$$\text{End - to - end delay} = \text{receive time} - \text{sent time} \quad (2.2)$$

2.1.3 Packet Loss

Menurut (Segara et al., 2022), *Packet Loss* adalah perbandingan jumlah paket data yang tidak mencapai tujuan atau penerima dibandingkan dengan jumlah paket yang berasal dari sumber atau pengirim. *Packet loss* dapat dicari dengan bantuan *software wireshark* atau menggunakan rumus *Packet Loss* yang menurut (Segara et al., 2022) ditunjukkan pada rumus 2.3

$$\text{Packet Loss} = \frac{\text{jumlah paket yang dikirim} - \text{jumlah paket yang diterima}}{\text{jumlah paket yang dikirim}} \times 100\% \quad (2.3)$$

2.1.4 Latency

Menurut (Agussalim et al., 2023), *Latency* adalah waktu rata-rata tempuh data yang dikirim dari titik sumber data ke titik tujuan data. Menurut (Novotný, 2023) rumus untuk mencari angka *latency* terdekat dapat dicari dengan rumus 2.4

$$\text{Latency} \approx \frac{\sim \text{Delay}}{2} \quad (2.4)$$

2.1.5 Jitter

Menurut (Agussalim et al., 2023), *Jitter* merupakan penundaan waktu yang terjadi dalam pengiriman data dari titik sumber data ke titik tujuan data melalui jaringan. *Jitter* menampilkan banyaknya variasi *delay* pada proses transmisi data di suatu jaringan. Hal ini diakibatkan oleh kemacetan jaringan dan kadangkala oleh perubahan rute. Seluruh hasil hitung *jitter* akan dirasionalkan dari berupa bilangan negatif menjadi bilangan positif. Rumus untuk perhitungan *Jitter* menurut (Cañar et al., 2020) ditunjukkan pada rumus 2.5

$$Jitt = \frac{[(Dly_A - Dly_p) - Jitt_p]}{16} \quad (2.5)$$

Penjelasan :

Jitt = Jitter

DlyA = Delay paket data A (sekarang)

DlyP = Delay paket data B (paket data sebelumnya)

JittP = Jitter di paket data sebelumnya

2.2 Protokol *Routing* Jaringan

Menurut (Meilisa et al., 2023), *Routing* adalah proses menghubungkan dan memindahkan suatu paket menggunakan *router-router* dari satu jaringan internet ke jaringan internet lainnya, protokol *routing* memiliki fungsi untuk menentukan jalur pada sebuah routing merupakan tugas dari protokol *routing* jaringan.

Di sisi lain, menurut (Mudhoep, 2021). Protokol *routing* adalah lapisan jaringan yang bertugas untuk membawa data melewati jaringan yang berbeda dengan cara memilih jalur yang terbaik untuk dilalui data dari titik pengirim ke titik tujuan. Protokol *routing* terbagi menjadi 2 jenis yaitu protokol *routing* statis dan protokol *routing* dinamis.

2.3 Routing Statis

Menurut (Farid, 2022), Protokol *routing* statis adalah protokol yang proses perutean pada jaringan tersebut didefinisikan secara manual oleh ahli jaringan dengan mengisi tabel penerusan jaringan pada setiap router yang terhubung ke jaringan. Ahli jaringan harus mengubah isi tabel penerusan jaringan ketika topologi jaringan mengalami perubahan.

2.4 Routing Dinamis

Protokol *routing* dinamis adalah protokol yang proses perutean pada jaringan tersebut didefinisikan secara otomatis menggunakan algortima tertentu oleh sistem sehingga antar *router* dapat berkomunikasi satu sama lain dengan memberikan informasi tentang jaringan dan

koneksi antar *router* (Farid, 2022). Terdapat beberapa protokol *routing* dinamis yang dikenal seperti *Routing Information Protocol (RIP)* dan *Open Shortest Path First (OSPF)*.

2.4.1 Routing Information Protocol (RIP)

Routing Information Protocol (RIP) merupakan *routing* protocol yang memakai algoritma *Distance-Vector Routing* atau protokol *routing* yang hanya melihat arah dan jarak untuk menentukan jalur ke jaringan tujuan. Protokol *RIP* menggunakan *hop count* sebagai metrik perhitungan dan rute dengan *hop count* terkecil yang akan menjadi rute terbaik (*best path*). Protokol *routing RIP* juga tidak memiliki peta yang lengkap tentang jaringan yang ada (Setiawan, 2022).

2.4.2 Open Shortest Path First (OSPF)

Menurut (Aulia et al., 2024), *Open Shortest Path First (OSPF)* adalah protokol *routing* dinamis (dynamic routing) yang menggunakan algoritma *Link-State* dan Djikstra. Protokol *OSPF* dapat mengatur dan menyebarkan informasi *routing* di antara jaringan sebagai respon terhadap setiap perubahan pada jaringan secara dinamis. Istilah "*Autonomous System*" (AS) yang digunakan di *OSPF* dapat menggambarkan kumpulan beberapa jaringan berorientasi perutean dengan metode yang serupa dan pengaturan kebijakan yang semuanya dapat dikontrol oleh ahli jaringan.

2.5 Internet of Things (IoT)

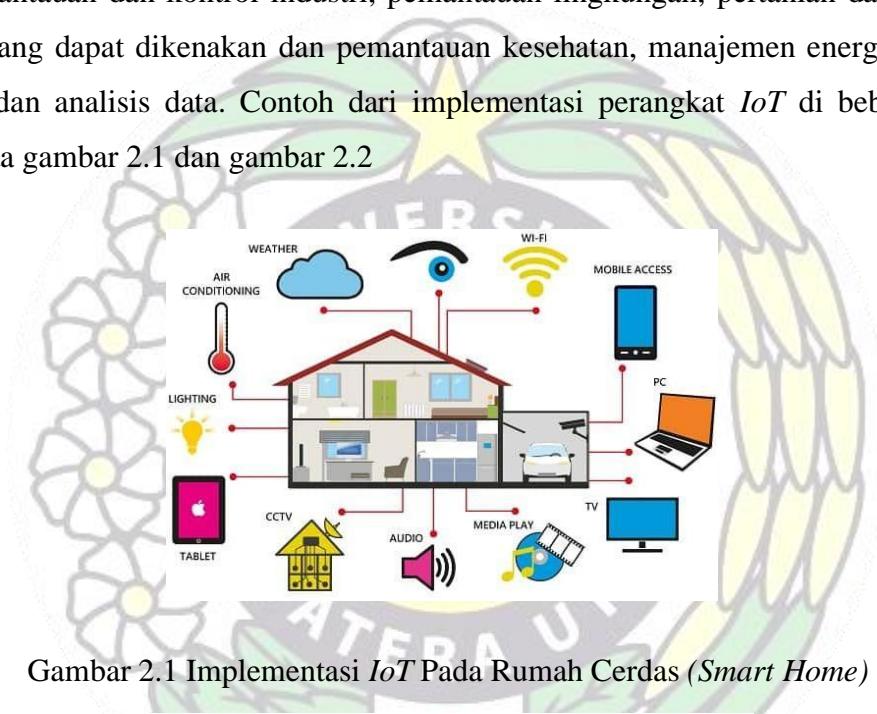
Internet of Things (IoT) merupakan suatu konsep yaitu satu atau lebih perangkat yang ditambahkan dengan sensor, perangkat lunak, dan teknologi lain yang dapat saling berkomunikasi dan melakukan pertukaran data melalui jaringan internet tanpa campur tangan manusia (Lutfi et al., 2021).

Di sisi lain. Menurut (Fuada et al., 2023), *Internet of Things (IoT)* merupakan sebuah teknologi yang diciptakan oleh manusia untuk menghubungkan berbagai objek fisik melalui koneksi Internet. Komponen dan perangkat tersebut dapat menerima data dan dapat dikontrol secara operasional menggunakan sensor dan aktuator jaringan.

Menurut (Hercog et al., 2023), karakteristik yang menunjukkan suatu sistem dapat diklasifikasikan sebagai *IoT* yaitu :

1. Perangkat harus dapat memperoleh data dari lingkungan sekitar dan mengirimkannya ke perangkat lain atau langsung ke Internet.
2. Perangkat harus dapat bereaksi sesuai dengan keadaan saat ini.
3. Perangkat harus dapat menerima informasi dari jaringan.
4. Perangkat harus mendukung komunikasi; perangkat IoT pada dasarnya merupakan bagian dari jaringan perangkat yang berkomunikasi satu sama lain melalui node pada jaringan yang sama.

IoT sebagai sebuah konsep dapat diimplementasikan pada berbagai bidang sebagaimana yang disampaikan juga menurut (Hercog et al., 2023) yaitu pada bidang rumah cerdas (*smart home*), pemantauan dan kontrol industri, pemantauan lingkungan, pertanian dan peternakan, perangkat yang dapat dikenakan dan pemantauan kesehatan, manajemen energi pintar, serta pencatatan dan analisis data. Contoh dari implementasi perangkat *IoT* di beberapa bidang terdapat pada gambar 2.1 dan gambar 2.2



Gambar 2.1 Implementasi *IoT* Pada Rumah Cerdas (*Smart Home*)



Gambar 2.2 Implementasi *IoT* Pada Pertanian

2.6 Mikrokontroller ESP32

Menurut (Hercog et al., 2023), mikrokontroller *ESP32* adalah perangkat *IoT* hasil produksi dari perusahaan Espressif Systems Company di kota Shanghai, Cina yang memiliki banyak fitur dan hemat biaya untuk pengembangan aplikasi *IoT*.



Gambar 2.3 Mikrokontroller ESP32

Terdapat beberapa keunggulan yang dimiliki oleh mikrokontroller *ESP32* dibandingkan dengan perangkat *IoT* lainnya sehingga banyak digunakan di berbagai aplikasi perangkat *IoT* sebagaimana yang dijabarkan oleh (El-Khozondar et al., 2024) :

5. Perangkat mikrokontroller *ESP32* memiliki perangkat keras bawaan *Wi-Fi* dan *Bluetooth* yang sangat memudahkan dalam pengaplikasian di dunia nyata.
6. Konsumsi daya rendah.
7. Memiliki kekuatan pemrosesan lebih baik dibandingkan dengan mikrokontroller lainnya dalam rentang biaya yang sama.
8. *ESP32* kompatibel dengan berbagai macam antarmuka *peripheral*. Diantaranya *I2C*, *SPI*, *UART*, *DAC* (*Digital-to-Analog Converter*), dan *ADC* (*Analog-to-Digital Converter*).
9. Berbiaya lebih rendah dibandingkan dengan mikrokontroller lainnya yang serupa.
10. Komunitas atau ekosistem yang cukup besar.
11. Memiliki fitur keamanan bawaan yang dapat dikonfigurasi.

Keunggulan lain dari mikrokontroller *ESP32* sebagaimana yang ditambahkan oleh (Hercog et al., 2023) yaitu dapat diprogram menggunakan berbagai macam *Integrated*

Development Environment (IDE) seperti Arduino IDE, PlatformIO, dan ESP-IDF (Espressif IoT Development Framework).

2.7 Message Queuing Telemetry Transport (MQTT)

Protokol *Message Queuing Telemetry Transport (MQTT)* adalah protokol yang menggunakan konsep *publish* dan *subscribe*, beroperasi dengan model klien-server di mana server pusat atau disebut *broker* bertugas sebagai penerima pesan dari klien yang dapat berupa topik *publish* atau *subscribe*, memungkinkan terjadinya komunikasi *many-to-many* atau banyak-ke-banyak (Hanif & Amri, 2023).

Di sisi lain. Menurut (Saiqul Umam et al., 2023), protokol *MQTT* merupakan sebuah protokol jaringan tipe *Machine to Machine (M2M)* ringan yang berbasis *publish-subscribe* yang dirancang untuk sebuah koneksi dengan keterbatasan spesifikasi mesin atau keterbatasan *bandwidth* jaringan sehingga bisa berjalan pada mesin berdaya rendah dan kepadatan data yang ringan. Protokol *MQTT* juga menerapkan komunikasi 2 arah antara mesin ke jaringan dan sebaliknya.

2.8 Graphical Network Simulator 3 (GNS3)

Menurut (Aeni et al., 2021). *Graphical Network Simulator 3 (GNS3)* merupakan sebuah *software* simulator yang dapat melakukan emulasi jaringan baik jaringan sederhana maupun yang kompleks. Simulator ini mampu mensimulasikan suatu rancangan jaringan sebelum diaplikasikan ke kondisi sebenarnya di lapangan, simulasi menggunakan *software GNS3* dapat bekerja tanpa harus memiliki perangkat jaringan yang asli seperti *router* dan *switch*. Hanya memerlukan *installer* berbasis *image* untuk melakukan simulasi dan dapat bekerja dan berperilaku sebagaimana perangkat jaringan yang sesungguhnya.

2.9 Quick Emulator (QEMU)

Menurut (Ghozali et al., 2022), *QEMU (Quick Emulator)* adalah emulator sumber terbuka dan tidak berbiaya. *QEMU* mampu mengemulasi prosesor komputer melalui terjemahan biner dinamis dan menyediakan serangkaian model perangkat keras dan perangkat yang

berbeda untuk mencapai kecepatan yang layak saat berjalan di arsitektur komputer sebagai *host*. Cara kerjanya adalah melakukan interoperasi dengan *Kernel Virtual Machine (KVM)* untuk menjalankan mesin virtual dengan kecepatan mendekati aslinya. *QEMU* juga dapat melakukan emulasi untuk proses tingkat pengguna, memungkinkan aplikasi yang dikompilasi untuk satu arsitektur dapat dijalankan di arsitektur lain. *QEMU* juga dapat memberikan dukungan percepatan modus campuran *binary translation* untuk *kernel code* dan *native execution* untuk *user code*.

2.10 Wireshark

Menurut (Hasbi & Saputra, 2021), Wireshark adalah *software network packet analyzer* yang digunakan untuk menganalisis paket data pada jaringan dengan fungsi menangkap setiap paket yang melewati jaringan dan dapat menampilkan semua informasi dari masing-masing paket data tersebut dengan detail. Semua jenis paket informasi dalam berbagai format protokol akan dengan mudah ditangkap dan dianalisis.

2.11 Penelitian Terdahulu

Hingga saat ini, sudah banyak penelitian terkait analisis protokol *routing* jaringan berdasarkan parameter *QoS* di jaringan yang tersusun perangkat *IoT*. Penelitian yang sudah dilakukan memiliki berbagai perbedaan seperti pada parameter yang digunakan, teknologi protokol *routing* yang digunakan, *software* simulasi yang digunakan, hingga pada jumlah titik sensor/*node* yang digunakan. Tabel 2.1 memuat beberapa penelitian terdahulu yang membahas tentang analisis protokol *routing* jaringan pada perangkat *IoT*:

Tabel 2.1 Penelitian Terdahulu

No.	Penulis	Judul	Tahun	Keterangan
1.	Agusssalim, Dhian Satria Yudha Kartika, Ani Dijah Rahajoe	Implementasi Ad-Hoc Protocol Pada Tandem Multihop	2023	Penelitian ini melakukan perbandingan protokol <i>routing</i> pada <i>IoT</i> tipe Ad-Hoc yaitu <i>OLSR</i> (<i>Optimized Link State Routing</i>) & <i>BATMAN</i> (<i>Better Approach to Mobile</i>)

No.	Penulis	Judul	Tahun	Keterangan
		Wireless Network		<p><i>Ad-hoc Networking).</i> Variabel yang diujikan adalah <i>Delivery Probability</i>, <i>Average Latency</i>, dan <i>Jitter</i>. Penelitian ini menggunakan <i>software</i> simulasi Mininet-WiFi dengan hasil penelitian yaitu Protokol <i>OLSR</i> & <i>BATMAN</i> terlihat stabil dan berperforma serupa di paket data sebesar 1 MB. Namun, pada paket data sebesar 2 MB, <i>OLSR</i> memiliki performa lebih cepat namun kurang stabil dan <i>BATMAN</i> memiliki performa lebih lambat namun lebih stabil</p>
2.	Alon Jala Tirta Segara, Aditya Wijayanto, Muhamad Azrino Gustalika, Afifah Dwi Ramadhani	Implementasi Mobile Ad-Hoc Network Pada Daerah Pasca Bencana Dengan Protokol DSR	2022	<p>Penelitian ini melakukan implementasi protokol <i>routing</i> pada <i>IoT</i> tipe Ad-Hoc yaitu <i>Dynamic Source Routing (DSR)</i> untuk area rawan bencana. Variabel yang diujikan adalah <i>Packet Delivery Ratio (PDR)</i>, <i>End-to-End Delay</i>, dan <i>Packet Loss</i>. Penelitian ini menggunakan <i>software</i> simulasi NS-2 Simulator dengan hasil penelitian yaitu protokol <i>DSR</i> memberikan peningkatan signifikan pada <i>PDR</i> jaringan dan menurunkan tingkat <i>delay</i> dan <i>packet loss</i> ke angka yang cukup rendah seiring dengan peningkatan jumlah <i>node</i>.</p>

No.	Penulis	Judul	Tahun	Keterangan
3.	Mohamad Lutfi, Primantara Hari Trisnawan, Rakhmadhany Primananda	Implementasi Routing Statis menggunakan Media Komunikasi LoRa dan Websocket untuk Pengiriman Data dari Sensor ke Cloud pada IoT	2021	Penelitian ini melakukan implementasi protokol <i>routing</i> statis pada <i>IoT</i> memanfaatkan LoRa dan <i>Websocket</i> sebagai <i>cloud data</i> . Penelitian dilakukan dengan menggunakan sensor <i>Raspberry Pi</i> , <i>Arduino Nano</i> , dan Modul <i>LoRa SX1278</i> . Variabel yang digunakan adalah <i>Successful Rate</i> , <i>Round Trip Time</i> , dan <i>Delay</i> dengan hasil yaitu sistem dapat berfungsi dengan baik, <i>successful rate</i> dengan angka diatas 90%, semua <i>node</i> memiliki <i>round trip time</i> dibawah 1 detik, dan <i>delay</i> dengan angka yang sangat rendah.
4.	Debbi Mudhoep, Irfan Linawati, Oka Saputra	Kombinasi Protokol Routing OSPF dan BGP dengan VRRP, HSRP, dan GLBP	2021	Penelitian ini mengimplementasikan protokol <i>routing</i> dinamis <i>OSPF</i> dan <i>BGP</i> dan protokol redundansi <i>gateway</i> <i>VRRP</i> , <i>HSRP</i> , dan <i>GLBP</i> pada perangkat jaringan komputer tradisional. Variabel yang digunakan adalah <i>Packet Loss</i> , <i>Throughput</i> , dan <i>Delay</i> dengan hasil yaitu protokol <i>routing</i> dinamis <i>OSPF</i> , protokol redundansi <i>gateway</i> <i>VRRP</i> performanya lebih baik. Kemudian, kombinasi <i>OSPF</i> dan <i>VRRP</i> menghasilkan kualitas jaringan yang paling baik berdasarkan parameter yang diujikan.

No.	Penulis	Judul	Tahun	Keterangan
5.	Md. Humayun Kabir, Md. Ahsan Kabir, Md. Saiful Islam, Mohammad Golam Mortuza, Mohammad Mohiuddin	Performance Analysis of Mesh Based Enterprise Network Using RIP, EIGRP and OSPF Routing Protocols	2021	Penelitian ini melakukan komparasi 3 protokol <i>routing</i> yaitu <i>Routing Information Protocol (RIP)</i> , <i>Open Shortest Path First (OSPF)</i> dan <i>Enhanced Interior Gateway Routing Protocol (EIGRP)</i> di <i>software</i> simulasi jaringan <i>Cisco Packet Tracer</i> dengan penekanan menggunakan topologi <i>full</i> dan <i>half mesh network</i> antar <i>router</i> pada <i>WSN (Wireless Sensor Network)</i> dengan data yang diujikan yaitu <i>voice packet</i> , <i>HTTP packet</i> , dan <i>video Packet</i> . Variabel yang dipilih adalah <i>end-to-end latency</i> , <i>ping time</i> , dan <i>throughput</i> . Hasil penelitiannya menunjukkan protokol <i>routing OSPF</i> memiliki performa terbaik dibandingkan protokol <i>routing</i> lainnya yang diujikan dan <i>RIP</i> memiliki performa terburuk
6.	José Luis García-Navas, Laura Garcia, Oscar Romero, Jaime Lloret, Pascal Lorenz	Comparative Study of RIP, OSPF and EIGRP Protocols to Manage WSN-IoT Traffic vs IPTV Traffic Using Cisco	2021	Melakukan komparasi 3 protokol <i>routing</i> yaitu <i>Routing Information Protocol (RIP)</i> , <i>Open Shortest Path First (OSPF)</i> dan <i>Enhanced Interior Gateway Routing Protocol (EIGRP)</i> di <i>software</i> simulasi jaringan <i>Cisco Packet Tracer</i> pada beberapa skenario dengan skenario pertama yaitu di perangkat <i>IoT</i> dan di

No.	Penulis	Judul	Tahun	Keterangan
		Packet Tracer		komputer tradisional. Lalu, skenario kedua pada perangkat <i>IoT</i> dan Komputer tradisional serta pada perangkat yang mendukung <i>Internet Protocol Television (IPTV)</i> . Variabel yang diujikan adalah <i>Round Trip Time (RTT)</i> dengan hasil penelitian yaitu untuk skenario yang menggunakan perangkat komputer tradisional dan perangkat <i>IoT</i> tanpa perangkat yang mendukung <i>IPTV</i> , protokol <i>routing RIP</i> memiliki performa terbaik dan <i>EIGRP</i> memiliki performa terburuk. Untuk skenario dengan tambahan perangkat <i>IPTV</i> , protokol <i>routing EIRGP</i> memiliki performa terbaik dan <i>RIP</i> terburuk. <i>OSPF</i> memiliki performa seimbang di 2 skenario pengujian.
7.	Cristian P. Saavedra Cañar, John J. Tucker Yépez, Heydi M. Roa López	Performance of Reactive Routing Protocols DSR and AODV in Vehicular Ad-Hoc Networks Based on Quality of Service (QoS) Metrics	2020	Penelitian ini melakukan pengujian berbagai protokol <i>routing IoT</i> berbasis VANET yaitu <i>Ad-hoc On-Demand Distance Vector Routing (AODV)</i> dan <i>Dynamic Source Routing (DSR)</i> . Variabel yang dipakai adalah <i>Throughput</i> , <i>Delay</i> , dan <i>Jitter</i> . Penelitian dilakukan dengan <i>software NS-3 Simulator</i> dengan hasil yaitu protokol <i>DSR</i> memiliki hasil yang lebih baik pada jumlah titik jaringan

No.	Penulis	Judul	Tahun	Keterangan
				yang lebih rendah, sebaliknya <i>AODV</i> memiliki hasil yang lebih baik pada jumlah titik jaringan yang lebih tinggi, sehingga <i>AODV</i> disimpulkan lebih baik untuk jaringan yang berkelanjutan atau akan berkembang di kemudian hari.

2.12 Perbedaan Penelitian

Perbedaan penelitian penulis dengan penelitian terdahulu yang telah dibahas pada tabel 2.1, yaitu pada penelitian ini, protokol *routing* statis, *RIP*, dan *OSPF* akan diterapkan pada perangkat berbasis *IoT* berbentuk virtual dan fisik. Selain itu, penulis juga menggunakan teknologi yang tidak digunakan di beberapa penelitian terdahulu seperti penggunaan *QEMU*, *MQTT*, dan *software GNS3* bersamaan dengan protokol *routing* yang akan diujikan. Selanjutnya, penelitian penulis lebih banyak menggunakan parameter *QoS* yang berbeda-beda dibandingkan dengan penelitian sebelumnya. Berbeda dengan penelitian yang dilakukan oleh Luis García-Navas et al. (2021) dan Kabir et al. (2021) yang memanfaatkan *software Cisco Packet Tracer* dengan parameter *QoS* yang lebih sedikit serta tidak memanfaatkan teknologi *QEMU* dan *MQTT*. Penelitian penulis juga berbeda dengan penelitian yang dilakukan oleh Mudhoep et al. (2021) yang hanya menggunakan protokol jaringan *OSPF* dan *BGP* tanpa menggunakan perangkat pengguna akhir yang berbasis *IoT*.

BAB 3

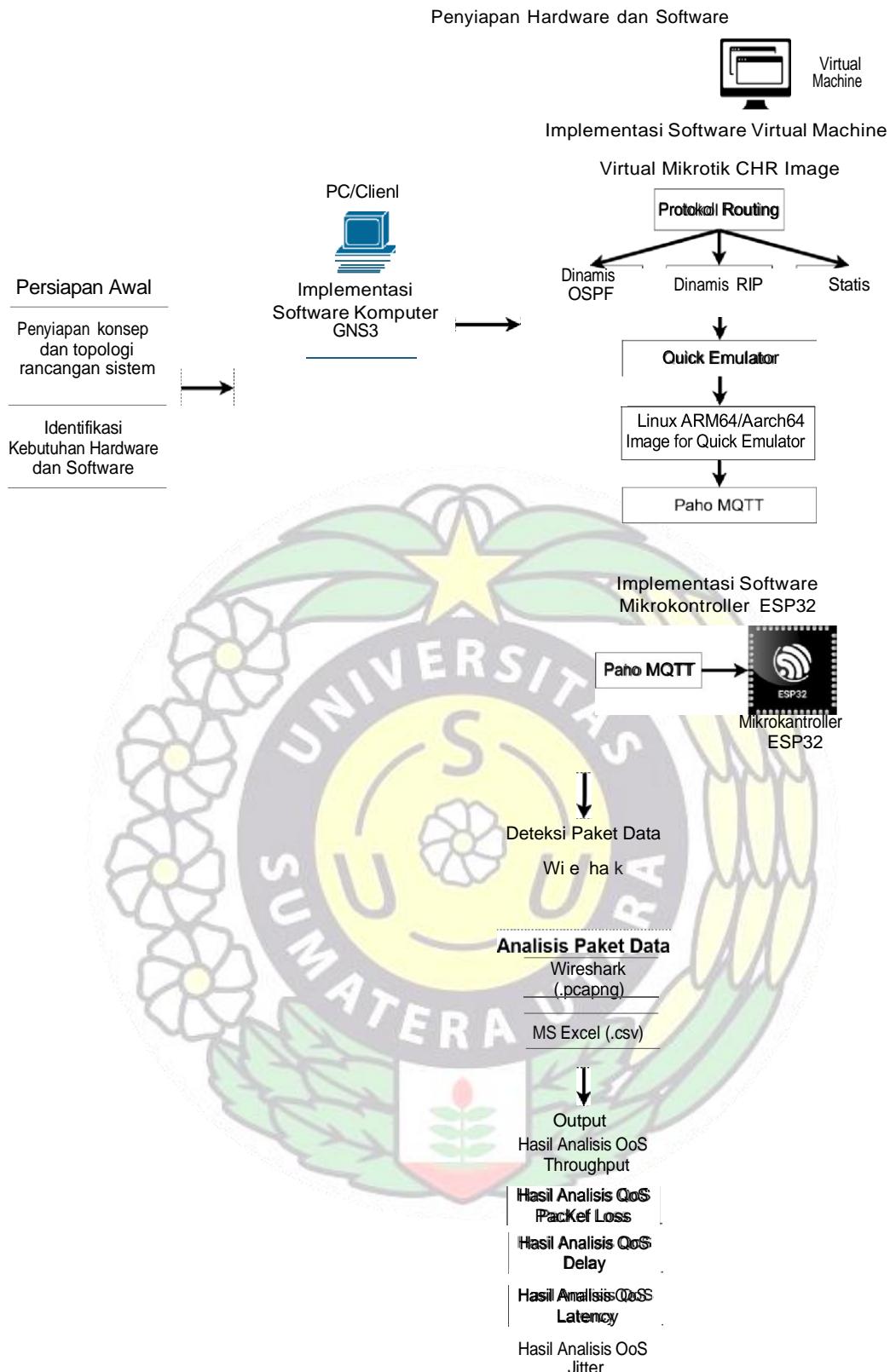
ANALISIS DAN PERANCANGAN SISTEM

3.1 Data Yang Digunakan

Data yang digunakan dalam penelitian ini yaitu data sensor dan data *string* dari perangkat *IoT*. Data tersebut akan dialirkan antar *node* untuk menghasilkan data lainnya yaitu data hasil penangkapan paket oleh *software Wireshark*. Data dari *Wireshark* akan dipelajari dan dianalisis berdasarkan parameter *Quality of Service* yaitu parameter *delay*, *jitter*, *latency*, *throughput*, dan *packet loss*. Analisis dilakukan ke seluruh protokol *routing* yang diujikan yaitu protokol *routing* statis, dinamis *RIP*, dan dinamis *OSPF*, serta dilakukan ke jalur-jalur data yang terpilih dari seluruh jalur data yang tersedia. Tujuan dari analisis data ini yaitu untuk memahami dan mengetahui kemampuan setiap *routing* jaringan dalam menghadapi transfer data antar perangkat berbasis *IoT*.

3.2 Arsitektur Umum Sistem

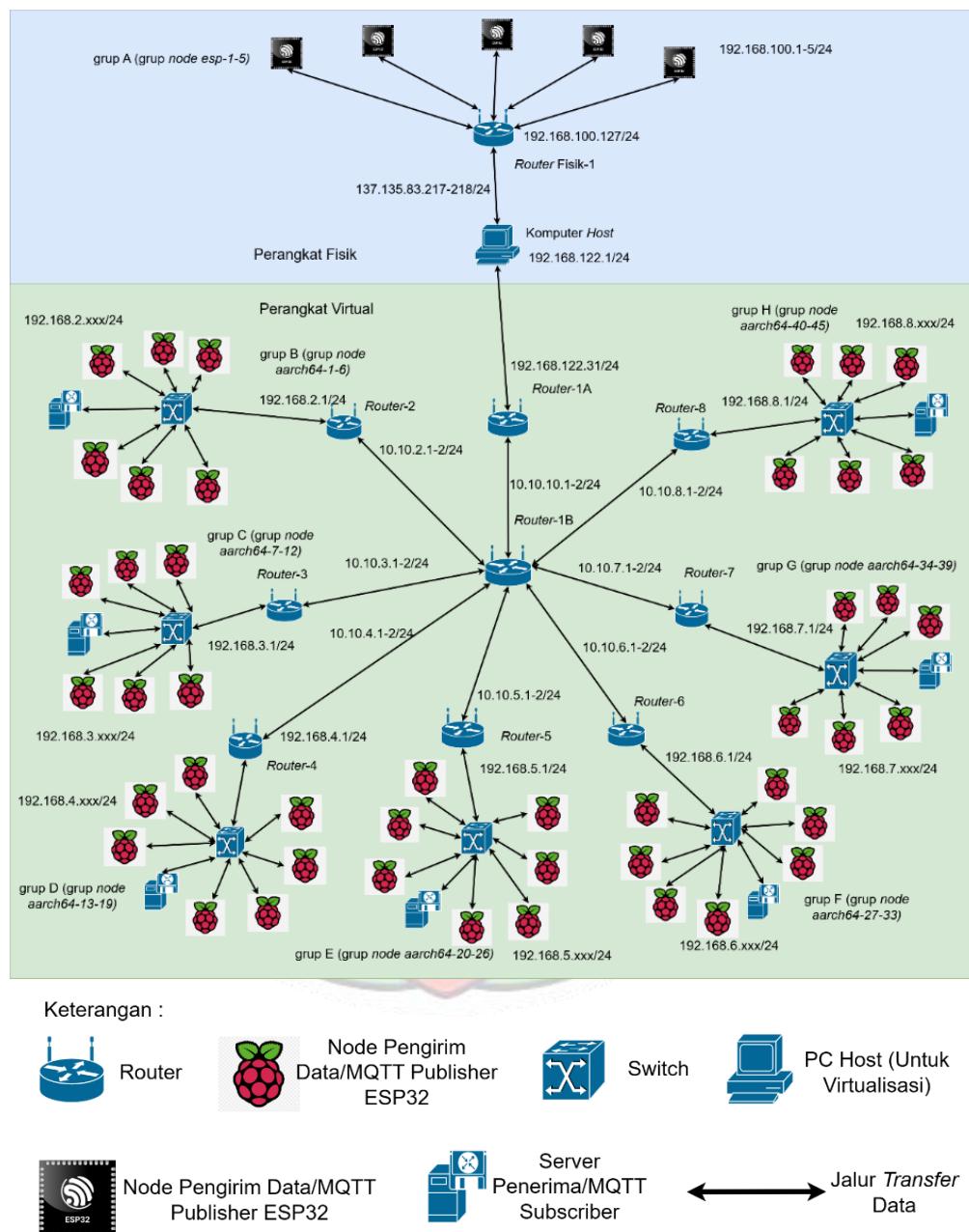
Arsitektur umum memuat gambaran terkait proses yang diperlukan untuk melakukan penelitian berupa pengujian *Quality of Service* pada protokol *routing* seperti yang telah direncanakan. Arsitektur umum sistem pada penelitian yang akan dibangun ini diilustrasikan pada gambar 3.1 dan penjelasan terkait tahapan-tahapan pada arsitektur umum tersebut terdapat pada sub-bab selanjutnya yaitu sub-bab 3.2.1 sampai dengan 3.2.6 yang menampilkan rancangan proses yang akan dijalankan untuk penelitian ini.



Gambar 3.1 Arsitektur Umum

3.2.1 Persiapan Awal

Tahap ini melakukan berbagai persiapan yang dibutuhkan untuk melakukan penelitian. Persiapan terdiri dari identifikasi alat baik *hardware* maupun *software*, topologi sistem, alamat *IP address* perangkat, dan protokol *routing* yang akan digunakan. Topologi yang dipilih adalah topologi jaringan *star* karena cukup mudah untuk melakukan konfigurasi di topologi jaringan tersebut. Berikut adalah rancangan topologi program setelah penggunaan topologi *star* yang diilustrasikan pada gambar 3.2 :



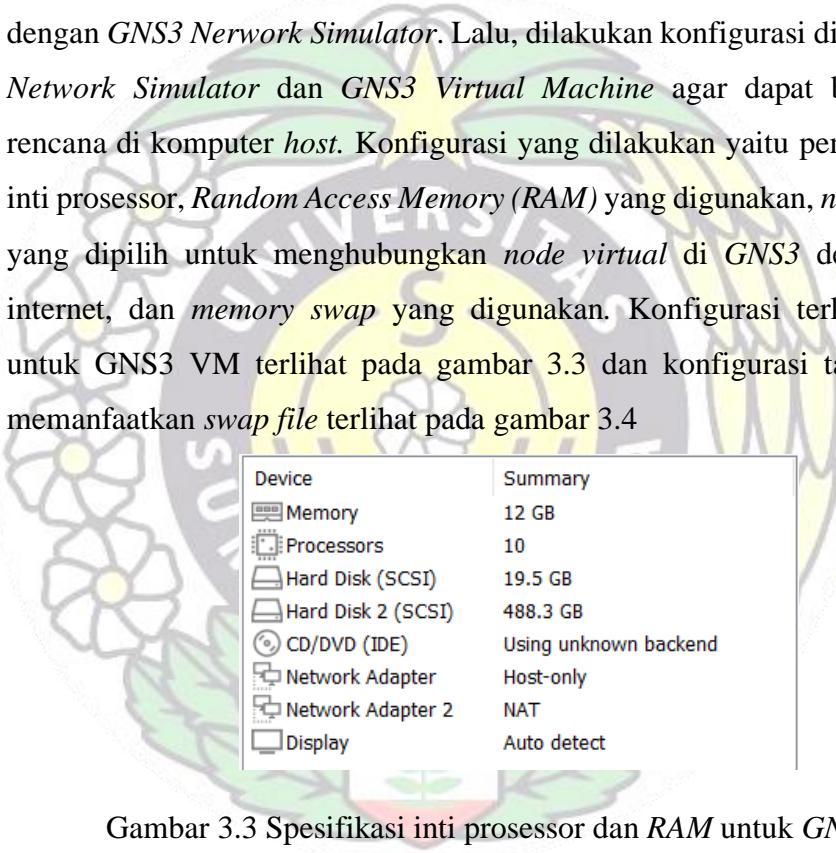
Gambar 3.2 Bentuk Topologi Jaringan

3.2.2 Penyiapan Hardware dan Software

Tahap ini merupakan tahap untuk melakukan konfigurasi berbagai peralatan *hardware* dan *software* untuk melaksanakan penelitian. Konfigurasi peralatan sendiri terbagi ke dalam 3 tahapan yaitu konfigurasi *Personal Computer (PC)* untuk *host*, lalu konfigurasi *virtual machine* di komputer *host* untuk menyiapkan *node IoT* virtual yang akan terintegrasi dengan aplikasi *software* di komputer *host*, lalu yang terakhir adalah konfigurasi mikrokontroller *ESP32* yang akan mewakili *node IoT* bentuk fisik.

3.2.2.1 Konfigurasi Software Personal Computer (PC) Host

Konfigurasi dilakukan dengan melakukan instalasi *software* yaitu *GNS3 Network Simulator*, *GNS3 Virtual Machine*, dan *Wireshark* yang sudah satu paket dengan *GNS3 Network Simulator*. Lalu, dilakukan konfigurasi di aplikasi *GNS3 Network Simulator* dan *GNS3 Virtual Machine* agar dapat berjalan sesuai rencana di komputer *host*. Konfigurasi yang dilakukan yaitu penentuan jumlah inti prosessor, *Random Access Memory (RAM)* yang digunakan, *network adapter* yang dipilih untuk menghubungkan *node virtual* di *GNS3* dengan jaringan internet, dan *memory swap* yang digunakan. Konfigurasi terkait spesifikasi untuk *GNS3 VM* terlihat pada gambar 3.3 dan konfigurasi tambahan *RAM* memanfaatkan *swap file* terlihat pada gambar 3.4



Device	Summary
Memory	12 GB
Processors	10
Hard Disk (SCSI)	19.5 GB
Hard Disk 2 (SCSI)	488.3 GB
CD/DVD (IDE)	Using unknown backend
Network Adapter	Host-only
Network Adapter 2	NAT
Display	Auto detect

Gambar 3.3 Spesifikasi inti prosessor dan *RAM* untuk *GNS3 VM*

```

gns3@gns3vm:~$ free -h
              total        used        free      shared  buff/cache   available
Mem:       11Gi       481Mi      10Gi      1.0Mi      612Mi      10Gi
Swap:      19Gi          0B      19Gi
gns3@gns3vm:~$ swapon --show
NAME           TYPE SIZE USED PRIO
/swapfile_extend_10GB_1 file 10G  0B  -3
/swap.img       file  2G  0B  -2
/opt/swap_test   file  8G  0B  -4
gns3@gns3vm:~$
```

Gambar 3.4 Tambahan *RAM* memanfaatkan *swap file* untuk *GNS3 VM*

3.2.2.2 Konfigurasi Software Virtual Machine

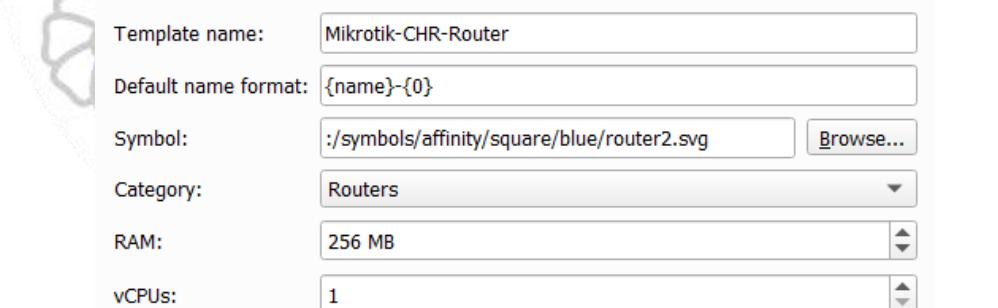
Konfigurasi dilakukan setelah konfigurasi di *GNS3 virtual machine* sudah bekerja dengan baik dan integrasi dengan *GNS3 Network Simulator* berjalan tanpa kendala. Lalu, dilanjutkan dengan melakukan instalasi berbagai *software* yang diperlukan di *GNS3 Virtual Machine*.

Software pertama yang diinstal yaitu MikroTik *Cloud Hosted Router (CHR)* image beserta seluruh konfigurasi ketiga protokol *routing* jaringan yang akan digunakan yaitu *routing* statis, dinamis *RIP*, dan dinamis *OSPF*.

Software kedua yang diinstal yaitu *Quick Emulator (QEMU)* untuk melakukan virtualisasi dengan perangkat yang berbeda arsitektur prosessor nya beserta *Operation System (OS) Linux ARM64/Aarch64* untuk *Quick Emulator*.

Software ketiga dan diinstalasi setelah *QEMU* dan *OS Linux ARM64/Aarch64* dapat berkerja dengan baik yaitu instalasi *software* Paho *Message Queue Telemetry Protocol (Paho-MQTT)* untuk sistem *Linux ARM64/Aarch64* beserta seluruh pendukungnya. *Software Paho-MQTT* memiliki fungsi agar perangkat *IoT* dapat saling mengirim dan menerima data antar perangkat *IoT* yang ada di sistem yang telah dibuat.

Spesifikasi inti prosessor dan *RAM* untuk MikroTik *CHR* terlihat pada gambar 3.5 dan spesifikasi *RAM* dan inti prosessor untuk *QEMU* terlihat pada gambar 3.6 dan gambar 3.7



Gambar 3.5 Spesifikasi inti prosessor dan *RAM* untuk MikroTik *CHR*

```
root@debian:~# free
              total        used        free      shared  buff/cache   available
Mem:       226624       40020     131292          336      62868    186604
Swap:          0          0          0
```

Gambar 3.6 Spesifikasi *RAM* untuk *QEMU*

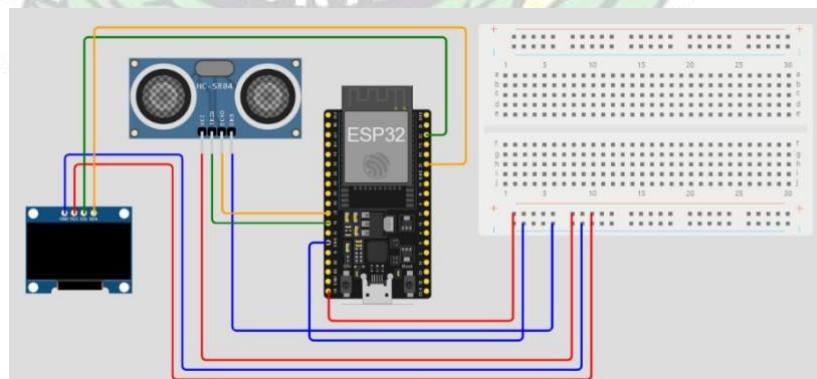
```
root@debian:~# lscpu
Architecture:           aarch64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:             Little Endian
CPU(s):                 1
On-line CPU(s) list:   0
Vendor ID:              ARM
Model name:             Cortex-A57
Model:                  0
Thread(s) per core:    1
Core(s) per cluster:   1
```

Gambar 3.7 Spesifikasi inti prosessor untuk *QEMU*

3.2.2.3 Konfigurasi Mikrokontroller ESP32

Konfigurasi diawali dengan melakukan perakitan perangkat untuk mikrokontroller *ESP32* sesuai dengan rancangan yang sudah direncanakan seperti yang ditunjukkan pada gambar 3.8, Rancangan perakitan digunakan untuk mikrokontroller *ESP32* yang menggunakan sensor yaitu mikrokontroller *ESP32-1* dan *ESP32-2*. Selanjutnya, untuk semua mikrokontroller dari *ESP32-1* sampai *ESP32-5* dilakukan instalasi berbagai *software* yang diperlukan di titik mikrokontroller *ESP32*.

Software yang diinstalasi di mikrokontroller *ESP32* setelah perakitan siap dilakukan yaitu *software Paho Message Queue Telemetry Protocol (Paho-MQTT)* untuk sistem mikrokontroller *ESP32* beserta seluruh pendukungnya. *Software Paho-MQTT* memiliki fungsi sebagai platform yang akan digunakan oleh perangkat *IoT* untuk saling mengirim dan menerima data antar perangkat *IoT* yang ada di sistem yang telah dibuat.



Gambar 3.8 Rangkaian detail sambungan kabel pada *ESP32-1* dan *ESP32-2*

3.2.3 Penentuan Alamat IP Address

Tahap ini merupakan tahap untuk melakukan penentuan alamat *IP (Internet Protocol) Address* yang lebih jelas dari yang ditunjukkan di gambar 3.2, alamat *IP* akan disusun dalam tabel *IP Address* pada tabel 3.1

Tabel 3.1 Tabel IP Address Rancangan Sistem

No.	Grup	Jumlah Node	Range IP Address
1.	Grup A	5 node (<i>esp-1</i> sampai <i>esp-5</i>)	192.168.100.1-5/24
2.	<i>Router fisik-1</i>	1 node, 2 interface berbeda <i>IP Address</i>	192.168.100.127/24 (<i>interface grup A</i>) 137.135.83.217/24 (<i>interface komputer host</i>)
3.	<i>Komputer host</i>	1 node, 2 interface berbeda <i>IP Address</i>	192.168.100.128/24 (<i>interface Router fisik-1</i>) 192.168.122.1/24 (<i>interface Router-1A</i>)
4.	<i>Router-IA</i>	1 node, 2 interface berbeda <i>IP Address</i>	192.168.122.31/24 (<i>interface Komputer Host</i>) 10.10.10.1/24 (<i>interface Router-1B</i>)
5.	<i>Router-IB</i>	1 node, 8 interface berbeda <i>IP Address</i>	10.10.10.2/24 (<i>interface Router-1B</i>) 10.10.2.1/24 (<i>interface Router-2</i>) 10.10.3.1/24 (<i>interface Router-3</i>) 10.10.4.1/24 (<i>interface Router-4</i>) 10.10.5.1/24 (<i>interface Router-5</i>) 10.10.6.1/24 (<i>interface Router-6</i>) 10.10.7.1/24 (<i>interface Router-7</i>) 10.10.8.1/24 (<i>interface Router-8</i>)
6.	<i>Router-2</i>	1 node, 2 interface berbeda <i>IP Address</i>	10.10.2.2/24 (<i>interface Router-1B</i>) 192.168.2.1/24 (<i>interface Grup B</i>)
7.	Grup B	7 node (<i>subscriber-1</i> , <i>aarch64-1</i> sampai <i>aarch64-6</i>)	192.168.2.2-8/24

No.	Grup	Jumlah Node	Range IP Address
8.	Router-3	1 node, 2 interface berbeda IP Address	10.10.3.2/24 (interface Router-1B) 192.168.3.1/24 (interface Grup C)
9.	Grup C	7 node (subscriber-2, aarch64-7 sampai aarch64-12)	192.168.3.2-8/24
10.	Router-4	1 node, 2 interface berbeda IP Address	10.10.4.2/24 (interface Router-1B) 192.168.4.1/24 (interface Grup D)
11.	Grup D	8 node (subscriber-3, aarch64-13 sampai aarch64-19)	192.168.4.2-9/24
12.	Router-5	1 node, 2 interface berbeda IP Address	10.10.5.2/24 (interface Router-1B) 192.168.5.1/24 (interface Grup E)
13.	Grup E	8 node (subscriber-4, aarch64-20 sampai aarch64-26)	192.168.5.2-9/24
14.	Router-6	1 node, 2 interface berbeda IP Address	10.10.6.2/24 (interface Router-1B) 192.168.6.1/24 (interface Grup F)
15.	Grup F	8 node (subscriber-5, aarch64-27 sampai aarch64-33)	192.168.6.2-9/24
16.	Router-7	1 node, 2 interface berbeda IP Address	10.10.7.2/24 (interface Router-1B) 192.168.7.1/24 (interface Grup G)
17.	Grup G	7 node (subscriber-6, aarch64-34 sampai aarch64-39)	192.168.7.2-8/24
18.	Router-8	1 node, 2 interface berbeda IP Address	10.10.8.2/24 (interface Router-1B) 192.168.8.1/24 (interface Grup H)
19.	Grup H	7 node (subscriber-7, aarch64-40 sampai aarch64-45)	192.168.8.2-8/24

3.2.4 Deteksi Paket Data

Deteksi paket data dilakukan dengan mengaktifkan *software Wireshark* diantara konfigurasi jalur data yang tersedia di *GNS3 Network Simulator* dengan lama waktu tertentu dan dilakukan pada seluruh protokol *routing* yang diujikan satu-persatu. Jalur data yang dipilih adalah jalur data antara *node router* akses internet dengan *node router* utama, dan diantara *node router* utama dengan beberapa *node router* cabang. Setelah deteksi paket data selesai dilakukan, diberikan perlakuan tertentu yang digunakan saat proses analisis paket data. Setiap parameter *QoS* yang diujikan mendapatkan perlakuan yang berbeda pula. Tabel 3.2 menunjukkan perlakuan yang diberikan setelah proses deteksi paket data telah dilakukan

Tabel 3.2 Perlakuan Setelah Deteksi Paket Data Selesai Dilaksanakan

No.	Parameter <i>QoS</i>	Perlakuan
1.	<i>Throughput</i>	Digunakan fitur statistik dari <i>software Wireshark</i> yaitu “ <i>Capture file properties</i> ” yang memuat angka <i>Throughput</i> dari hasil deteksi paket data.
2.	<i>Packet Loss</i>	Digunakan fitur <i>packet search</i> dengan memasukkan baris perintah <code>tcp.analysis.retransmission //</code> <code>tcp.analysis.lost_segment //</code> <code>tcp.analysis.duplicate_ack //</code> <code>tcp.analysis.out_of_order</code> untuk menampilkan hasil deteksi yang menunjukkan <i>packet loss</i> , dilanjutkan dengan menampilkan statistik menggunakan “ <i>Capture file properties</i> ” yang memuat angka <i>Packet Loss</i> dari hasil deteksi paket data.
3.	<i>Delay</i>	Digunakan fitur <i>packet search</i> dengan memasukkan baris perintah <code>_ws.col.info == "Publish Message [IoT/sub/device/aarch64-xx]"</code> untuk <i>node</i> virtual dan <code>_ws.col.info == "Publish Message [IoT/sub/device/esp-x]"</code> untuk <i>node</i> fisik. Kemudian, dilakukan pelacakan sumber paket dan perhitungan perbandingan waktu antara paket dikirim dan diterima untuk <i>node</i> virtual dan <i>node</i> fisik.

No.	Parameter <i>QoS</i>	Perlakuan
4.	<i>Latency</i>	Perhitungan <i>Latency</i> menggunakan hasil hitung dari rerata <i>delay</i> . Sehingga, proses hitung <i>Latency</i> dilakukan setelah proses hitung <i>Delay</i> telah dilakukan
5.	<i>Jitter</i>	Perhitungan <i>Jitter</i> menggunakan hasil hitung dari perhitungan <i>delay</i> setiap <i>node</i> yang dipilih untuk perhitungan <i>delay</i> . Sehingga, proses hitung <i>Latency</i> dilakukan setelah proses hitung <i>Delay</i> telah dilakukan

3.2.5 Analisis Paket Data

Analisis paket data dilakukan setelah deteksi paket data telah selesai dilakukan dan telah didapatkan *file* hasil uji deteksi data dalam format *file .pcapng*. *File .pcapng* ini akan dianalisis berdasarkan parameter *Quality of Service* dengan bantuan alat dari *Software Wireshark* dan menggunakan rumus-rumus yang dapat digunakan untuk menghitung parameter *Quality of Service* tersebut. Pada saat analisis paket data dilaksanakan, bentuk format data dan topik *MQTT* yang digunakan dapat terlihat melalui alat milik *Wireshark*. Tabel 3.3 menampilkan format data dan topik *MQTT* yang digunakan setiap *node* oleh sistem.

Tabel 3.3 Format Data dan Topik *MQTT* yang Digunakan Setiap *Node*

No.	node	Topik <i>MQTT</i>	Format Data
1.	<i>esp32-1</i> sampai <i>esp32-2</i>	IoT/sub/device/esp-x + client_id mqtt_IoT_ESP32_x	<i>Float</i> dengan 224 decimal <i>value</i>
2.	<i>esp32-3</i> sampai <i>esp32-5</i>	IoT/sub/device/esp-x + client_id mqtt_IoT_ESP32_x	<i>String</i> dengan 35 kata dan 298 karakter
3.	<i>aarch64-1</i> sampai <i>aarch64-45</i>	IoT/sub/device/aarch64- xx	<i>String</i> dengan 368 kata dan 2559 karakter
4.	<i>Subscriber-1</i> sampai <i>subscriber-7</i>	IoT/sub/device/#	<i>String</i> , dengan data diterima dari <i>node publisher</i> lain

3.2.6 Keluaran Akhir

Hasil akhir dari penelitian ini yaitu hasil dari perhitungan setiap parameter *Quality of Service* yang diujikan yaitu *throughput*, *packet loss*, *delay*, *latency*, dan *jitter* untuk setiap protokol *routing* yang telah diujikan yaitu protokol *routing* statis, dinamis *RIP*, dan dinamis *OSPF* dengan data hasil ditampilkan dalam bentuk tabel dan data telah melalui tahap perlakuan khusus seperti yang telah disampaikan di subbab sebelumnya. Data dari hasil akhir diambil dari hasil pengujian di setiap jalur jaringan yang terpilih, yaitu jalur jaringan yang merupakan pusat dari seluruh data yang datang dan meninggalkan sistem pengujian dan jalur jaringan tepi yaitu jalur yang dekat dengan *node publisher* dan *node subscriber* yang diujikan. Masing-masing jalur jaringan tersebut diambil 1 buah contoh untuk pengujian.



BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Sistem

Implementasi sistem Analisis Quality of Service (QoS) Menggunakan Routing Statis dan Dinamis Pada Jaringan Internet of Things (IoT) adalah sebagai berikut.

4.1.1 Perangkat Keras dan Perangkat Lunak

Berikut detail perangkat keras dan perangkat lunak yang dipakai untuk menjalankan penelitian ini.

1. Laptop Acer Aspire 5 A515-45-R3TY N18Q13
2. Processor AMD^R RyzenTM 5 5500U 2.10Ghz
3. GPU AMD RadeonTM Graphics 512MB
4. AMD Virtualization AMD-VTM
5. Storage SSD 1TB (512GB NVMe, 512GB SATA)
6. RAM 16GB
7. Sistem operasi Windows 11 Home Single Language 64-Bit
8. VMWare Workstation 17 Player
9. GNS3 2.2.46 & GNS3 VM 2.2.46
10. Wireshark 4.2.4
11. Python 3.9.2
12. Paho MQTT 1.6.6
13. Visual Studio Code 1.92.2
14. NodeMCU ESP32-WROOM-32E
15. Arduino IDE 2.2.1

4.1.2 Alat

Berikut detail alat yang dipakai untuk menjalankan penelitian ini.

1. Mikrokontroller NodeMCU ESP32-WROOM-32E
2. Sensor ultrasonik HC-SR04

3. *Breadboard*
4. Kabel *jumper (male to male, male to female)*
5. Kabel *USB*

4.2 Implementasi Program

Implementasi program pada penelitian ini terbagi menjadi 3 tahap, yaitu implementasi program di komputer *host*, di *virtual machine*, dan di mikrokontroller. Berikut tahapan prosesnya.

4.2.1 Implementasi Program di Komputer Host

Tahap implemenetasi dan penggeraan pada bagian ini diawali dengan melakukan instalasi *software* berikut :

- *GNS3*
- *GNS3 Virtual Machine (GNS3 VM)*
- *Wireshark*

Instalasi *software* tersebut diperlukan karena akan menjadi tempat peletakan dan konfigurasi untuk *node* virtual dan menjadi alat untuk melakukan pendektsian dan analisis paket data.

4.2.2 Implementasi Program di Virtual Machine

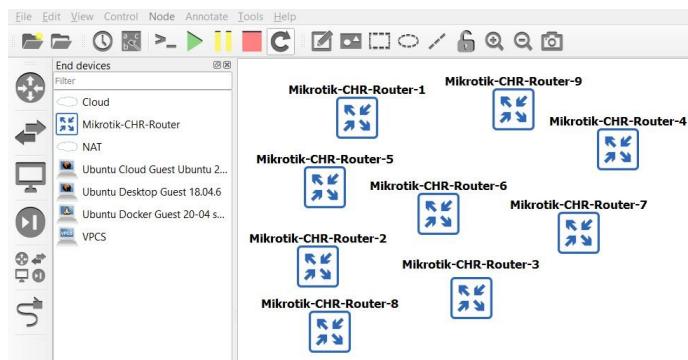
Tahap implemenetasi dan penggeraan selanjutnya dilakukan pada bagian *GNS3 Virtual Machine (GNS3 VM)* dengan memanfaatkan *software GNS3* sebagai antarmuka penghubungnya. Bagian implementasi ini terbagi menjadi beberapa tahapan, yaitu :

4.2.2.1 Implementasi Router Mikrotik CHR (Cloud Hosted Router)

Implementasi dan konfigurasi *Mikrotik CHR (Cloud Hosted Router)* dilakukan agar masing-masing titik jaringan dapat mengirim dan menerima data yang telah melewati jaringan yang berbeda-beda, implementasi *Mikrotik CHR (Cloud Hosted Router)* juga diperlukan karena ketiga protokol *routing* yang akan diujikan yaitu protokol *routing* statis, dinamis *RIP*, dan dinamis *OSPF* akan dipasangkan implementasinya di *router* tersebut.

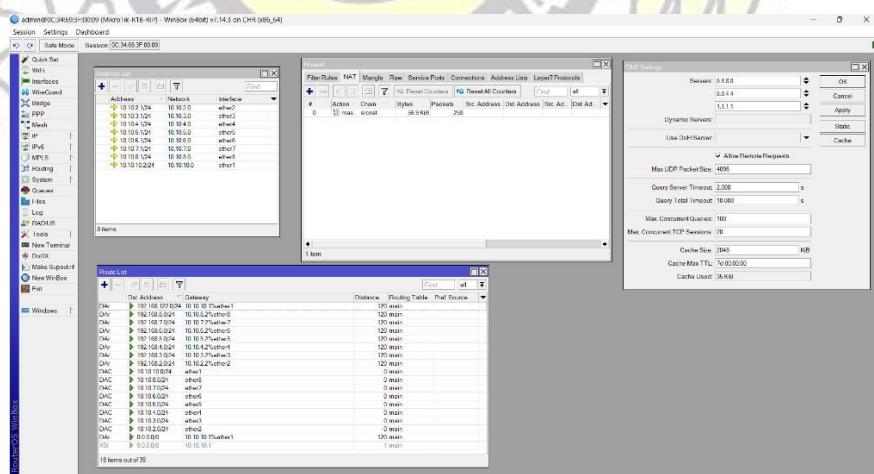
Implementasi diawali dengan mendapatkan *file* berekstensi *.img* yang mewakili *software router* tersebut agar dapat dimasukkan ke *Virtual Machine*.

Setelah *file* tersebut didapatkan, lakukan instalasi *file* ke *GNS3 VM* dengan memanfaatkan *GNS3* sebagai antarmuka. Lalu, masukkan *node mikrotik router* ke bidang kerja di *GNS3* sesuai kebutuhan dan *node* tersebut dihidupkan.



Gambar 4.1 Memasukkan *Mikrotik CHR* ke Bidang Kerja *GNS3*

Kemudian, dilakukan konfigurasi *router* menggunakan bantuan *software winbox*. Konfigurasi yang dilakukan yaitu konfigurasi dasar seperti konfigurasi *IP address*, *DNS server*, *DHCP server*, dan *NAT firewall*. Setelah itu, dilakukan juga konfigurasi protokol untuk *routing* yaitu konfigurasi protokol *routing* statis, dinamis *RIP*, dan dinamis *OSPF*. Protokol yang telah dikonfigurasi tidak diaktifkan semua dalam satu waktu. Namun, hanya satu protokol *routing* yang aktif dalam satu waktu. Gambar 4.6 menunjukkan hasil dari konfigurasi *routing* melalui *software Winbox*. Konfigurasi serupa juga dilakukan untuk semua *router* lainnya.



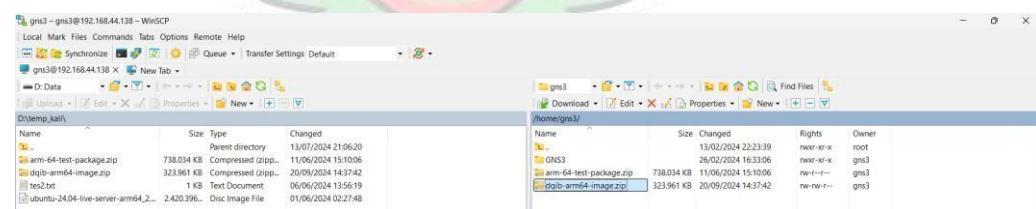
Gambar 4.2 Hasil Konfigurasi Dilihat Melalui *Software Winbox*

4.2.2.2 Implementasi Node MQTT Publisher

Implementasi dan konfigurasi *node MQTT Publisher* dilakukan agar masing-masing titik jaringan yang dimaksud dapat mengirim data menggunakan protokol *MQTT* dan mengirimkan data tersebut dengan perantara protokol *routing* yang akan diujikan serta diupayakan agar titik jaringan ini memiliki perilaku sistem seperti perangkat *IoT*, dalam hal ini dipilih perangkat *IoT* berbasis arsitektur *ARM64* yang dapat dijumpai di perangkat seperti *Raspberry pi*, *Orange pi*, dan sejenisnya.

Implementasi menggunakan *node Ubuntu Docker Guest* serta melibatkan *software* yaitu *Quick Emulator (QEMU)* dan *Linux ARM64* yang diperlukan untuk memungkinkan terjadinya simulasi *node* berbasis *Internet of Things (IoT)*, dengan kemampuan *QEMU* yang dapat menjalankan simulasi sistem berdasarkan arsitektur prosessor jenis lain. Sehingga, menjadikan *QEMU software* yang dibutuhkan untuk penelitian ini. Pada *QEMU* akan mensimulasikan perangkat dengan arsitektur prosessor *ARM64/Aarch64* yang umum digunakan di perangkat *IoT* lainnya seperti *Raspberry pi*, *Orange pi*, dan sejenisnya. Karena sistem akan berjalan sebagai perangkat *ARM64*, maka dibutuhkan sistem operasi *linux* berbasis *ARM64*, di penelitian ini digunakan sistem operasi *Debian* untuk sistem *ARM64*.

Implementasi diawali dengan mendapatkan *folder* berekstensi *.zip* yang berisi segala macam *file* yang dibutuhkan untuk melakukan simulasi *QEMU*. Salah satu penyedia berbasis terbuka yang menyediakan pengembangan *folder* ini adalah *Debian Quick Image Baker (DQIB)*. Setelah *folder* *.zip* didapatkan, *folder* tersebut akan dikirim ke *GNS3 Virtual Machine* menggunakan bantuan *software* *WinSCP*.



Gambar 4.3 Memindahkan *Folder DQIB* ke *GNS3 VM*

Setelah dilakukan pemindahan *folder* ke *GNS3 VM*, maka selanjutnya dilakukan pemindahan *folder* tersebut dari *GNS3 VM* ke *node* virtual yang ada di *dashboard GNS3*.

```

gns3@gns3vm:~/opt/gns3/projects/69eb1883-ee38-4048-8194-823d501e1fea/project-files/docker/0f05da8c-b921-41ec-9777-a6bac6767278/gns3volumes/root
gns3@gns3vm:~$ ls -l
total 1062012
-rw-r--r-- 1 gns3 gns3 755746389 Jun 11 08:10 arm-64-test-package.zip
-rw-rw-r-- 1 gns3 gns3 331735334 Sep 20 07:37 dqib-arm64-image.zip
drwxr-xr-x 5 gns3 gns3 4096 Feb 26 2024 GNS3
gns3@gns3vm:~$ cp -r dqib-arm64-image.zip /opt/gns3/projects/69eb1883-ee38-4048-8194-823d501e1fea/project-files/docker/0f05da8c-b921-41ec-9777-a6bac6767278/gns3volumes/root
gns3@gns3vm:~/opt/gns3/projects/69eb1883-ee38-4048-8194-823d501e1fea/project-files/docker/0f05da8c-b921-41ec-9777-a6bac6767278/gns3volumes/root$ ls -l
total 323972
drwxrwxr-x 2 gns3 gns3 4096 Jun 10 08:48 arm-64-test
-rw-rw-r-- 1 gns3 gns3 331735334 Sep 20 08:01 dqib-arm64-image.zip
gns3@gns3vm:/opt/gns3/projects/69eb1883-ee38-4048-8194-823d501e1fea/project-files/docker/0f05da8c-b921-41ec-9777-a6bac6767278/gns3volumes/root$ ls -l
total 323972
drwxrwxr-x 2 gns3 gns3 4096 Jun 10 08:48 arm-64-test
-rw-rw-r-- 1 gns3 gns3 331735334 Sep 20 08:01 dqib-arm64-image.zip
gns3@gns3vm:/opt/gns3/projects/69eb1883-ee38-4048-8194-823d501e1fea/project-files/docker/0f05da8c-b921-41ec-9777-a6bac6767278/gns3volumes/root$ 

```

Gambar 4.4 Memindahkan *Folder DQIB* ke *Node Virtual GNS3*

Setelah memindahkan *folder* ke *node* virtual. *Node* tersebut dihidupkan untuk dilakukan ekstraksi isi *folder .zip* tersebut. Ekstraksi dilakukan dengan baris perintah “*unzip*”

```

MQTT-Publisher-44:~# ls -l
total 323972
drwxrwxr-x 2 1000 1000 4096 Jun 10 15:48 arm-64-test
-rw-rw-r-- 1 1000 1000 331735334 Sep 20 15:01 dqib-arm64-image.zip
root@MQTT-Publisher-44:~# unzip dqib-arm64-image.zip
Archive: dqib-arm64-image.zip
  creating: dqib_arm64-virt/
  inflating: dqib_arm64-virt/image.qcow2
  inflating: dqib_arm64-virt/initrd
  inflating: dqib_arm64-virt/kernel
  inflating: dqib_arm64-virt/readme.txt
  inflating: dqib_arm64-virt/ssh_user_ecdsa_key
  inflating: dqib_arm64-virt/ssh_user_ed25519_key
  inflating: dqib_arm64-virt/ssh_user_rsa_key
root@MQTT-Publisher-44:~# ls -l
total 323976
drwxrwxr-x 2 1000 1000 4096 Jun 10 15:48 arm-64-test
-rw-rw-r-- 1 1000 1000 331735334 Sep 20 15:01 dqib-arm64-image.zip
drwxr-xr-x 2 root root 4096 Jun 2 12:26 dqib_arm64-virt
root@MQTT-Publisher-44:~# cd dqib_arm64-virt/
root@MQTT-Publisher-44:~/dqib_arm64-virt# ls -l
total 795416
-rw-r--r-- 1 root root 747896832 Jun 2 12:26 image.qcow2
-rw-r--r-- 1 root root 29957606 Jun 2 12:07 initrd
-rw-r--r-- 1 root root 36630464 Jun 1 01:24 kernel
-rw-r--r-- 1 root root 903 Jun 2 12:07 readme.txt
-rw-r--r-- 1 root root 513 Jun 2 12:04 ssh_user_ecdsa_key
-rw-r--r-- 1 root root 411 Jun 2 12:04 ssh_user_ed25519_key
-rw-r--r-- 1 root root 2602 Jun 2 12:04 ssh_user_rsa_key
root@MQTT-Publisher-44:~/dqib_arm64-virt# 

```

Gambar 4.5 Ekstraksi isi *Folder DQIB* di *Node Virtual GNS3*

Selanjutnya, *node* virtual tersebut akan menjalankan program *QEMU* yang berjalan berdasarkan arsitektur prosessor *ARM64* yang juga berjalan di sistem *IoT*

seperti *Raspberry pi*, program *QEMU* dijalankan berdasarkan isi *file* yang ada di *folder* yang sudah diekstraksi tersebut.

```
root@MQTT-Publisher-44:~/dqib_arm64-virt# ls -l
total 795416
-rw-r--r-- 1 root root 747896832 Jun  2 12:26 image.qcow2
-rw-r--r-- 1 root root 29957606 Jun  2 12:07 initrd
-rw-r--r-- 1 root root 36630464 Jun  1 01:24 kernel
-rw-r--r-- 1 root root   903 Jun  2 12:07 readme.txt
-rw-r--r-- 1 root root    513 Jun  2 12:04 ssh_user_ecdsa_key
-rw-r--r-- 1 root root    411 Jun  2 12:04 ssh_user_ed25519_key
-rw-r--r-- 1 root root   2602 Jun  2 12:04 ssh_user_rsa_key
root@MQTT-Publisher-44:~/dqib_arm64-virt# qemu-system-aarch64 -machine 'virt' -cpu 'cortex-a57' -m 256 -device virtio-blk-device,drive=hd -drive file=image.qcow2,if=none,id=hd0 -device virtio-net-device,netdev=net -netdev user,id=net,hostfwd=tcp::2222-:22 -kernel kernel -initrd initrd -nographic -append "root=LABEL=rootfs console=ttyAMA0"
```

Gambar 4.6 Baris Perintah untuk Menjalankan Program *QEMU*

```
Debian GNU/Linux trixie/sid debian ttyAMA0

debian login: root
Password:
Linux debian 6.8.12-arm64 #1 SMP Debian 6.8.12-1 (2024-05-31) aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@debian:~# uname -a
Linux debian 6.8.12-arm64 #1 SMP Debian 6.8.12-1 (2024-05-31) aarch64 GNU/Linux
root@debian:~# uname -m
aarch64
root@debian:~#
```

Gambar 4.7 *QEMU* Berjalan dengan Arsitektur Prosessor ARM64/Aarch64

Selanjutnya, dimasukkan baris perintah *apt-get update* dan *apt-get upgrade* untuk melakukan pembaruan pada versi aplikasi sistem dan dapat melakukan instalasi *software* lain. Setelah pembaruan dilakukan, dimasukkan baris perintah *apt-get install python3* dan *apt-get install python3-pip* untuk melakukan pengecekan instalasi *software python* dan *python3-pip* di *node QEMU* tersebut. Jika *python* dan *python3-pip* belum terpasang maka akan dilakukan instalasi *python* dan *python3-pip*. Namun, jika sudah maka sistem operasi akan memberitahu bahwa program sudah terpasang di sistem tersebut.

Setelah itu, dimasukkan baris perintah *pip3 install "paho-mqtt<2.0.0"--break-system-packages* untuk melakukan pengecekan instalasi *paho MQTT* di *node QEMU* tersebut. Jika *paho MQTT* belum terpasang maka akan dilakukan instalasi *paho MQTT*. Namun, jika sudah maka sistem operasi akan memberitahu bahwa program sudah terpasang di sistem tersebut.

Setelah dilakukan instalasi program yang dibutuhkan. Selanjutnya, dibuat file .py di node publisher tersebut dengan baris perintah *touch paho_mqtt_publisher.py*. File berekstensi .py tersebut akan dimasukkan kode program python yang berfungsi untuk menjalankan *MQTT publisher*. Setelah baris kode program dimasukkan ke file .py, file tersebut dapat disimpan di sistem *MQTT publisher* agar dapat diaktifkan ketika pengujian dilakukan. Berikut adalah pseudocode dari program python untuk node *MQTT publisher*.

```
import paho.mqtt.client
import time
```

Initialize MQTT broker information

1. mqtt_broker = "mqtt.eclipseprojects.io"
2. port = 1883

Set up MQTT topic and time interval

1. topic_pub = 'IoT/sub/device/aarch64-1 to 45'
2. time_interval = 1 second

Define callback function on_connect:

When a connection is established:

1. Print the connection result code
2. Subscribe to the topic_pub

Create MQTT client instance

Create client

Assign callback function to on_connect:

```
client.on_connect = on_connect
```

Connect to MQTT broker

```
client.connect(mqtt_broker, port)
```

Publish a message to topic_pub

Publish message: 'A very long string, up to 1500 bytes total'

Start infinite loop:

1. Store current time in _time_0
2. Set cnt = 0
3. **Inside the loop:**
 - Call client.loop() to handle incoming messages
 - **If the client is connected:**
 - Check if time_interval has passed:
 - Construct message pesan with placeholder {} replaced by cnt
 - The message content is a long string
 - Publish pesan to topic_pub
 - Print the sent message
 - **Wait for 1 second**
 - Pause the loop for 1 second using sleep(1)
 - Increment cnt by 1
 - Update _time_0 to the current time

4.2.2.3 Implementasi Node MQTT Subscriber

Implementasi dan konfigurasi *node MQTT Subscriber* menggunakan *node Ubuntu Docker Guest* lalu dilakukan instalasi *software python3*, *python3-pip*, dan *paho MQTT* agar dapat berfungsi sebagai *node MQTT subscriber*. *Node MQTT subscriber* diperlukan untuk menerima data yang dikirimkan oleh *node MQTT publisher* dan telah melewati perantara *MQTT broker*.

Implementasi diawali dengan menghidupkan *node MQTT subscriber* yang telah dipasangkan ke *GNS3 dashboard* dan memasukkan baris perintah *apt-get update* dan *apt-get upgrade* untuk melakukan pembaruan pada versi aplikasi sistem dan dapat melakukan instalasi *software* lain. Setelah pembaruan dilakukan, dilakukan instalasi *software python*, *python3-pip*, dan *paho-mqtt* di *node subscriber* tersebut.

Selanjutnya, dimasukkan baris perintah *apt-get install python3* dan *apt-get install python3-pip* untuk melakukan pengecekan instalasi *software python* dan *python3-pip* di *node QEMU* tersebut. Jika *python* dan *python3-pip* belum terpasang maka akan dilakukan instalasi *python* dan *python3-pip*. Namun, jika sudah maka sistem operasi akan memberitahu bahwa program sudah terpasang di sistem tersebut.

Setelah itu, dimasukkan baris perintah *pip3 install "paho-mqtt<2.0.0"--break-system-packages* untuk melakukan pengecekan instalasi *paho MQTT* di *node subscriber* tersebut. Jika *paho MQTT* belum terpasang maka akan dilakukan instalasi *paho MQTT*. Namun, jika sudah maka sistem operasi akan memberitahu bahwa *software* tersebut telah terpasang di *node* tersebut.

Setelah dilakukan instalasi program yang dibutuhkan. Selanjutnya, dibuat *file .py* di *node subscriber* tersebut dengan baris perintah *touch paho_mqtt_subscriber.py*. *File* berekstensi *.py* tersebut akan dimasukkan kode program *python* yang berfungsi untuk menjalankan program *MQTT subscriber*. Setelah baris kode program dimasukkan ke *file .py*, *file* tersebut dapat disimpan di sistem *MQTT subscriber* agar dapat diaktifkan ketika pengujian dilakukan. Berikut adalah *pseudocode* dari program *python* untuk *node MQTT subscriber*.

```
import paho.mqtt.client
import time

Initialize MQTT broker details
1. mqtt_broker = "mqtt.eclipseprojects.io"
2. port = 1883

Set MQTT topic and time interval
1. topic_sub = 'IoT/sub/device/#'
2. time_interval = 1 second
```

Define callback function on_connect:

When a connection is established:

1. Print connection result code
2. Subscribe to topic_sub

Define callback function on_message:

When a message is received:

1. Print the topic and payload of the received message

Create MQTT client instance

1. Create client

Assign callback functions to on_connect and on_message:

1. Set client.on_connect = on_connect
2. Set client.on_message = on_message

Connect to the MQTT broker

1. client.connect(mqtt_broker, port)

Start the loop to handle MQTT events continuously:

1. Use client.loop_forever() to keep listening and processing messages indefinitely.

4.2.3 Implementasi Program di Mikrokontroller ESP32

Tahap implemenetasi dan penggeraan selanjutnya dilakukan pada bagian Mikrokontroler *ESP32* dengan memanfaatkan *software Arduino IDE* sebagai *software Integrated development environment (IDE)* untuk mikrokontroller berbasis *Arduino* dan *ESP32*. Bagian implementasi ini diawali dengan melakukan instalasi *software Arduino IDE* agar dapat melakukan pemrograman untuk perangkat *IoT* berarsitektur *esp32*.

Setelah melalukan instalasi *Arduino IDE*, dilakukan proses pembagian data yang akan dikirim oleh mikrokontroller *ESP32*. Dengan 5 buah mikrokontroller *ESP32* yang digunakan, kelimanya akan dibagi menjadi 2 tipe data yang akan dikirim *ESP32* yaitu

ESP32 yang akan mengirim data berupa hasil deteksi sensor jarak *HC-SR04* yang ditandai dengan kode *ESP32-1* dan *ESP32-2*, dan *ESP32* yang akan mengirim data berbentuk *random string* yang ditandai dengan kode *ESP32-3*, *ESP32-4*, dan *ESP32-5*. Kedua tipe ini memiliki proses penggerjaan yang berbeda antara satu dengan yang lain.

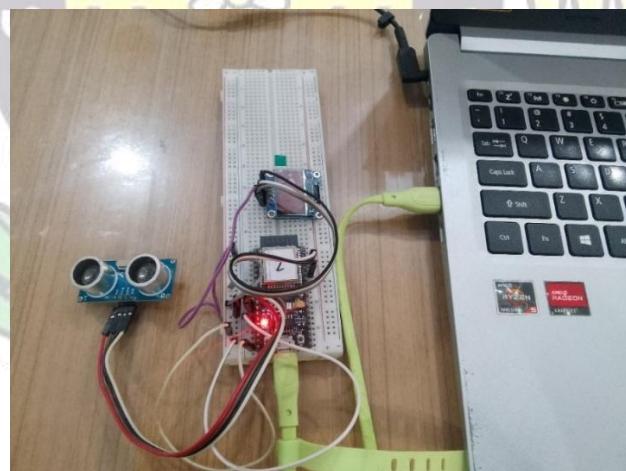
4.2.3.1 Implementasi *ESP32* dengan Sensor Jarak *HC-SR04*

Implementasi dan konfigurasi *ESP32* dengan sensor jarak *HC-SR04* diawali dengan melakukan penyusunan mikrokontroller dan sensor yang diperlukan di *breadboard*.



Gambar 4.8 Hasil Penyusunan Mikrokontroller dan Sensor di *Breadboard*

Setelah *ESP32* dan sensor telah disusun dengan benar, rangkaian tersebut dihubungkan ke komputer *host* untuk dilakukan konfigurasi kode program.



Gambar 4.9 Menghubungkan rangkaian sensor ke komputer *host*

Setelah rangkaian dihubungkan ke komputer *host*, dilakukan *upload* kode program untuk melakukan fungsi *node MQTT* sebagai *node publisher*. Setelah *upload* baris kode program ke mikrokontroller selesai, *file* program akan

tersimpan di mikrokontroller *ESP32* dan akan dijalankan secara otomatis apabila rangkaian sensor tersebut mendapatkan arus listrik dan akses internet yang diperlukan untuk terhubung dengan *MQTT broker*. Berikut adalah *pseudocode* untuk *node publisher* *ESP32* dengan sensor.

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
```

Define constants:

1. Set SCREEN_WIDTH = 128 (as OLED display width)
2. Set SCREEN_HEIGHT = 64 (as OLED display height)
3. Set SOUND_SPEED = 0.034 (as speed of sound in cm/ μ s)

Declare global variables:

Create display object using screen dimensions

Set trigPin = 14, echoPin = 27 for the ultrasonic sensor (HCSR-04)

Set WiFi credentials:

1. ssid = "wifi-SSID"
2. password = "wifi-password"

Define MQTT Broker information:

1. mqtt_broker = "mqtt.eclipseprojects.io"
2. topic_pub = 'IoT/sub/device/esp-2 or 1'
3. mqtt_port = 1883

Initialize sensor data variables:

1. duration
2. distanceCm

Define unique MQTT client_id = "mqtt_IoT_ESP32_2 or 1"

Define waktu_sebelum, waktu_interval = 1*1000

Declare data_pot_sebelum = ""

Declare functions:

1. connectWifi() - connects to Wi-Fi
2. connectMQTTServer() - connects to the MQTT server

Create WiFi and MQTT client objects

Setup Function:

1. Start serial communication
2. Configure trigPin as OUTPUT and echoPin as INPUT
3. Connect to Wi-Fi using connectWifi()
4. Connect to MQTT using connectMQTTServer()
5. Initialize the OLED display
6. Display setup message and publish "Hello world!" to topic_pub
7. Store the current time in waktu_sebelum

Main loop:

1. Continuously run client.loop() to handle MQTT messages
2. Clear trigPin (set to LOW), wait 20µs
3. Set trigPin to HIGH for 100µs, then set it to LOW
4. Measure sound wave travel time using pulseIn on echoPin
5. Calculate distance using the formula $\text{distanceCm} = \text{duration} * \text{SOUND_SPEED} / 2$
6. Print the distance to the serial monitor
7. Delay for 1 second
8. Update OLED display to show distance
9. Create a string data_pot based on distanceCm
10. **Check if the time interval has passed:**
 1. If yes, publish data_pot to topic_pub
 2. Print success or failure message to the serial monitor
 3. Update waktu_sebelum to the current time
 4. Save the current data_pot

Function connectWifi():

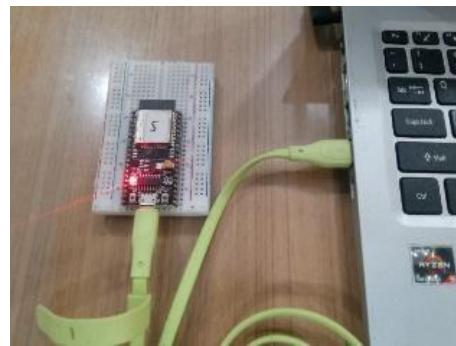
1. Start Wi-Fi connection using ssid and password
2. Keep trying to connect until successful
3. Print connection details (SSID, RSSI, MAC address, IP)

Function connectMQTTServer():

1. Start connecting to the MQTT broker
2. Retry connection if not successful and print the connection state

4.2.3.2 Implementasi ESP32 tanpa Sensor

Implementasi dan konfigurasi *ESP32* tanpa sensor diawali dengan menghubungkan *ESP32* ke komputer *host* untuk dilakukan konfigurasi kode program



Gambar 4.10 Menghubungkan *ESP32* ke komputer *host*

Setelah mikrokontroller dihubungkan ke komputer *host*, dilakukan *upload* kode program untuk melakukan fungsi *node MQTT* sebagai *node publisher*. Kode yang digunakan memiliki perbedaan dengan rangkaian mikrokontroller dengan sensor. Setelah *upload* baris kode program ke mikrokontroller selesai, file program akan tersimpan di mikrokontroller *ESP32* dan akan dijalankan secara otomatis apabila mikrokontroller *ESP32* tersebut mendapatkan arus listrik dan akses internet yang diperlukan untuk terhubung dengan *MQTT broker*. Berikut adalah *pseudocode* untuk *node MQTT publisher* *ESP32*.

```
#include <WiFi.h>
#include <PubSubClient.h>

Define constants:
1. Set ssid = "wifi-SSID" (Wi-Fi name)
2. Set password = "wifi-password" (Wi-Fi password)
3. Set MQTT broker details:
```

- ```

1. mqtt_broker = "mqtt.eclipseprojects.io"
2. topic_pub = "IoT/sub/device/esp-5 or 3/4"
3. mqtt_port = 1883
4. Define unique MQTT client_id = "mqtt_IoT_ESP32_5 or 3/4"

```

**Declare global variables:**

1. Initialize timing variables waktu\_sebelum, waktu\_interval = 1000ms
2. Initialize data\_pot\_sebelum = ""

**Declare functions:**

1. connectWifi() - connect to the Wi-Fi network
2. connectMQTTServer() - connect to the MQTT server

**Create Wi-Fi and MQTT client objects**

**Setup function:**

1. Start serial communication with a baud rate of 115200
2. Call connectWifi() to establish Wi-Fi connection
3. Call connectMQTTServer() to connect to the MQTT broker
4. Publish "Hello world!" to the topic\_pub
5. Store the current time in waktu\_sebelum

**Main loop:**

1. Continuously run client.loop() to handle MQTT operations
2. Define a string analog\_pot with a message of up to 25 words
3. Convert analog\_pot to data\_pot
4. Delay for 1 second
5. **Check if the time interval has passed:**

1. If yes, publish data\_pot to the MQTT topic (topic\_pub)
2. Print the status of the publish operation (success or failure)
3. Update waktu\_sebelum to the current time
4. Store data\_pot as data\_pot\_sebelum

**Function connectWifi():**

1. Attempt to connect to Wi-Fi using the ssid and password
2. Retry connection if unsuccessful
3. Print Wi-Fi connection details once connected (SSID, RSSI, MAC address, IP)

**Function connectMQTTServer():**

1. Attempt to connect to the MQTT broker using mqtt\_broker and mqtt\_port
2. Retry connection if unsuccessful and print the connection state

### 4.3 Implementasi Pengambilan Data

Implementasi pengambilan data pada penelitian ini dilakukan setelah semua konfigurasi telah selesai dilakukan. Proses pengambilan data terbagi menjadi 4 tahap, yaitu pengaktifan protokol *routing* pilihan, pengaktifan *software wireshark* di jalur data yang sudah ditentukan, pengaktifan *node publisher* dan *subscriber*, dan penghitungan waktu. Berikut tahapan prosesnya.

#### 4.3.1 Pengaktifan Protokol Routing Pilihan

Tahap pengaktifan protokol *routing* pilihan dilakukan untuk menentukan protokol *routing* yang akan dipakai untuk pengujian. Pemilihan protokol *routing* dapat dilakukan melalui *software winbox* yang terhubung dengan *router mikrotik* di antarmuka *GNS3*. Setiap sesi pengujian hanya memungkinkan satu protokol *routing* yang hidup karena satu protokol *routing* dapat menimpa protokol *routing* lainnya.

| Route List |                |            |          |               |              |
|------------|----------------|------------|----------|---------------|--------------|
|            | Dst. Address   | Gateway    | Distance | Routing Table | Pref. Source |
| DAC        | ▶ 10.10.0.0/24 | ether1     | 0        | main          |              |
| DAC        | ▶ 10.10.8.0/24 | ether8     | 0        | main          |              |
| DAC        | ▶ 10.10.7.0/24 | ether7     | 0        | main          |              |
| DAC        | ▶ 10.10.6.0/24 | ether6     | 0        | main          |              |
| DAC        | ▶ 10.10.5.0/24 | ether5     | 0        | main          |              |
| DAC        | ▶ 10.10.4.0/24 | ether4     | 0        | main          |              |
| DAC        | ▶ 10.10.3.0/24 | ether3     | 0        | main          |              |
| DAC        | ▶ 10.10.2.0/24 | ether2     | 0        | main          |              |
| AS         | ▶ 0.0.0.0/0    | 10.10.10.1 | 1        | main          |              |

9 items out of 30

Gambar 4.11 Protokol *Routing Statis Aktif*

| Route List |                    |                   |          |               |              |
|------------|--------------------|-------------------|----------|---------------|--------------|
|            | Dst. Address       | Gateway           | Distance | Routing Table | Pref. Source |
| DAr        | ▶ 192.168.122.0/24 | 10.10.10.1%ether1 | 120      | main          |              |
| DAr        | ▶ 192.168.8.0/24   | 10.10.8.2%ether8  | 120      | main          |              |
| DAr        | ▶ 192.168.7.0/24   | 10.10.7.2%ether7  | 120      | main          |              |
| DAr        | ▶ 192.168.6.0/24   | 10.10.6.2%ether6  | 120      | main          |              |
| DAr        | ▶ 192.168.5.0/24   | 10.10.5.2%ether5  | 120      | main          |              |
| DAr        | ▶ 192.168.4.0/24   | 10.10.4.2%ether4  | 120      | main          |              |
| DAr        | ▶ 192.168.3.0/24   | 10.10.3.2%ether3  | 120      | main          |              |
| DAr        | ▶ 192.168.2.0/24   | 10.10.2.2%ether2  | 120      | main          |              |
| DAC        | ▶ 10.10.10.0/24    | ether1            | 0        | main          |              |
| DAC        | ▶ 10.10.8.0/24     | ether8            | 0        | main          |              |
| DAC        | ▶ 10.10.7.0/24     | ether7            | 0        | main          |              |
| DAC        | ▶ 10.10.6.0/24     | ether6            | 0        | main          |              |
| DAC        | ▶ 10.10.5.0/24     | ether5            | 0        | main          |              |
| DAC        | ▶ 10.10.4.0/24     | ether4            | 0        | main          |              |
| DAC        | ▶ 10.10.3.0/24     | ether3            | 0        | main          |              |
| DAC        | ▶ 10.10.2.0/24     | ether2            | 0        | main          |              |
| DAr        | ▶ 0.0.0.0/0        | 10.10.10.1%ether1 | 120      | main          |              |
| XSI        | ▶ 0.0.0.0/0        | 10.10.10.1        | 1        | main          |              |

18 items out of 39

Gambar 4.12 Protokol *Routing Dinamis RIP Aktif*

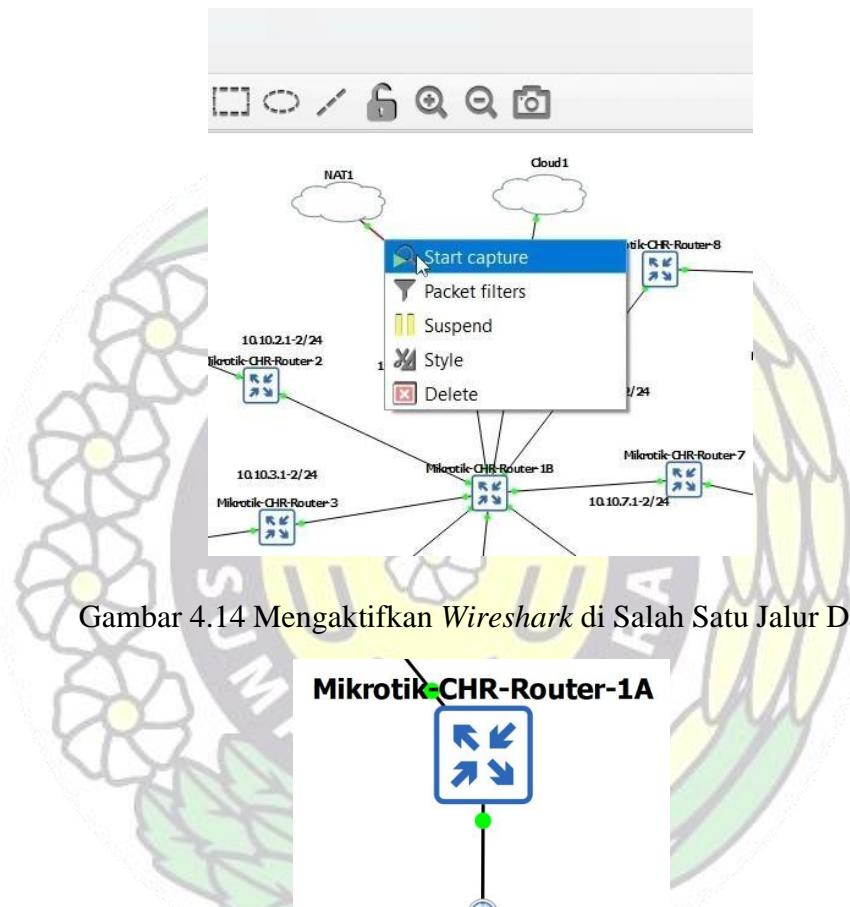
| Route List |                    |                   |          |               |              |
|------------|--------------------|-------------------|----------|---------------|--------------|
|            | Dst. Address       | Gateway           | Distance | Routing Table | Pref. Source |
| DAo        | ▶ 192.168.122.0/24 | 10.10.10.1%ether1 | 110      | main          |              |
| DAo        | ▶ 192.168.8.0/24   | 10.10.8.2%ether8  | 110      | main          |              |
| DAo        | ▶ 192.168.7.0/24   | 10.10.7.2%ether7  | 110      | main          |              |
| DAo        | ▶ 192.168.6.0/24   | 10.10.6.2%ether6  | 110      | main          |              |
| DAo        | ▶ 192.168.4.0/24   | 10.10.4.2%ether4  | 110      | main          |              |
| DAo        | ▶ 192.168.3.0/24   | 10.10.3.2%ether3  | 110      | main          |              |
| DAo        | ▶ 192.168.2.0/24   | 10.10.2.2%ether2  | 110      | main          |              |
| DAC        | ▶ 10.10.10.0/24    | ether1            | 0        | main          |              |
| DAC        | ▶ 10.10.8.0/24     | ether8            | 0        | main          |              |
| DAC        | ▶ 10.10.7.0/24     | ether7            | 0        | main          |              |
| DAC        | ▶ 10.10.6.0/24     | ether6            | 0        | main          |              |
| DAC        | ▶ 10.10.5.0/24     | ether5            | 0        | main          |              |
| DAC        | ▶ 10.10.4.0/24     | ether4            | 0        | main          |              |
| DAC        | ▶ 10.10.3.0/24     | ether3            | 0        | main          |              |
| DAC        | ▶ 10.10.2.0/24     | ether2            | 0        | main          |              |
| DAo        | ▶ 0.0.0.0/0        | 10.10.10.1%ether1 | 110      | main          |              |
| XSI        | ▶ 0.0.0.0/0        | 10.10.10.1        | 1        | main          |              |

17 items out of 38

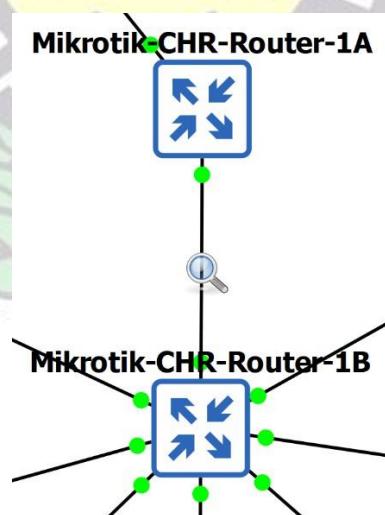
Gambar 4.13 Protokol *Routing Dinamis OSPF Aktif*

#### 4.3.2 Pengaktifan Software Wireshark

Tahap pengaktifan *software wireshark* di jalur data yang sudah ditentukan dilakukan agar data yang melewati jalur jaringan dapat dideteksi dan dapat dianalisis. Pengaktifan *software wireshark* dilakukan di beberapa jalur jaringan. Yaitu, di jalur yang banyak dilalui data dan jalur jaringan lain yang dapat dipilih secara acak. Pengaktifan *software wireshark* dilakukan sebelum *node publisher* dan *subscriber* diaktifkan kode programnya agar hasil deteksi menjadi optimal dan menampilkan data yang selengkap mungkin.



Gambar 4.14 Mengaktifkan Wireshark di Salah Satu Jalur Data



Gambar 4.15 Logo Kaca Pembesar Menandai Aktifnya Wireshark di Jalur Data Terpilih

### 4.3.3 Pengaktifan Node Publisher dan Subscriber

Tahap pengaktifan *node publisher* dan *subscriber* dilakukan agar terdapat data yang melewati jalur jaringan dan proses tersebut dapat dideteksi oleh *wireshark* sehingga dapat dilakukan dianalisis. Pengaktifan *node publisher* dan *subscriber* secara berurut dilakukan dari pengaktifan *node subscriber*, lalu *node publisher* di *GNS3*, dan terakhir *node publisher* dari *ESP32*.



```
MQTT-SI x MQTT-Sub x MQTT-Sub x MQTT-Sub x MQTT-Sub x MQTT-Sub x MQTT-Sub | +
```

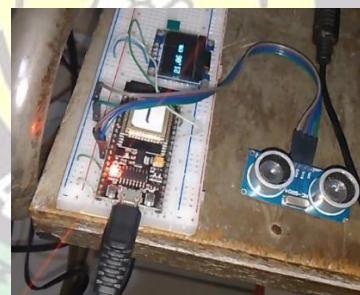
```
root@MQTT-Subscriber-2:~# python3 paho_mqtt_subscriber.py
Connected with result code 0
```

Gambar 4.16 Mengaktifkan *Node MQTT Subscriber* di *GNS3*



```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@debian:~# python3 paho-mqtt-publisher.py
Connected with result code 0
```

Gambar 4.17 Mengaktifkan *Node MQTT Publisher* di *GNS3*



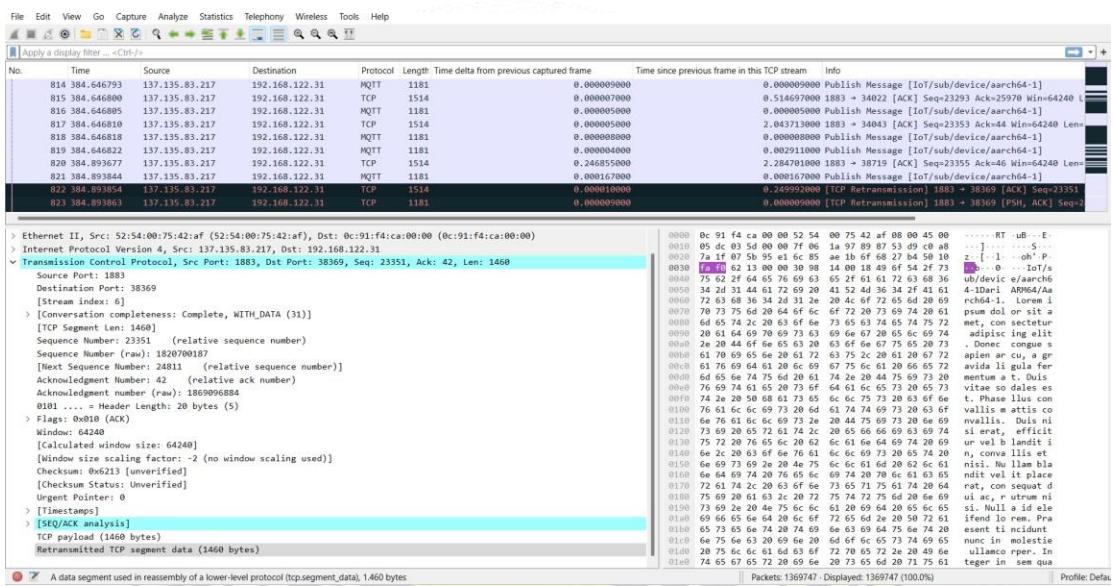
Gambar 4.18 Mengaktifkan *Node MQTT Publisher* *ESP32* Dengan Sensor



Gambar 4.19 Mengaktifkan *Node MQTT Publisher* *ESP32* Tanpa Sensor

#### 4.3.4 Penghitungan Waktu

Tahap penghitungan waktu dilakukan setelah semua *node publisher* dan *subscriber* sudah dinyalakan dan *wireshark* sudah dapat melakukan deteksi data. Penghitungan waktu yang diberikan untuk penelitian ini adalah 10 menit untuk setiap pengujian protokol *routing*. Setelah 10 menit telah terlewati, dilakukan pengecekan hasil deteksi dari *wireshark* untuk melihat apakah hasil deteksi memenuhi syarat untuk dilakukan analisis. Apabila syaratnya telah terpenuhi maka yang dilakukan selanjutnya adalah mematikan semua *node publisher*, lalu mematikan semua *node subscriber*, kemudian menghentikan proses deteksi data, dan menyimpan file hasil deteksi jalur data yang selanjutnya akan dianalisis.



Gambar 4.20 Hasil Deteksi Software Wireshark di Salah Satu Jalur Data

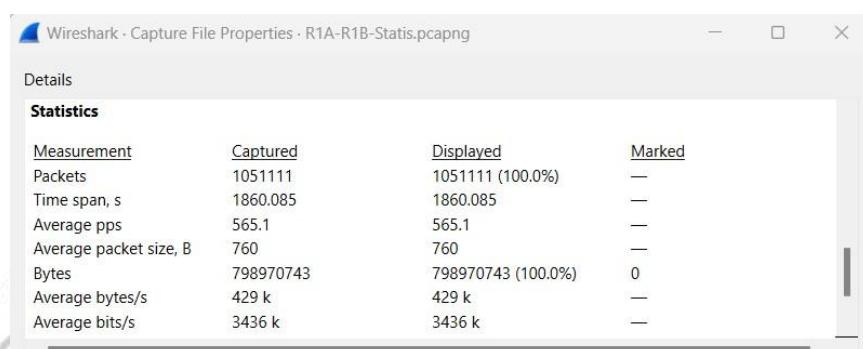
#### 4.4 Analisis QoS Protokol Routing Berdasarkan Hasil Deteksi Wireshark

Analisis *Quality of Service (QoS)* protokol *routing* dilakukan setelah semua protokol *routing* yang diujikan telah dilakukan pengambilan datanya menggunakan *software Wireshark*. Terdapat 5 parameter *QoS* yang akan dianalisis yaitu *throughput*, *delay*, *packet loss*, *latency*, dan *jitter*. Analisis akan dilakukan per setiap parameter *QoS* terhadap 3 protokol *routing* yang diujikan dan dipilih diantara beberapa jalur data yang digunakan pada pengujian serta dilalui konfigurasi dari protokol *routing* yang diujikan. Yaitu antara jalur jaringan *Router 1A* ke *Router*

1B yang merupakan jalur jaringan utama dan jalur jaringan *Router 1B* ke *Router 4* yang merupakan jalur jaringan tepi.

#### 4.4.1 Analisis QoS Berdasarkan Throughput

Perhitungan *throughput* dapat dilakukan dengan 2 cara. Yaitu menggunakan rumus perhitungan *throughput* atau menggunakan fitur statistik yang disediakan oleh *software Wireshark* yaitu “*Capture file properties*”. Perbandingan akan dilakukan pada kedua jalur yang dipasangkan *Wireshark* dan ketiga protokol *routing* yang diujikan dengan *file* hasil uji untuk kedua jalur dan ketiga protokol *routing* tersebut dapat langsung diaplikasikan proses perhitungan tersebut.



Gambar 4.21 Hasil Penggunaan Fitur *Wireshark* Untuk Perhitungan *Throughput*

##### 4.4.1.1 Jalur Router 1A ke Router 1B

Tabel 4.1 menunjukkan statistik angka *bytes* dan lama waktu uji dari ketiga protokol *routing* yang diujikan pada jalur jaringan *Router 1A* ke *Router 1B*.

Tabel 4.1 Statistik Hasil Pengujian *Throughput* jalur *R1A-R1B*

| No. | Protokol Routing | Angka bytes         | Lama waktu uji |
|-----|------------------|---------------------|----------------|
| 1.  | Statis           | 798.970.743 bytes   | 1860,085 detik |
| 2.  | Dinamis RIP      | 1.056.076.992 bytes | 2821.557 detik |
| 3.  | Dinamis OSPF     | 728.804.289 bytes   | 1934.223 detik |

Kemudian dilakukan perhitungan *throughput* menggunakan rumus hitung *throughput* pada ketiga protokol *routing*.

$$\text{Throughput Statis} = 798.970.743 / 1860,085$$

$$\begin{aligned}
 &= 429.534 \text{ bytes/detik} \\
 &= 429.534 / 1000 = 429,534 \text{ KB/detik} \\
 &= 429,534 * 8 = 3.436 \text{ Kb/detik}
 \end{aligned}$$

$$\text{Throughput Dinamis } RIP = 1.056.076.992 / 2821.557$$

$$\begin{aligned}
 &= 374.288 \text{ bytes/detik} \\
 &= 374.288 / 1000 = 374,288 \text{ KB/detik} \\
 &= 374,288 * 8 = 2.994 \text{ Kb/detik}
 \end{aligned}$$

$$\text{Throughput Dinamis } OSPF = 728.804.289 / 1934.223$$

$$\begin{aligned}
 &= 376.794 \text{ bytes/detik} \\
 &= 376.794 / 1000 = 376,794 \text{ KB/detik} \\
 &= 376,794 * 8 = 3.014 \text{ Kb/detik}
 \end{aligned}$$

#### 4.4.1.2 Jalur Router 1B ke Router 4

Tabel 4.2 menunjukkan statistik angka *bytes* dan lama waktu uji dari ketiga protokol *routing* yang diujikan pada jalur jaringan *Router 1B* ke *Router 4*.

Tabel 4.2 Statistik Hasil Pengujian *Throughput* Jalur R1B-R4

| No. | Protokol <i>Routing</i> | Angka <i>bytes</i>       | Lama waktu uji |
|-----|-------------------------|--------------------------|----------------|
| 1.  | Statis                  | 114.432.453 <i>bytes</i> | 1860,084 detik |
| 2.  | Dinamis <i>RIP</i>      | 150.651.194 <i>bytes</i> | 2793,811 detik |
| 3.  | Dinamis <i>OSPF</i>     | 104.895.096 <i>bytes</i> | 1927,659 detik |

Kemudian dilakukan perhitungan *throughput* menggunakan rumus hitung *throughput* pada ketiga protokol *routing*.

$$\text{Throughput Statis} = 114.432.453 / 1860,084$$

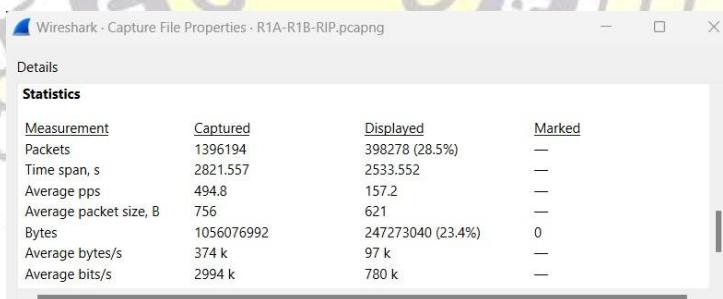
$$\begin{aligned}
 &= 61.520 \text{ bytes/detik} \\
 &= 61.520 / 1000 = 61,52 \text{ KB/detik} \\
 &= 61,52 * 8 = 492 \text{ Kb/detik}
 \end{aligned}$$

$$\begin{aligned}
 \text{Throughput Dinamis } RIP &= 150.651.194 / 2793,811 \\
 &= 53.923 \text{ bytes/detik} \\
 &= 53.923 / 1000 = 53,923 \text{ KB/detik} \\
 &= 53,923 * 8 = 431 \text{ Kb/detik}
 \end{aligned}$$

$$\begin{aligned}
 \text{Throughput Dinamis } OSPF &= 104.895.096 / 1927,659 \\
 &= 54.415 \text{ bytes/detik} \\
 &= 54.415 / 1000 = 54,415 \text{ KB/detik} \\
 &= 54,415 * 8 = 435 \text{ Kb/detik}
 \end{aligned}$$

#### 4.4.2 Analisis QoS Berdasarkan Packet Loss

Perhitungan *packet loss* dapat dilakukan dengan 2 cara. Yaitu menggunakan rumus perhitungan *packet loss* atau menggunakan fitur *packet search* dengan memasukkan baris perintah `tcp.analysis.retransmission // tcp.analysis.lost_segment // tcp.analysis.duplicate_ack // tcp.analysis.out_of_order` untuk melakukan penyaringan hasil deteksi yang menunjukkan *packet loss* dan statistik yang disediakan oleh *software Wireshark* yaitu “*Capture file properties*”. Perbandingan akan dilakukan pada kedua jalur yang dipasangkan *Wireshark* dan ketiga protokol *routing* yang diujikan dengan *file hasil uji* untuk kedua jalur dan ketiga protokol *routing* tersebut dapat langsung diaplikasikan proses perhitungan tersebut.



Gambar 4.22 Hasil Penggunaan Fitur *Wireshark* Untuk Perhitungan *Packet Loss*

##### 4.4.2.1 Jalur Router 1A ke Router 1B

Tabel 4.3 menunjukkan statistik angka paket ditandai *packet loss* dan total paket data dari ketiga protokol *routing* yang diujikan pada jalur jaringan *Router 1A* ke *Router 1B*.

Tabel 4.3 Statistik Hasil Pengujian *Packet Loss* Jalur *R1A-R1B*

| No. | Protokol <i>Routing</i> | Angka <i>Packet Loss</i> | Total Paket Data     |
|-----|-------------------------|--------------------------|----------------------|
| 1.  | Statis                  | 281.574 paket data       | 1.051.111 paket data |
| 2.  | Dinamis <i>RIP</i>      | 398.278 paket data       | 1.396.194 paket data |
| 3.  | Dinamis <i>OSPF</i>     | 184.815 paket data       | 957.273 paket data   |

Kemudian dilakukan perhitungan *packet loss* menggunakan rumus hitung *packet loss* pada ketiga protokol *routing*.

$$\begin{aligned} \text{Packet Loss Statis} &= 281.574 / 1.051.111 * 100\% \\ &= 26,78\% \approx 26,8\% \end{aligned}$$

$$\begin{aligned} \text{Packet Loss Dinamis } \textit{RIP} &= 398.278 / 1.396.194 * 100\% \\ &= 28,52\% \approx 28,5\% \end{aligned}$$

$$\begin{aligned} \text{Packet Loss Dinamis } \textit{OSPF} &= 184.815 / 957.273 * 100\% \\ &= 19,3\% \end{aligned}$$

#### 4.4.2.2 Jalur Router 1B ke Router 4

Tabel 4.4 menunjukkan statistik angka paket ditandai *packet loss* dan total paket data dari ketiga protokol *routing* yang diujikan pada jalur jaringan *Router 1B* ke *Router 4*.

Tabel 4.4 Statistik Hasil Pengujian *Packet Loss* Jalur *R1B-R4*

| No. | Protokol <i>Routing</i> | Angka <i>Packet Loss</i> | Total Paket Data   |
|-----|-------------------------|--------------------------|--------------------|
| 1.  | Statis                  | 49.115 paket data        | 163.423 paket data |
| 2.  | Dinamis <i>RIP</i>      | 66.103 paket data        | 214.461 paket data |
| 3.  | Dinamis <i>OSPF</i>     | 35.862 paket data        | 147.839 paket data |

Kemudian dilakukan perhitungan *packet loss* menggunakan rumus hitung *packet loss* pada ketiga protokol *routing*.

$$\begin{aligned} \text{Packet Loss Statis} &= 49.115 / 163.423 * 100\% \\ &= 30,05\% \approx 30,1\% \end{aligned}$$

$$\begin{aligned} \text{Packet Loss Dinamis RIP} &= 66.103 / 214.461 * 100\% \\ &= 30,82\% \approx 30,8\% \end{aligned}$$

$$\begin{aligned} \text{Packet Loss Dinamis OSPF} &= 35.862 / 147.839 * 100\% \\ &= 24,25\% \approx 24,3\% \end{aligned}$$

#### 4.4.3 Analisis QoS Berdasarkan Delay

Perhitungan *delay* diawali dengan menggunakan fitur *packet search* untuk mencari paket yang dikirim dari *node* fisik dan *node* virtual dengan memasukkan baris perintah `_ws.col.info == "Publish Message [IoT/sub/device/aarch64-13]"` untuk paket dari *node* virtual dan `_ws.col.info == "Publish Message [IoT/sub/device/esp-3]"` untuk paket dari *node* fisik. Dengan angka di *node* virtual dapat dipilih acak dari *aarch64-1* sampai *aarch64-45* lalu angka di *node* fisik dapat dipilih acak dari *esp-1* sampai *esp-5*. Namun, untuk di penelitian ini akan dipilih *node aarch64-13*, *esp-1*, dan *esp-3* untuk jalur jaringan *R1A-R1B* dan jalur jaringan *R1B-R4*.

Setelah penyaringan selesai dilakukan pelacakan sumber paket dan perhitungan perbandingan waktu antara paket yang dikirim dengan yang diterima untuk *node* virtual dan dilakukan perbandingan waktu untuk setiap paket yang datang untuk *node* fisik. Setelah itu, diaplikasikan rumus hitung *delay* untuk setiap paket yang terbaca. Perbandingan akan dilakukan pada kedua jalur yang dipasangkan *Wireshark* dan ketiga protokol *routing* yang diujikan dengan *file* hasil uji untuk kedua jalur dan ketiga protokol *routing* tersebut dapat langsung diaplikasikan proses perhitungan tersebut.



Gambar 4.23 Hasil Pelacakan Paket Untuk Perhitungan *Delay* pada *Node aarch-xx*

| ws.col.info == "Publish Message [IoT/sub/device/esp-5]" |                  |                         |             |          |                                        |
|---------------------------------------------------------|------------------|-------------------------|-------------|----------|----------------------------------------|
| No.                                                     | Time             | Source                  | Destination | Protocol | Info                                   |
| 283859 *REF*                                            | 295758 11.860210 | mqtt.eclipseprojects.io | 10.10.10.2  | MQTT     | Publish Message [IoT/sub/device/esp-5] |
| 326444 57.176358                                        | 327112 58.201977 | mqtt.eclipseprojects.io | 10.10.10.2  | MQTT     | Publish Message [IoT/sub/device/esp-5] |
|                                                         |                  | mqtt.eclipseprojects.io | 10.10.10.2  | MQTT     | Publish Message [IoT/sub/device/esp-5] |

Gambar 4.24 Hasil Pelacakan Paket Untuk Perhitungan *Delay* pada *Node esp-x*

#### 4.4.3.1 Jalur Router 1A ke Router 1B

Tabel 4.5 menunjukkan statistik angka waktu terima dan kirim data dari ketiga protokol *routing* yang diujikan pada jalur jaringan *Router 1A* ke *Router 1B*. Data yang diambil adalah data 3 paket data terawal yang dideteksi oleh *Wireshark*.

Tabel 4.5 Statistik Data Waktu Terima dan Kirim Paket Data Jalur *R1A-R1B*

| No. | Protokol<br><i>Routing</i> | Node     | Waktu Terima<br>(server) | Waktu Kirim (client) |
|-----|----------------------------|----------|--------------------------|----------------------|
| 1.  | Statis                     | aarch-13 | 0,535256 detik           | 0,000000 detik       |
|     |                            |          | 2,467160 detik           | 1,945541 detik       |
|     |                            |          | 4,866239 detik           | 4,345389 detik       |
|     | esp-1                      | esp-1    | 0,000287 detik           | 0,000000 detik       |
|     |                            |          | 17,566206 detik          | 0,000287 detik       |
|     |                            |          | 17,585645 detik          | 17,566206 detik      |
|     | esp-3                      | esp-3    | 0,194726 detik           | 0,000000 detik       |
|     |                            |          | 1,022758 detik           | 0,194726 detik       |

| No. | Protokol<br><i>Routing</i> | Node     | Waktu Terima<br>(server) | Waktu Kirim ( <i>client</i> ) |
|-----|----------------------------|----------|--------------------------|-------------------------------|
| 2.  | Dinamis<br><i>RIP</i>      | aarch-13 | 2,031079 detik           | 1,022758 detik                |
|     |                            |          | 0,512980 detik           | 0,000000 detik                |
|     |                            |          | 2,371530 detik           | 1,857574 detik                |
|     |                            | esp-1    | 4,410757 detik           | 3,900158 detik                |
|     |                            |          | 0,004083 detik           | 0,000000 detik                |
|     |                            |          | 0,004147 detik           | 0,004083 detik                |
|     |                            |          | 1,067032 detik           | 0,004147 detik                |
|     |                            | esp-3    | 1,095233 detik           | 0,000000 detik                |
|     |                            |          | 2,115313 detik           | 1,095233 detik                |
| 3.  | Dinamis<br><i>OSPF</i>     | aarch-13 | 2,115416 detik           | 2,115313 detik                |
|     |                            |          | 0,514920 detik           | 0,000000 detik                |
|     |                            |          | 2,576969 detik           | 2,060209 detik                |
|     |                            | esp-1    | 4,592178 detik           | 4,085732 detik                |
|     |                            |          | 1,010783 detik           | 0,000000 detik                |
|     |                            |          | 7,529407 detik           | 1,010783 detik                |
|     |                            |          | 29,495985 detik          | 7,529407 detik                |
|     |                            | esp-3    | 3,689232 detik           | 0,000000 detik                |
|     |                            |          | 24,450692 detik          | 3,689232 detik                |
|     |                            |          | 25,500608 detik          | 24,450692 detik               |

Kemudian dilakukan perhitungan *delay* menggunakan rumus hitung *delay* pada ketiga protokol *routing* pada *node* jaringan yang dipilih.

Protokol *routing* statis *node* aarch64-13.

- $Delay 1 = 0,535256 - 0,000000 = 0,535256$  detik
- $Delay 2 = 2,467160 - 1,945541 = 0,521619$  detik
- $Delay 3 = 4,866239 - 4,345389 = 0,520850$  detik
- Rerata =  $(0,535256 + 0,521619 + 0,520850) / 3 = 0,525908$  detik

Protokol *routing statis node esp-1.*

- $\text{Delay 1} = 0,000287 - 0,000000 = 0,000287 \text{ detik}$
- $\text{Delay 2} = 17,566206 - 0,000287 = 17,565919 \text{ detik}$
- $\text{Delay 3} = 17,585645 - 17,566206 = 0,019439 \text{ detik}$
- Rerata =  $(0,000287 + 17,565919 + 0,019439) / 3 = 5,861881 \text{ detik}$

Protokol *routing statis node esp-3.*

- $\text{Delay 1} = 0,194726 - 0,000000 = 0,194726 \text{ detik}$
- $\text{Delay 2} = 1,022758 - 0,194726 = 0,828032 \text{ detik}$
- $\text{Delay 3} = 2,031079 - 1,022758 = 1,008321 \text{ detik}$
- Rerata =  $(0,194726 + 0,828032 + 1,008321) / 3 = 0,677026 \text{ detik}$

Protokol *routing dinamis RIP node aarch64-13.*

- $\text{Delay 1} = 0,512980 - 0,000000 = 0,512980 \text{ detik}$
- $\text{Delay 2} = 2,371530 - 1,857574 = 0,513956 \text{ detik}$
- $\text{Delay 3} = 4,410757 - 3,900158 = 0,510599 \text{ detik}$
- Rerata =  $(0,512980 + 0,513956 + 0,510599) / 3 = 0,512511 \text{ detik}$

Protokol *routing dinamis RIP node esp-1.*

- $\text{Delay 1} = 0,004083 - 0,000000 = 0,004083 \text{ detik}$
- $\text{Delay 2} = 0,004147 - 0,004083 = 0,000064 \text{ detik}$
- $\text{Delay 3} = 1,067032 - 0,004147 = 1,062885 \text{ detik}$
- Rerata =  $(0,004083 + 0,000064 + 1,062885) / 3 = 0,355677 \text{ detik}$

Protokol *routing dinamis RIP node esp-3.*

- $\text{Delay 1} = 1,095233 - 0,000000 = 1,095233 \text{ detik}$
- $\text{Delay 2} = 2,115313 - 1,095233 = 1,020080 \text{ detik}$
- $\text{Delay 3} = 2,115416 - 2,115313 = 0,000103 \text{ detik}$
- Rerata =  $(1,095233 + 1,020080 + 0,000103) / 3 = 0,705138 \text{ detik}$

Protokol *routing* dinamis *OSPF node aarch64-13*.

- $\text{Delay 1} = 0,514920 - 0,000000 = 0,514920$  detik
- $\text{Delay 2} = 2,576969 - 2,060209 = 0,516760$  detik
- $\text{Delay 3} = 4,592178 - 4,085732 = 0,506446$  detik
- Rerata =  $(0,514920 + 0,516760 + 0,506446) / 3 = 0,512708$  detik

Protokol *routing* dinamis *OSPF node esp-1*.

- $\text{Delay 1} = 1,010783 - 0,000000 = 1,010783$  detik
- $\text{Delay 2} = 7,529407 - 1,010783 = 6,518624$  detik
- $\text{Delay 3} = 29,495985 - 7,529407 = 21,966578$  detik
- Rerata =  $(1,010783 + 6,518624 + 21,966578) / 3 = 9,831995$  detik

Protokol *routing* dinamis *OSPF node esp-3*.

- $\text{Delay 1} = 3,689232 - 0,000000 = 3,689232$  detik
- $\text{Delay 2} = 24,450692 - 3,689232 = 20,76146$  detik
- $\text{Delay 3} = 25,500608 - 24,450692 = 1,049916$  detik
- Rerata =  $(3,689232 + 20,76146 + 1,049916) / 3 = 8,500202$  detik

#### 4.4.3.2 Jalur Router 1B ke Router 4

Tabel 4.6 menunjukkan statistik angka waktu terima dan kirim data dari ketiga protokol *routing* yang diujikan pada jalur jaringan *Router 1B* ke *Router 4*. Data yang diambil adalah data 3 paket data terawal yang dideteksi oleh *Wireshark*.

Tabel 4.6 Statistik Data Waktu Terima dan Kirim Paket Data Jalur *R1B-R4*

| No. | Protokol<br><i>Routing</i> | Node            | Waktu Terima<br>(server) | Waktu Kirim (client) |
|-----|----------------------------|-----------------|--------------------------|----------------------|
| 1.  | Statis                     | <i>aarch-13</i> | 0,539867 detik           | 0,000000 detik       |
|     |                            |                 | 2,477375 detik           | 1,946267 detik       |
|     |                            |                 | 4,870227 detik           | 4,345966 detik       |
|     | <i>esp-1</i>               |                 | 2,260627 detik           | 0,000000 detik       |
|     |                            |                 | 23,553075 detik          | 2,260627 detik       |

| No. | Protokol<br><i>Routing</i> | Node            | Waktu Terima<br>(server) | Waktu Kirim ( <i>client</i> ) |
|-----|----------------------------|-----------------|--------------------------|-------------------------------|
| 2.  | Dinamis<br><i>RIP</i>      | <i>esp-3</i>    | 27,789520 detik          | 23,553075 detik               |
|     |                            |                 | 2,031708 detik           | 0,000000 detik                |
|     |                            |                 | 3,440892 detik           | 2,031708 detik                |
|     |                            |                 | 4,201342 detik           | 3,440892 detik                |
|     |                            | <i>aarch-13</i> | 0,512980 detik           | 0,000000 detik                |
|     |                            |                 | 2,371530 detik           | 1,857574 detik                |
|     |                            |                 | 4,410757 detik           | 3,900158 detik                |
|     |                            | <i>esp-1</i>    | 1,144625 detik           | 0,000000 detik                |
|     |                            |                 | 36,450562 detik          | 1,144625 detik                |
|     |                            |                 | 38,509086 detik          | 36,450562 detik               |
| 3.  | Dinamis<br><i>OSPF</i>     | <i>esp-3</i>    | 8,096692 detik           | 0,000000 detik                |
|     |                            |                 | 13,858658 detik          | 8,096692 detik                |
|     |                            |                 | 26,913704 detik          | 13,858658 detik               |
|     |                            | <i>aarch-13</i> | 0,516869 detik           | 0,000000 detik                |
|     |                            |                 | 2,579446 detik           | 2,059213 detik                |
|     |                            |                 | 4,593771 detik           | 4,084404 detik                |
|     |                            | <i>esp-1</i>    | 4,049832 detik           | 0,000000 detik                |
|     |                            |                 | 11,175996 detik          | 4,049832 detik                |
|     |                            |                 | 12,168501 detik          | 11,175996 detik               |
|     |                            | <i>esp-3</i>    | 59,890572 detik          | 0,000000 detik                |
|     |                            |                 | 63,147854 detik          | 59,890572 detik               |
|     |                            |                 | 66,163035 detik          | 63,147854 detik               |

Kemudian dilakukan perhitungan *delay* menggunakan rumus hitung *delay* pada ketiga protokol *routing* pada *node* jaringan yang dipilih.

Protokol *routing* statis *node aarch64-13*.

- $Delay 1 = 0,539867 - 0,000000 = 0,539867$  detik
- $Delay 2 = 2,477375 - 1,946267 = 0,531108$  detik

- $\text{Delay 3} = 4,870227 - 4,345966 = 0,524261$  detik
- Rerata  $= (0,539867 + 0,531108 + 0,524261) / 3 = 0,531745$  detik

Protokol *routing statis node esp-1.*

- $\text{Delay 1} = 2,260627 - 0,000000 = 2,260627$  detik
- $\text{Delay 2} = 23,553075 - 2,260627 = 21,292448$  detik
- $\text{Delay 3} = 27,78952 - 23,553075 = 4,236445$  detik
- Rerata  $= (2,260627 + 21,292448 + 4,236445) / 3 = 9,263173$  detik

Protokol *routing statis node esp-3.*

- $\text{Delay 1} = 2,031708 - 0,000000 = 2,031708$  detik
- $\text{Delay 2} = 3,440892 - 2,031708 = 1,409184$  detik
- $\text{Delay 3} = 4,201342 - 3,440892 = 0,760450$  detik
- Rerata  $= (2,031708 + 1,409184 + 0,760450) / 3 = 1,400447$  detik

Protokol *routing dinamis RIP node aarch64-13.*

- $\text{Delay 1} = 0,512980 - 0,000000 = 0,512980$  detik
- $\text{Delay 2} = 2,371530 - 1,857574 = 0,513956$  detik
- $\text{Delay 3} = 4,410757 - 3,900158 = 0,510599$  detik
- Rerata  $= (0,512980 + 0,513956 + 0,510599) / 3 = 0,512511$  detik

Protokol *routing dinamis RIP node esp-1.*

- $\text{Delay 1} = 1,144625 - 0,000000 = 1,144625$  detik
- $\text{Delay 2} = 36,450562 - 1,144625 = 35,305937$  detik
- $\text{Delay 3} = 38,509086 - 36,450562 = 2,058524$  detik
- Rerata  $= (1,144625 + 35,305937 + 2,058524) / 3 = 12,836362$  detik

Protokol *routing dinamis RIP node esp-3.*

- $\text{Delay 1} = 8,096692 - 0,000000 = 8,096692$  detik
- $\text{Delay 2} = 13,858658 - 8,096692 = 5,761966$  detik

- $\text{Delay 3} = 26,913704 - 13,858658 = 13,055046$  detik
- Rerata =  $(8,096692 + 5,761966 + 13,055046) / 3 = 8,971234$  detik

Protokol *routing* dinamis *OSPF node aarch64-13*.

- $\text{Delay 1} = 0,516869 - 0,000000 = 0,516869$  detik
- $\text{Delay 2} = 2,579446 - 2,059213 = 0,520233$  detik
- $\text{Delay 3} = 4,593771 - 4,084404 = 0,509367$  detik
- Rerata =  $(0,516869 + 0,520233 + 0,509367) / 3 = 0,515522$  detik

Protokol *routing* dinamis *OSPF node esp-1*.

- $\text{Delay 1} = 59,890572 - 0,000000 = 59,890572$  detik
- $\text{Delay 2} = 63,147854 - 59,890572 = 3,257282$  detik
- $\text{Delay 3} = 66,163035 - 63,147854 = 3,015181$  detik
- Rerata =  $(59,890572 + 3,257282 + 3,015181) / 3 = 22,054345$  detik

Protokol *routing* dinamis *OSPF node esp-3*.

- $\text{Delay 1} = 4,049832 - 0,000000 = 4,049832$  detik
- $\text{Delay 2} = 11,175996 - 4,049832 = 7,126164$  detik
- $\text{Delay 3} = 12,168501 - 11,175996 = 0,992505$  detik
- Rerata =  $(4,049832 + 7,126164 + 0,992505) / 3 = 4,056167$  detik

#### 4.4.4 Analisis QoS Berdasarkan Latency

Perhitungan *Latency* dilakukan dengan memasukkan rumus hitung *latency* yang menggunakan angka rerata *delay*. Dengan perhitungan di *node* virtual dapat dipilih acak dari angka *aarch64-1* sampai *aarch64-45* lalu angka di *node* fisik dapat dipilih acak dari *esp-1* sampai *esp-5*. Namun, untuk di penelitian ini akan dipilih *node aarch64-13*, *esp-1*, dan *esp-3* untuk jalur jaringan *RIA-RIB* dan jalur jaringan *RIB-R4*.

Perbandingan akan dilakukan pada kedua jalur yang dipasangkan *Wireshark* dan ketiga protokol *routing* yang diujikan dengan *file* hasil uji untuk kedua jalur dan ketiga protokol *routing* tersebut dapat langsung diaplikasikan proses perhitungan tersebut.

#### 4.4.4.1 Jalur Router 1A ke Router 1B

Tabel 4.7 menunjukkan statistik angka rerata *delay* dari ketiga protokol *routing* yang diujikan pada jalur jaringan *Router 1A* ke *Router 1B*.

Tabel 4.7 Statistik Hasil Hitung Rerata *Delay* di Jalur *R1A-R1B*

| No. | Protokol <i>Routing</i> | Node     | Waktu Rerata <i>Delay</i> (detik) |
|-----|-------------------------|----------|-----------------------------------|
| 1.  | Statis                  | aarch-13 | 0,525908 detik                    |
|     |                         | esp-1    | 5,861881 detik                    |
|     |                         | esp-3    | 0,677026 detik                    |
| 2.  | Dinamis RIP             | aarch-13 | 0,512511 detik                    |
|     |                         | esp-1    | 0,355677 detik                    |
|     |                         | esp-3    | 0,705138 detik                    |
| 3.  | Dinamis OSPF            | aarch-13 | 0,512708 detik                    |
|     |                         | esp-1    | 9,831995 detik                    |
|     |                         | esp-3    | 8,500202 detik                    |

Kemudian dilakukan perhitungan *latency* menggunakan rumus hitung *latency* pada ketiga protokol *routing* pada *node* jaringan yang dipilih.

Protokol *routing* statis.

- Node aarch64-13 =  $0,525908 / 2 = 0,262954$  detik
- Node esp-1 =  $5,861881 / 2 = 2,930940$  detik
- Node esp-3 =  $0,677026 / 2 = 0,338513$  detik
- Rerata =  $(0,262954 + 2,930940 + 0,338513) / 3 = 1,177469$  detik

Protokol *routing* dinamis *RIP*.

- Node aarch64-13 =  $0,512511 / 2 = 0,256255$  detik
- Node esp-1 =  $0,355677 / 2 = 0,177838$  detik
- Node esp-3 =  $0,705138 / 2 = 0,352569$  detik

- Rerata  $= (0,256255 + 0,177838 + 0,352569) / 3 = 0,262220$  detik

Protokol *routing* dinamis *OSPF*.

- *Node aarch64-13*  $= 0,512708 / 2 = 0,256354$  detik
- *Node esp-1*  $= 9,831995 / 2 = 4,915997$  detik
- *Node esp-3*  $= 8,500202 / 2 = 4,250101$  detik
- Rerata  $= (0,256354 + 4,915997 + 4,250101) / 3 = 3,140817$  detik

#### 4.4.4.2 Jalur Router 1B ke Router 4

Tabel 4.8 menunjukkan statistik angka rerata *delay* dari ketiga protokol *routing* yang diujikan pada jalur jaringan *Router 1B* ke *Router 4*.

Tabel 4.8 Statistik Hasil Hitung Rerata *Delay* di Jalur *R1B-R4*

| No. | Protokol <i>Routing</i> | Node            | Waktu Rerata <i>Delay</i> (detik) |
|-----|-------------------------|-----------------|-----------------------------------|
| 1.  | Statis                  | <i>aarch-13</i> | 0,531745 detik                    |
|     |                         | <i>esp-1</i>    | 9,263173 detik                    |
|     |                         | <i>esp-3</i>    | 1,400447 detik                    |
| 2.  | Dinamis <i>RIP</i>      | <i>aarch-13</i> | 0,512511 detik                    |
|     |                         | <i>esp-1</i>    | 12,836362 detik                   |
|     |                         | <i>esp-3</i>    | 8,971234 detik                    |
| 3.  | Dinamis <i>OSPF</i>     | <i>aarch-13</i> | 0,515522 detik                    |
|     |                         | <i>esp-1</i>    | 22,054345 detik                   |
|     |                         | <i>esp-3</i>    | 4,056167 detik                    |

Kemudian dilakukan perhitungan *latency* menggunakan rumus hitung *latency* pada ketiga protokol *routing* pada *node* jaringan yang dipilih.

Protokol *routing* statis.

- *Node aarch64-13*  $= 0,531745 / 2 = 0,265873$  detik
- *Node esp-1*  $= 9,263173 / 2 = 4,631586$  detik

- $Node esp-3$  =  $1,400447 / 2 = 0,700224$  detik
- Rerata =  $(0,265873 + 4,631586 + 0,700224) / 3 = 1,865894$  detik

Protokol *routing* dinamis *RIP*.

- $Node aarch64-13$  =  $0,512511 / 2 = 0,256255$  detik
- $Node esp-1$  =  $12,836362 / 2 = 6,418181$  detik
- $Node esp-3$  =  $8,971234 / 2 = 4,485617$  detik
- Rerata =  $(0,256255 + 6,418181 + 4,485617) / 3 = 3,720017$  detik

Protokol *routing* dinamis *OSPF*.

- $Node aarch64-13$  =  $0,515522 / 2 = 0,257761$  detik
- $Node esp-1$  =  $22,054345 / 2 = 11,027125$  detik
- $Node esp-3$  =  $4,056167 / 2 = 2,028084$  detik
- Rerata =  $(0,257761 + 11,027125 + 2,028084) / 3 = 4,437672$  detik

#### 4.4.5 Analisis QoS Berdasarkan Jitter

Perhitungan *jitter* dilakukan dengan menggunakan rumus hitung *jitter* yang menggunakan angka dari perhitungan *delay*. Dengan perhitungan di *node* virtual dapat dipilih acak dari angka *aarch64-1* sampai *aarch64-45* lalu angka di *node* fisik dapat dipilih acak dari *esp-1* sampai *esp-5*. Namun, untuk di penelitian ini akan dipilih *node aarch64-13*, *esp-1*, dan *esp-3* untuk jalur jaringan *R1A-R1B* dan jalur jaringan *R1B-R4*.

Perbandingan akan dilakukan pada kedua jalur yang dipasangkan *Wireshark* dan ketiga protokol *routing* yang diujikan dengan *file* hasil uji untuk kedua jalur dan ketiga protokol *routing* tersebut dapat langsung diaplikasikan proses perhitungan tersebut.

##### 4.4.5.1 Jalur Router 1A ke Router 1B

Tabel 4.9 menunjukkan statistik angka waktu *delay* dari ketiga protokol *routing* yang diujikan pada jalur jaringan *Router 1A* ke *Router 1B*. Data yang diambil adalah data hasil hitung 3 *delay* terawal yang telah dihitung sebelumnya.

Tabel 4.9 Statistik Data Waktu Delay Paket Data Jalur RIA-RIB

| No. | Protokol<br>Routing    | Node     | Waktu delay ke- <i>i</i><br>(detik) | Waktu delay ke- <i>i+1</i><br>(detik) |
|-----|------------------------|----------|-------------------------------------|---------------------------------------|
| 1.  | Statis                 | aarch-13 | 0,535256 detik                      | 0,521619 detik                        |
|     |                        |          | 0,521619 detik                      | 0,520850 detik                        |
|     |                        |          | 0,520850 detik                      | 0,522604 detik                        |
|     |                        | esp-1    | 0,000287 detik                      | 17,565919 detik                       |
|     |                        |          | 17,565919 detik                     | 0,019439 detik                        |
|     |                        |          | 0,019439 detik                      | 1,612929 detik                        |
|     |                        | esp-3    | 0,194726 detik                      | 0,828032 detik                        |
|     |                        |          | 0,828032 detik                      | 1,008321 detik                        |
|     |                        |          | 1,008321 detik                      | 0,000096 detik                        |
| 2.  | Dinamis<br><i>RIP</i>  | aarch-13 | 0,512980 detik                      | 0,513956 detik                        |
|     |                        |          | 0,513956 detik                      | 0,510599 detik                        |
|     |                        |          | 0,510599 detik                      | 0,574915 detik                        |
|     |                        | esp-1    | 0,004083 detik                      | 0,000064 detik                        |
|     |                        |          | 0,000064 detik                      | 1,062885 detik                        |
|     |                        |          | 1,062885 detik                      | 1,127319 detik                        |
|     |                        | esp-3    | 1,095233 detik                      | 1,020080 detik                        |
|     |                        |          | 1,020080 detik                      | 0,000103 detik                        |
|     |                        |          | 0,000103 detik                      | 1,295443 detik                        |
| 3.  | Dinamis<br><i>OSPF</i> | aarch-13 | 0,514920 detik                      | 0,516760 detik                        |
|     |                        |          | 0,516760 detik                      | 0,506446 detik                        |
|     |                        |          | 0,506446 detik                      | 0,514549 detik                        |
|     |                        | esp-1    | 1,010783 detik                      | 6,518624 detik                        |
|     |                        |          | 6,518624 detik                      | 21,966578 detik                       |
|     |                        |          | 21,966578 detik                     | 5,799702 detik                        |
|     |                        | esp-3    | 3,689232 detik                      | 20,76146 detik                        |
|     |                        |          | 20,76146 detik                      | 1,049916 detik                        |
|     |                        |          | 1,049916 detik                      | 3,070790 detik                        |

Kemudian dilakukan perhitungan *jitter* menggunakan rumus hitung *jitter* pada ketiga protokol *routing* pada *node* jaringan yang dipilih.

Protokol *routing* statis *node aarch64-13*.

- $Jitter\ 1 = [(0,521619 - 0,535256) - 0] / 16 = | -0,000852\ detik | = 0,000852\ detik$
- $Jitter\ 2 = [(0,520850 - 0,521619) - 0,000852] / 16 = | -0,000101\ detik | = 0,000101\ detik$
- $Jitter\ 3 = [(0,522604 - 0,520850) - 0,000101] / 16 = | 0,000103\ detik | = 0,000103\ detik$
- Rerata  $= (0,000852 + 0,000101 + 0,000103) / 3 = 0,000352\ detik$

Protokol *routing* statis *node esp-1*.

- $Jitter\ 1 = [(17,565919 - 0,000287) - 0] / 16 = | 1,097852\ detik | = 1,097852\ detik$
- $Jitter\ 2 = [(0,019439 - 17,565919) - 1,097852] / 16 = | -1,165270\ detik | = 1,165270\ detik$
- $Jitter\ 3 = [(1,612929 - 0,019439) - 1,165270] / 16 = | 0,026763\ detik | = 0,026763\ detik$
- Rerata  $= (1,097852 + 1,165270 + 0,026763) / 3 = 0,763295\ detik$

Protokol *routing* statis *node esp-3*.

- $Jitter\ 1 = [(0,828032 - 0,194726) - 0] / 16 = | 0,039581\ detik | = 0,039581\ detik$
- $Jitter\ 2 = [(1,008321 - 0,828032) - 0,039581] / 16 = | 0,008794\ detik | = 0,008794\ detik$
- $Jitter\ 3 = [(0,000096 - 1,008321) - 0,008794] / 16 = | -0,063563\ detik | = 0,063563\ detik$
- Rerata  $= (0,039581 + 0,008794 + 0,063563) / 3 = 0,037313\ detik$

Protokol *routing* dinamis *RIP node aarch64-13.*

- $Jitter\ 1 = [(0,513956 - 0,512980) - 0] / 16 = | 0,000061\ detik |$   
 $= 0,000061\ detik$
- $Jitter\ 2 = [(0,510599 - 0,513956) - 0,000061] / 16 = | -0,000214\ detik |$   
 $= 0,000214\ detik$
- $Jitter\ 3 = [(0,574915 - 0,510599) - 0,000214] / 16 = | 0,004006\ detik |$   
 $= 0,004006\ detik$
- Rerata  $= (0,000061 + 0,000214 + 0,004006) / 3 = 0,001427\ detik$

Protokol *routing* dinamis *RIP node esp-1.*

- $Jitter\ 1 = [(0,000064 - 0,004083) - 0] / 16 = | -0,000251\ detik |$   
 $= 0,000251\ detik$
- $Jitter\ 2 = [(1,062885 - 0,000064) - 0,000251] / 16 = | 0,066410\ detik |$   
 $= 0,066410\ detik$
- $Jitter\ 3 = [(1,127319 - 1,062885) - 0,066410] / 16 = | -0,000123\ detik |$   
 $= 0,000123\ detik$
- Rerata  $= (0,000251 + 0,066410 + 0,000123) / 3 = 0,022261\ detik$

Protokol *routing* dinamis *RIP node esp-3.*

- $Jitter\ 1 = [(1,020080 - 1,095233) - 0] / 16 = | -0,004697\ detik |$   
 $= 0,004697\ detik$
- $Jitter\ 2 = [(0,000103 - 1,020080) - 0,004697] / 16 = | -0,064042\ detik |$   
 $= 0,064042\ detik$
- $Jitter\ 3 = [(1,295443 - 0,000103) - 0,064042] / 16 = | 0,076954\ detik |$   
 $= 0,076954\ detik$
- Rerata  $= (0,004697 + 0,064042 + 0,076954) / 3 = 0,048564\ detik$

Protokol *routing* dinamis *OSPF node aarch64-13.*

- $Jitter\ 1 = [(0,516760 - 0,514920) - 0] / 16 = | 0,000115\ detik |$   
 $= 0,000115\ detik$
- $Jitter\ 2 = [(0,506446 - 0,516760) - 0,000115] / 16 = | -0,000651\ detik |$

$$= 0,000651 \text{ detik}$$

- $Jitter 3 = [(0,514549 - 0,506446) - 0,000651] / 16 = | 0,000465 \text{ detik} |$   
 $= 0,000465 \text{ detik}$
- Rerata  $= (0,000115 + 0,000651 + 0,000465) / 3 = 0,000410 \text{ detik}$

Protokol *routing* dinamis *OSPF node esp-1*.

- $Jitter 1 = [(6,518624 - 1,010783) - 0] / 16 = | 0,344240 \text{ detik} |$   
 $= 0,344240 \text{ detik}$
- $Jitter 2 = [(21,966578 - 6,518624) - 0,344240] / 16 = | 0,943982 \text{ detik} |$   
 $= 0,943982 \text{ detik}$
- $Jitter 3 = [(5,799702 - 21,966578) - 0,943982] / 16 = | -1,069428 \text{ detik} |$   
 $= 1,069428 \text{ detik}$
- Rerata  $= (0,344240 + 0,943982 + 1,069428) / 3 = 0,785883 \text{ detik}$

Protokol *routing* dinamis *OSPF node esp-3*.

- $Jitter 1 = [(20,76146 - 3,689232) - 0] / 16 = | 1,067014 \text{ detik} |$   
 $= 1,067014 \text{ detik}$
- $Jitter 2 = [(1,049916 - 20,76146) - 1,067014] / 16 = | -1,298659 \text{ detik} |$   
 $= 1,298659 \text{ detik}$
- $Jitter 3 = [(3,070790 - 1,049916) - 1,298659] / 16 = | 0,045138 \text{ detik} |$   
 $= 0,045138 \text{ detik}$
- Rerata  $= (1,067014 + 1,298659 + 0,045138) / 3 = 0,803603 \text{ detik}$

#### 4.4.5.2 Jalur Router 1B ke Router 4

Tabel 4.10 menunjukkan statistik angka waktu *delay* dari ketiga protokol *routing* yang diujikan pada jalur jaringan *Router 1B* ke *Router 4*. Data yang diambil adalah data hasil hitung 3 *delay* terawal yang telah dihitung sebelumnya.

Tabel 4.10 Statistik Data Waktu Delay Paket Data Jalur R1B-R4

| No. | Protokol<br>Routing    | Node     | Waktu delay ke- <i>i</i><br>(detik) | Waktu delay ke- <i>i+1</i><br>(detik) |
|-----|------------------------|----------|-------------------------------------|---------------------------------------|
| 1.  | Statis                 | aarch-13 | 0,539867 detik                      | 0,531108 detik                        |
|     |                        |          | 0,531108 detik                      | 0,524261 detik                        |
|     |                        |          | 0,524261 detik                      | 0,528232 detik                        |
|     |                        | esp-1    | 2,260627 detik                      | 21,292448 detik                       |
|     |                        |          | 21,292448 detik                     | 4,236445 detik                        |
|     |                        |          | 4,236445 detik                      | 18,052331 detik                       |
|     |                        | esp-3    | 2,031708 detik                      | 1,409184 detik                        |
|     |                        |          | 1,409184 detik                      | 0,760450 detik                        |
|     |                        |          | 0,760450 detik                      | 11,352334 detik                       |
| 2.  | Dinamis<br><i>RIP</i>  | aarch-13 | 0,512980 detik                      | 0,513956 detik                        |
|     |                        |          | 0,513956 detik                      | 0,510599 detik                        |
|     |                        |          | 0,510599 detik                      | 0,581050 detik                        |
|     |                        | esp-1    | 1,144625 detik                      | 35,305937 detik                       |
|     |                        |          | 35,305937 detik                     | 2,058524 detik                        |
|     |                        |          | 2,058524 detik                      | 2,007081 detik                        |
|     |                        | esp-3    | 8,096692 detik                      | 5,761966 detik                        |
|     |                        |          | 5,761966 detik                      | 13,055046 detik                       |
|     |                        |          | 13,055046 detik                     | 150,980024 detik                      |
| 3.  | Dinamis<br><i>OSPF</i> | aarch-13 | 0,516869 detik                      | 0,520233 detik                        |
|     |                        |          | 0,520233 detik                      | 0,509367 detik                        |
|     |                        |          | 0,509367 detik                      | 0,515881 detik                        |
|     |                        | esp-1    | 59,890572 detik                     | 3,257282 detik                        |
|     |                        |          | 3,257282 detik                      | 3,015181 detik                        |
|     |                        |          | 3,015181 detik                      | 2,116821 detik                        |
|     |                        | esp-3    | 4,049832 detik                      | 7,126164 detik                        |
|     |                        |          | 7,126164 detik                      | 0,992505 detik                        |
|     |                        |          | 0,992505 detik                      | 1,911778 detik                        |

Kemudian dilakukan perhitungan *jitter* menggunakan rumus hitung *jitter* pada ketiga protokol *routing* pada *node* jaringan yang dipilih.

Protokol *routing* statis *node aarch64-13*.

- $Jitter\ 1 = [(0,531108 - 0,539867) - 0] / 16 = |-0,000547\ detik| = 0,000547\ detik$
- $Jitter\ 2 = [(0,524261 - 0,531108) - 0,000547] / 16 = |-0,000462\ detik| = 0,000462\ detik$
- $Jitter\ 3 = [(0,528232 - 0,524261) - 0,000462] / 16 = |0,000219\ detik| = 0,000219\ detik$
- Rerata  $= (0,000547 + 0,000462 + 0,000219) / 3 = 0,000409\ detik$

Protokol *routing* statis *node esp-1*.

- $Jitter\ 1 = [(3,257282 - 59,890572) - 0] / 16 = |-3,539580\ detik| = 3,539580\ detik$
- $Jitter\ 2 = [(3,015181 - 3,257282) - 3,539580] / 16 = |-0,236355\ detik| = 0,236355\ detik$
- $Jitter\ 3 = [(2,116821 - 3,015181) - 0,236355] / 16 = |-0,070919\ detik| = 0,070919\ detik$
- Rerata  $= (3,539580 + 0,236355 + 0,070919) / 3 = 1,282285\ detik$

Protokol *routing* statis *node esp-3*.

- $Jitter\ 1 = [(1,409184 - 2,031708) - 0] / 16 = |-0,038907\ detik| = 0,038907\ detik$
- $Jitter\ 2 = [(0,760450 - 1,409184) - 0,038907] / 16 = |-0,042977\ detik| = 0,042977\ detik$
- $Jitter\ 3 = [(11,352334 - 0,760450) - 0,042977] / 16 = |0,659306\ detik| = 0,659306\ detik$
- Rerata  $= (0,038907 + 0,042977 + 0,659306) / 3 = 0,247063\ detik$

Protokol *routing* dinamis *RIP node aarch64-13.*

- $Jitter\ 1 = [(0,513956 - 0,512980) - 0] / 16 = | 0,000061\ detik | = 0,000061\ detik$
- $Jitter\ 2 = [(0,510599 - 0,513956) - 0,000061] / 16 = | -0,000214\ detik | = 0,000214\ detik$
- $Jitter\ 3 = [(0,581050 - 0,510599) - 0,000214] / 16 = | 0,004389\ detik | = 0,004389\ detik$
- Rerata  $= (0,000061 + 0,000214 + 0,004389) / 3 = 0,001554\ detik$

Protokol *routing* dinamis *RIP node esp-1.*

- $Jitter\ 1 = [(35,305937 - 1,144625) - 0] / 16 = | 2,135082\ detik | = 2,135082\ detik$
- $Jitter\ 2 = [(2,058524 - 35,305937) - 2,135082] / 16 = | -2,211405\ detik | = 2,211405\ detik$
- $Jitter\ 3 = [(2,007081 - 2,058524) - 2,211405] / 16 = | -0,141428\ detik | = 0,141428\ detik$
- Rerata  $= (2,135082 + 2,211405 + 0,141428) / 3 = 1,495971\ detik$

Protokol *routing* dinamis *RIP node esp-3.*

- $Jitter\ 1 = [(5,761966 - 8,096692) - 0] / 16 = | -0,145920\ detik | = 0,145920\ detik$
- $Jitter\ 2 = [(13,055046 - 5,761966) - 0,145920] / 16 = | 0,446697\ detik | = 0,446697\ detik$
- $Jitter\ 3 = [(150,980024 - 13,055046) - 0,446697] / 16 = | 8,592392\ detik | = 8,592392\ detik$
- Rerata  $= (0,145920 + 0,446697 + 8,592392) / 3 = 3,061669\ detik$

Protokol *routing* dinamis *OSPF node aarch64-13.*

- $Jitter\ 1 = [(0,520233 - 0,516869) - 0] / 16 = | 0,000210\ detik | = 0,000210\ detik$
- $Jitter\ 2 = [(0,509367 - 0,520233) - 0,000210] / 16 = | -0,000692\ detik |$

$$= 0,000692 \text{ detik}$$

- $Jitter 3 = [(0,515881 - 0,509367) - 0,000692] / 16 = | 0,000363 \text{ detik} |$   
 $= 0,000363 \text{ detik}$
- Rerata  $= (0,000210 + 0,000692 + 0,000363) / 3 = 0,000421 \text{ detik}$

Protokol *routing* dinamis *OSPF node esp-1*.

- $Jitter 1 = [(3,257282 - 59,890572) - 0] / 16 = | -3,539580 \text{ detik} |$   
 $= 3,539580 \text{ detik}$
- $Jitter 2 = [(3,015181 - 3,257282) - 3,539580] / 16 = | -0,236355 \text{ detik} |$   
 $= 0,236355 \text{ detik}$
- $Jitter 3 = [(2,116821 - 3,015181) - 0,236355] / 16 = | -0,070919 \text{ detik} |$   
 $= 0,070919 \text{ detik}$
- Rerata  $= (3,539580 + 0,236355 + 0,070919) / 3 = 1,282285 \text{ detik}$

Protokol *routing* dinamis *OSPF node esp-3*.

- $Jitter 1 = [(7,126164 - 4,049832) - 0] / 16 = | 0,192270 \text{ detik} |$   
 $= 0,192270 \text{ detik}$
- $Jitter 2 = [(0,992505 - 7,126164) - 0,192270] / 16 = | -0,395370 \text{ detik} |$   
 $= 0,395370 \text{ detik}$
- $Jitter 3 = [(1,911778 - 0,992505) - 0,395370] / 16 = | 0,032743 \text{ detik} |$   
 $= 0,032743 \text{ detik}$
- Rerata  $= (0,192270 + 0,395370 + 0,032743) / 3 = 0,206794 \text{ detik}$

#### 4.4.6 Faktor yang Memengaruhi Hasil Analisis QoS

Setelah pengambilan dan analisis data dilakukan sepenuhnya, terdapat beberapa faktor yang berkemungkinan memengaruhi dan memberikan hasil pengujian yang diluar perkiraan awal. Faktor-faktor ini dapat memberikan pengaruh terhadap hasil pengambilan dan analisis data baik secara sebagian maupun sepenuhnya.

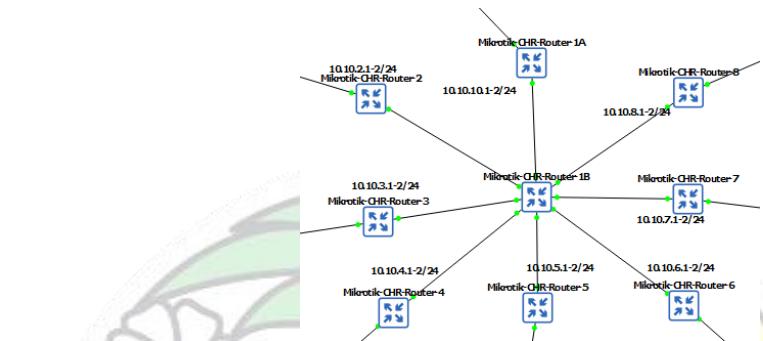
##### 4.4.6.1 Koneksi Internet

Koneksi internet dari *Router* virtual ke penyedia internet memiliki kemungkinan untuk menimbulkan terjadinya perubahan dan variasi kecepatan dari satu waktu

ke waktu lainnya. Perbedaan variasi kecepatan tentu saja dapat menimbulkan perbedaan hasil pada keseluruhan parameter *QoS* yang diujikan. Internet juga mungkin saja mengalami gangguan yang tidak dapat disadari dan dideteksi ketika pengujian sedang berlangsung.

#### 4.4.6.2 Penyumbatan Jalur Jaringan

Gambar 4.26 menunjukkan *Router 1B* yang terhubung dan mengirimkan serta menerima paket data dari berbagai macam jalur jaringan.



Gambar 4.25 *Router 1B* Terhubung ke Seluruh Jalur Jaringan

Terlihat pada gambar 4.26 bahwa *Router 1B* memiliki keadaan yang memungkinkan *router* tersebut menerima dan mengirim data ke jalur jaringan cabang dari *router* 2 sampai ke *router* 8 dengan di saat yang sama harus mengirimkan dan menerima data-data tersebut ke satu jalur yang ada ke *router* 1A. Perbedaan jumlah jalur ini membuat *router* 1B sangat mungkin mengalami penyumbatan data dalam prosesnya mengirim dan menerima data ke tujuan seharusnya. Penyumbatan dapat menimbulkan perubahan hasil data pada parameter *QoS* yang tidak dapat dicegah dan dideteksi saat proses pengujian sedang berlangsung.

#### 4.4.6.3 Perbedaan Arsitektur Prosessor Perangkat IoT

Setelah semua perhitungan parameter *QoS* dilakukan, terdapat beberapa perhitungan yang menunjukkan perbedaan angka yang signifikan terutama pada perangkat *IoT* yang berarsitektur prosessor *esp32* dibandingkan dengan perangkat *IoT* yang berarsitektur prosessor *ARM64/Aarch64*. Hal ini dapat menunjukkan bahwa lebih rendahnya kemampuan komputasi yang dimiliki oleh *esp32* dibandingkan dengan *aarch64* menimbulkan kemungkinan lebih lambatnya

proses pengolahan data untuk mengirim dan menerima data di arsitektur prosessor tersebut sehingga baik prosessor berbasis *esp32* ataupun protokol *routing* yang diujikan tidak dapat benar-benar menyesuaikan waktu pengiriman dan penerimaan data yang sesuai untuk sistem *esp32* sehingga terjadi perbedaan kemampuan yang signifikan antara perangkat *IoT* yang berarsitektur prosessor *esp32* dengan perangkat *IoT* yang berarsitektur prosessor *ARM64/Aarch64*.

#### **4.4.7 Hasil Akhir Keseluruhan Analisis QoS**

Setelah seluruh proses analisis berdasarkan *Quality of Service (QoS)* telah dilakukan. Ditampilkan seluruh hasil analisis untuk menentukan protokol *routing* yang terbaik untuk penggunaan di perangkat berbasis *Internet of Things (IoT)* berdasarkan ketiga protokol *routing* yang diujikan.

##### **4.4.7.1 Hasil Akhir Keseluruhan Analisis QoS Berdasarkan Throughput**

Tabel 4.11 menunjukkan hasil keseluruhan analisis *QoS* berdasarkan parameter *throughput* yang telah dilakukan di ketiga protokol *routing* yang diujikan di dua jalur jaringan *RIA-RIB* dan *RIB-R4*.

Untuk analisis *throughput*, hasil *throughput* terbaik ditunjukkan dengan hasil yang terbesar dan hasil *throughput* terburuk ditunjukkan dengan hasil yang terkecil. Notasi *throughput* ditunjukkan sebagai kilobit per detik (Kb/detik)

Tabel 4.11 Hasil Analisis Berdasarkan *Throughput*

| No. | Protokol <i>Routing</i> | Jalur Jaringan | Throughput     |
|-----|-------------------------|----------------|----------------|
| 1.  | Statis                  | <i>RIA-RIB</i> | 3.436 Kb/detik |
|     |                         | <i>RIB-R4</i>  | 492 Kb/detik   |
| 2.  | Dinamis <i>RIP</i>      | <i>RIA-RIB</i> | 2.994 Kb/detik |
|     |                         | <i>RIB-R4</i>  | 431 Kb/detik   |
| 3.  | Dinamis <i>OSPF</i>     | <i>RIA-RIB</i> | 3.014 Kb/detik |
|     |                         | <i>RIB-R4</i>  | 435 Kb/detik   |

Berdasarkan tabel 4.11, baik di jalur jaringan *RIA-RIB* maupun *RIB-R4* menunjukkan hasil bahwa *throughput* terbesar pada protokol *routing* statis, lalu dinamis *OSPF*, dan terendah ditunjukkan di protokol *routing* dinamis *RIP*. Dengan

demikian, berdasarkan parameter *throughput*. Protokol *routing* statis berada pada urutan terbaik pertama, lalu *routing* dinamis *OSPF* di urutan kedua, dan *routing* dinamis *RIP* di urutan terakhir.

Hal ini menunjukkan bahwa protokol *routing* statis memiliki kemampuan terbaik untuk memberikan performa *throughput* terbesar pada jaringan yang berbasis *IoT*, kemudian selanjutnya pada protokol *routing* dinamis *OSPF*, dan yang berkemampuan terendah untuk memberikan performa *throughput* adalah protokol *routing* dinamis *RIP*.

#### **4.4.7.2 Hasil Akhir Keseluruhan Analisis QoS Berdasarkan Packet Loss**

Tabel 4.12 menunjukkan hasil keseluruhan analisis *QoS* berdasarkan parameter *packet loss* yang telah dilakukan di ketiga protokol *routing* yang diujikan di dua jalur jaringan *R1A-R1B* dan *R1B-R4*.

Untuk analisis *packet loss*, hasil *packet loss* terbaik ditunjukkan dengan hasil yang terkecil dan hasil *packet loss* terburuk ditunjukkan dengan hasil yang terbesar. Notasi *packet loss* ditunjukkan dalam persen.

Tabel 4.12 Hasil Analisis Berdasarkan *Packet Loss*

| No. | Protokol <i>Routing</i> | Jalur Jaringan | Packet Loss |
|-----|-------------------------|----------------|-------------|
| 1.  | Statis                  | <i>R1A-R1B</i> | 26,8%       |
|     |                         | <i>R1B-R4</i>  | 30,1%       |
| 2.  | Dinamis <i>RIP</i>      | <i>R1A-R1B</i> | 28,5%       |
|     |                         | <i>R1B-R4</i>  | 30,8%       |
| 3.  | Dinamis <i>OSPF</i>     | <i>R1A-R1B</i> | 19,3%       |
|     |                         | <i>R1B-R4</i>  | 24,3%       |

Berdasarkan tabel 4.12, baik di jalur jaringan *R1A-R1B* maupun *R1B-R4* menunjukkan hasil bahwa *packet loss* terbesar pada protokol *routing* dinamis *RIP*, lalu *routing* statis, dan terendah pada *routing* dinamis *OSPF*. Dengan demikian, berdasarkan parameter *packet loss*. Protokol *routing* dinamis *OSPF* berada pada urutan terbaik pertama, lalu *routing* statis di urutan kedua, dan *routing* dinamis *RIP* di urutan terakhir.

Hal ini menunjukkan bahwa protokol *routing* dinamis *OSPF* memiliki kemampuan terbaik untuk menangani kejadian *packet loss* pada jaringan yang berbasis *IoT*, kemudian selanjutnya pada protokol *routing* statis, dan yang berkemampuan terendah untuk menangani *packet loss* adalah protokol *routing* dinamis *RIP*.

#### 4.4.7.3 Hasil Akhir Keseluruhan Analisis QoS Berdasarkan Delay

Tabel 4.13 menunjukkan hasil rerata keseluruhan analisis *QoS* berdasarkan parameter *delay* yang telah dilakukan di ketiga protokol *routing* yang diujikan di dua jalur jaringan *R1A-R1B* dan *R1B-R4* dan pada *node* yang telah dipilih untuk pengujian.

Untuk analisis *delay*, hasil *delay* terbaik ditunjukkan dengan hasil yang terkecil dan hasil *delay* terburuk ditunjukkan dengan hasil yang terbesar. Notasi *delay* ditunjukkan dalam detik dan *node* *aarch-13*, *esp-1*, dan *esp-3* dipilih sebagai sampel data.

Tabel 4.13 Hasil Analisis Berdasarkan *Delay*

| No. | Protokol<br><i>Routing</i> | Node            | Rerata <i>Delay</i> jalur<br><i>R1A-R1B</i> (detik) | Rerata <i>Delay</i> jalur<br><i>R1B-R4</i> (detik) |
|-----|----------------------------|-----------------|-----------------------------------------------------|----------------------------------------------------|
| 1.  | Statis                     | <i>aarch-13</i> | 0,525908 detik                                      | 0,531745 detik                                     |
|     |                            | <i>esp-1</i>    | 5,861881 detik                                      | 9,263173 detik                                     |
|     |                            | <i>esp-3</i>    | 0,677026 detik                                      | 1,400447 detik                                     |
| 2.  | Dinamis<br><i>RIP</i>      | <i>aarch-13</i> | 0,512511 detik                                      | 0,512511 detik                                     |
|     |                            | <i>esp-1</i>    | 0,355677 detik                                      | 12,836362 detik                                    |
|     |                            | <i>esp-3</i>    | 0,705138 detik                                      | 8,971234 detik                                     |
| 3.  | Dinamis<br><i>OSPF</i>     | <i>aarch-13</i> | 0,512708 detik                                      | 0,515522 detik                                     |
|     |                            | <i>esp-1</i>    | 9,831995 detik                                      | 22,054345 detik                                    |
|     |                            | <i>esp-3</i>    | 8,500202 detik                                      | 4,056167 detik                                     |

Berdasarkan tabel 4.13, baik di jalur jaringan *R1A-R1B* maupun *R1B-R4* menunjukkan hasil bahwa *delay* terbesar pada protokol *routing* dinamis *OSPF*, lalu *routing* dinamis *RIP*, dan terendah pada protokol *routing* statis. Dengan

demikian, berdasarkan parameter *delay*. Protokol *routing* statis berada pada urutan terbaik pertama, lalu *routing* dinamis *RIP* di urutan kedua, dan *routing* dinamis *OSPF* di urutan terakhir.

Selain itu, angka *delay* terbesar terlihat pada *node esp-1* baik pada ketiga protokol *routing* yang diujikan, pada *node esp-3* terlihat pada protokol dinamis *OSPF* dan dinamis *RIP* memiliki angka *delay* yang besar. *Node aarch-13* memiliki angka yang stabil di ketiga protokol yang diujikan.

Hal ini menunjukkan bahwa protokol *routing* statis memiliki kemampuan terbaik untuk menangani kejadian *delay* pada jaringan yang berbasis *IoT*, kemudian pada protokol *routing* dinamis *RIP*, dan yang berkemampuan terendah untuk menangani *packet loss* adalah protokol *routing* dinamis *OSPF*. Selain itu, terlihat bahwa perangkat *IoT* yang memiliki basis prosessor *esp32 (espressif)* lebih rentan mengalami *delay* dibandingkan dengan perangkat *IoT* yang memiliki basis prosessor *arm64*.

#### 4.4.7.4 Hasil Akhir Keseluruhan Analisis QoS Berdasarkan Latency

Tabel 4.14 menunjukkan hasil rerata keseluruhan analisis *QoS* berdasarkan parameter *latency* yang telah dilakukan di ketiga protokol *routing* yang diujikan di dua jalur jaringan *R1A-R1B* dan *R1B-R4* dan pada *node* yang telah dipilih untuk pengujian.

Untuk analisis *latency*, hasil *latency* terbaik ditunjukkan dengan hasil yang terkecil dan hasil *latency* terburuk ditunjukkan dengan hasil yang terbesar. Notasi *packet loss* ditunjukkan dalam detik.

Tabel 4.14 Hasil Analisis Berdasarkan *Latency*

| No. | Protokol <i>Routing</i> | Jalur Jaringan | Latency        |
|-----|-------------------------|----------------|----------------|
| 1.  | Statis                  | <i>R1A-R1B</i> | 1,177469 detik |
|     |                         | <i>R1B-R4</i>  | 1,865894 detik |
| 2.  | Dinamis <i>RIP</i>      | <i>R1A-R1B</i> | 0,262220 detik |
|     |                         | <i>R1B-R4</i>  | 3,720017 detik |
| 3.  | Dinamis <i>OSPF</i>     | <i>R1A-R1B</i> | 3,140817 detik |
|     |                         | <i>R1B-R4</i>  | 4,437672 detik |

Berdasarkan tabel 4.14, dapat diambil kesimpulan bahwa baik di jalur jaringan *R1A-R1B* maupun *R1B-R4* menunjukkan hasil bahwa *latency* terbesar terdapat pada protokol *routing* dinamis *OSPF*, lalu statis, dan terendah pada dinamis *RIP*. Dengan demikian, berdasarkan parameter *latency*. Protokol *routing* dinamis *RIP* berada pada urutan terbaik pertama, lalu *routing* statis di urutan kedua, dan *routing* dinamis *OSPF* di urutan terakhir.

Hal ini menunjukkan bahwa protokol *routing* dinamis *OSPF* memiliki kemampuan terburuk untuk menangani *latency* pada jaringan yang berbasis *IoT*, kemudian selanjutnya pada protokol *routing* statis, dan yang memiliki *latency* terendah pada protokol *routing* dinamis *RIP*.

#### 4.4.7.5 Hasil Akhir Keseluruhan Analisis QoS Berdasarkan Jitter

Tabel 4.15 menunjukkan hasil rerata keseluruhan analisis *QoS* berdasarkan parameter *jitter* yang telah dilakukan di ketiga protokol *routing* yang diujikan di dua jalur jaringan *R1A-R1B* dan *R1B-R4* dan pada *node* yang telah dipilih untuk pengujian.

Untuk analisis *jitter*, hasil *jitter* terbaik ditunjukkan dengan hasil yang terkecil dan hasil *jitter* terburuk ditunjukkan dengan hasil yang terbesar. Notasi *delay* ditunjukkan dalam detik dan *node* *aarch-13*, *esp-1*, dan *esp-3* dipilih sebagai sampel data.

Tabel 4.15 Hasil Analisis Berdasarkan *Jitter*

| No. | Protokol<br><i>Routing</i> | Node            | Rerata <i>Jitter</i> jalur<br><i>R1A-R1B</i> (detik) | Rerata <i>Jitter</i> jalur<br><i>R1B-R4</i> (detik) |
|-----|----------------------------|-----------------|------------------------------------------------------|-----------------------------------------------------|
| 1.  | Statis                     | <i>aarch-13</i> | 0,000352 detik                                       | 0,000409 detik                                      |
|     |                            | <i>esp-1</i>    | 0,763295 detik                                       | 1,040684 detik                                      |
|     |                            | <i>esp-3</i>    | 0,037313 detik                                       | 0,247063 detik                                      |
| 2.  | Dinamis<br><i>RIP</i>      | <i>aarch-13</i> | 0,001427 detik                                       | 0,001554 detik                                      |
|     |                            | <i>esp-1</i>    | 0,022261 detik                                       | 1,495971 detik                                      |
|     |                            | <i>esp-3</i>    | 0,048564 detik                                       | 3,061669 detik                                      |
| 3.  | Dinamis<br><i>OSPF</i>     | <i>aarch-13</i> | 0,000410 detik                                       | 0,000421 detik                                      |
|     |                            | <i>esp-1</i>    | 0,785883 detik                                       | 1,282285 detik                                      |

| No. | Protokol<br><i>Routing</i> | <i>Node</i>  | Rerata <i>Jitter</i> jalur<br><i>RIA-RIB</i> (detik) | Rerata <i>Jitter</i> jalur<br><i>RIB-R4</i> (detik) |
|-----|----------------------------|--------------|------------------------------------------------------|-----------------------------------------------------|
|     |                            | <i>esp-3</i> | 0,803603 detik                                       | 0,206794 detik                                      |

Berdasarkan tabel 4.15, baik di jalur jaringan *RIA-RIB* maupun *RIB-R4* menunjukkan hasil bahwa *jitter* terbesar pada protokol *routing* dinamis *RIP*, lalu *routing* dinamis *OSPF*, dan terendah pada protokol *routing* statis. Dengan demikian, berdasarkan parameter *jitter*. Protokol *routing* statis berada pada urutan terbaik pertama, lalu *routing* dinamis *OSPF* di urutan kedua, dan *routing* dinamis *RIP* di urutan terakhir.

Selain itu, angka *jitter* terbesar terlihat pada *node esp-1* pada ketiga protokol *routing* dan *esp-3* pada jalur *RIB-R4* protokol *routing RIP*. *Node aarch-13* memiliki angka yang stabil di ketiga protokol yang diujikan.

Hal ini menunjukkan bahwa protokol *routing* statis memiliki variasi *delay* atau *jitter* terendah pada jaringan yang berbasis *IoT*, selanjutnya pada protokol *routing* dinamis *OSPF*, dan yang memiliki *jitter* tertinggi adalah protokol *routing* dinamis *RIP*. Selain itu, terlihat juga bahwa perangkat *IoT* yang memiliki basis prosessor *esp32 (espressif)* lebih rentan mengalami *delay* dibandingkan dengan perangkat *IoT* yang memiliki basis prosessor *arm64*.

#### 4.4.8 Ringkasan Akhir Untuk Keseluruhan Hasil Analisis QoS

Setelah seluruh hasil analisis berdasarkan *Quality of Service (QoS)* telah ditampilkan. Ditunjukkan ringkasan akhir dari seluruh analisis parameter *Quality of Service (QoS)* dalam bentuk peringkat angka untuk setiap protokol *routing* yang diujikan.

Peringkat angka ditunjukkan dari angka 1 sampai dengan angka 3. Dengan angka 1 menunjukkan protokol *routing* yang lebih baik di parameter *QoS* yang dipilih sebaliknya angka 3 menunjukkan protokol *routing* yang lebih buruk di parameter *QoS* yang dipilih. Peringkat angka ditunjukkan pada tabel 4.16

Tabel 4.16 Ringkasan Hasil Analisis Dengan Peringkat Angka

| No. | Parameter <i>QoS</i> | Protokol <i>Routing</i><br>Statis | Protokol <i>Routing</i><br>Dinamis <i>RIP</i> | Protokol <i>Routing</i><br>Dinamis <i>OSPF</i> |
|-----|----------------------|-----------------------------------|-----------------------------------------------|------------------------------------------------|
| 1.  | <i>Throughput</i>    | 1                                 | 3                                             | 2                                              |

| No. | Parameter <i>QoS</i> | Protokol <i>Routing</i><br>Statis | Protokol <i>Routing</i><br>Dinamis <i>RIP</i> | Protokol <i>Routing</i><br>Dinamis <i>OSPF</i> |
|-----|----------------------|-----------------------------------|-----------------------------------------------|------------------------------------------------|
| 2.  | <i>Packet Loss</i>   | 2                                 | 3                                             | 1                                              |
| 3.  | <i>Delay</i>         | 1                                 | 2                                             | 3                                              |
| 4.  | <i>Latency</i>       | 2                                 | 1                                             | 3                                              |
| 5.  | <i>Jitter</i>        | 1                                 | 3                                             | 2                                              |

Berdasarkan tabel 4.16, sebagaimana penelitian yang telah dilakukan. Pemeringkatan menunjukkan bahwa secara umum protokol *routing* statis memiliki performa yang terbaik dengan keunggulan di parameter *QoS Throughput*, *Delay*, dan *Jitter* serta tidak ada angka 3 pada hasil ringkasan analisis berdasarkan tabel 4.16. Kemudian performa menengah pada protokol *routing* dinamis *OSPF* dengan keunggulan di parameter *QoS Packet Loss* dan terdapat 2 buah angka 3 pada hasil ringkasan analisis berdasarkan tabel 4.16. Lalu, performa terburuk secara umum pada protokol *routing* dinamis *RIP* dengan keunggulan di parameter *QoS Latency* dan terdapat 3 buah angka 3 pada hasil ringkasan analisis berdasarkan tabel 4.16.

Protokol *routing* statis secara umum lebih baik disebabkan karena protokol ini termasuk protokol yang memiliki cara kerja yang sederhana, protokol ini hanya menyimpan datar tabel *routing* yang dimasukkan oleh ahli jaringan secara manual dan tidak memiliki algoritma serta kemampuan untuk mencari jalur lain apabila jalur yang dipilih mengalami gangguan kecuali apabila terdapat intervensi kembali oleh ahli jaringan untuk mengubah jalur jaringan tersebut. Kesederhanaan protokol *routing* statis sesuai untuk implementasi jaringan *IoT* yang berjumlah sedikit atau untuk jaringan yang dapat terhubung langsung ke internet.

Protokol *routing* dinamis *OSPF* secara umum menjadi protokol *routing* yang berperforma menengah disebabkan oleh adanya proses algoritma *Link-State* dan Djikstra untuk mencari jalur terbaik untuk dilewati oleh paket data sehingga memberikan sedikit hambatan untuk proses transfer paket data. Walaupun demikian, algoritma *Link-State* dan Djikstra memiliki sisi positif yaitu memungkinkan protokol untuk mencari jalur lain apabila jalur utama mengalami gangguan atau terjadinya penambahan *router* atau terdapat jalur baru di jaringan sehingga sistem dapat bekerja

secara normal tanpa perlu intervensi dari ahli jaringan. Protokol *routing* dinamis seperti *OSPF* sesuai untuk implementasi jaringan *IoT* yang berjumlah banyak atau untuk jaringan yang tidak dapat terhubung langsung ke internet.

Protokol *routing* dinamis *RIP* secara umum menjadi protokol *routing* yang berperforma terburuk disebabkan oleh adanya proses algoritma *Distance-Vector Routing* yang hanya memperhitungan arah dan jarak untuk menentukan jalur ke jaringan tujuan. Protokol *RIP* juga dibatasi oleh *hop count* sebagai metrik perhitungan dan rute dengan *hop count* terkecil yang akan menjadi rute terbaik (*best path*). Kelemahan dari *hop count* yaitu hanya dapat mengenali *hop count* sampai angka ke-15. Lebih dari 15 jalur maka protokol *routing RIP* akan berhenti bekerja. Walaupun demikian, sebagaimana protokol *routing* dinamis lainnya, protokol *routing RIP* dapat mencari jalur jaringan secara mandiri tanpa perlu intervensi dari ahli jaringan dan dapat digunakan untuk implementasi jaringan *IoT* yang berjumlah banyak atau untuk jaringan yang tidak dapat terhubung langsung ke internet.

Hasil dari penelitian ini juga sesuai dengan beberapa penelitian terdahulu seperti penelitian dari (Lutfi et al., 2021) yang menunjukkan keunggulan dari protokol *routing* statis untuk perangkat *IoT* yang berjumlah sedikit dan menunjukkan performa yang mumpuni dengan angka *delay* yang rendah. Penelitian lain yang sesuai juga dari (Mudhoep et al., 2021), (Kabir et al., 2021), dan (Luis García-Navas et al., 2021) yang selalu menunjukkan dominasi performa protokol *routing OSPF* atas protokol *routing RIP* berdasarkan berbagai parameter analisis *QoS* untuk perbandingan antar sesama protokol *routing* dinamis.

## BAB 5

### KESIMPULAN DAN SARAN

#### 5.1 Kesimpulan

Berdasarkan proses penelitian yang sudah dilakukan sebelumnya, diperoleh hasil sebagai berikut :

1. Protokol *routing* statis memiliki nilai *throughput* terbesar, angka *delay* terendah, dan angka *jitter* terendah.
2. Protokol *routing* dinamis *OSPF* memiliki angka *packet loss* terendah.
3. Protokol *routing* dinamis *RIP* memiliki angka *latency* terendah.
4. Protokol *routing* statis sesuai untuk perangkat *IoT* yang dapat secara langsung terhubung ke penyedia layanan internet.
5. Protokol *routing* dinamis baik *RIP* atau *OSPF* sesuai untuk perangkat *IoT* yang banyak melewati jalur data atau tidak dapat secara langsung terhubung ke penyedia layanan internet.

#### 5.2 Saran

Penelitian yang sudah dilakukan ini tentu saja memiliki kekurangan, untuk itu peneliti memberikan saran yang dapat diberikan sehingga penelitian ini dapat lebih baik kedepannya, beberapa saran yang diberikan antara lain :

1. Menggunakan *software* simulasi lain dan *software* deteksi paket data lain yang memiliki kemampuan dan performa lebih baik.
2. Menambahkan kejadian-kejadian selama proses pengambilan data di jalur jaringan seperti terdapat momen matinya *router* tertentu atau pengaktifan *node* secara bertahap.
3. Meningkatkan jumlah *node* yang digunakan.
4. Menggunakan protokol *routing* lain yang berkemampuan lebih baik.
5. Menggunakan parameter *QoS* lainnya yang dapat dijadikan tolok ukur untuk mengukur kualitas protokol *routing* yang diujikan.

## DAFTAR PUSTAKA

- Aeni, D. N., Fauzi Ikhsan, A., & Susilawati, H. (2021). Analisis Trafik Jaringan Wifi dan Simulasi GNS3. *Jurnal FUSE-Teknik Elektro* /Vol. 1 /, 2.
- Agussalim, A., Kartika, D. S. Y., & Rahajoe, A. D. (2023). Implementasi Ad-Hoc Protocol Pada Tandem Multihop Wireless Network. *Jurnal Sistem Dan Teknologi Informasi (JustIN)*, 11(2), 377. <https://doi.org/10.26418/justin.v11i2.58051>
- Aulia, R., Liza, R., & Dafitri, H. (2024). Analisis Routing Loop dalam Open Shortest Path First (OSPF) Routing Menggunakan Teknik Spanning Tree di Jaringan Multi Area. *Hello World : Jurnal Ilmu Komputer*, 2(4). <https://doi.org/10.56211/helloworld.v2i3.419>
- Ayu, M. G. (2023, April 11). *BSSN: Jumlah Pengguna IoT Lebih Banyak Dibandingkan Smartphone*. <https://www.cloudcomputing.id/berita/bssn-menyatakan-pengguna-iot-lebih-banyak>
- Cañar, C. P. S., Yépez, J. J. T., & López, H. M. R. (2020). Performance of Reactive Routing Protocols DSR and AODV in Vehicular Ad-Hoc Networks Based on Quality of Service (Qos) Metrics. *International Journal of Engineering and Advanced Technology*, 9(4), 2033–2039. <https://doi.org/10.35940/ijeat.C6608.049420>
- El-Khozondar, H. J., Mtair, S. Y., Qoffa, K. O., Qasem, O. I., Munyarawi, A. H., Nassar, Y. F., Bayoumi, E. H. E., & Halim, A. A. E. B. A. El. (2024). A smart energy monitoring system using ESP32 microcontroller. *E-Prime - Advances in Electrical Engineering, Electronics and Energy*, 9. <https://doi.org/10.1016/j.prime.2024.100666>
- Farid, A. (2022, November 7). *Perbedaan Routing Statis dan Dinamis Secara Lengkap*. <https://www.exabytes.co.id/blog/perbedaan-routing-statis-dan-dinamis/>
- Fuada, S., Setyowati, E., Aulia, G. I., & Riani, D. W. (2023). NARATIVE REVIEW PEMANFAATAN INTERNET-OF-THINGS UNTUK APLIKASI SEED MONITORING AND MANAGEMENT SYSTEM PADA MEDIA TANAMAN HIDROPONIK DI INDONESIA. *INFOTECH Journal*, 9(1), 38–45. <https://doi.org/10.31949/infotech.v9i1.4439>
- Ghozali, T., Amantha, L., Sahat Manik, O., Ghozali, I. T., & Mahyastuty, V. W. (2022). Perancangan Jaringan Internet Menggunakan GNS3, Qemu, dan Virtual Box. *Jurnal Elektro Vol.15 No.1 April 2022*, 15(1).
- Hanif, A., & Amri, R. (2023). Implementasi Internet Of Things Pada Protokol MQTT Dan HTTP Dalam Sistem Pendekripsi Banjir. *JURNAL INOVTEK POLBENG - SERI INFORMATIKA*, 8(2).

- Hasbi, M., & Saputra, N. R. (2021). ANALISIS QUALITY OF SERVICE (QOS) JARINGAN INTERNET KANTOR PUSAT KING BUKOPIN DENGAN MENGGUNAKAN WIRESHARK. *Just IT: Jurnal Sistem Informasi, Teknologi Informasi Dan Komputer*, 12(1), 17–23. <https://jurnal.umj.ac.id/index.php/just-it/index>
- Hercog, D., Lerher, T., Trunčić, M., & Težak, O. (2023). Design and Implementation of ESP32-Based IoT Devices. *MDPI - Sensors*, 23(15). <https://doi.org/10.3390/s23156739>
- Kabir, M. H., Kabir, M. A., Islam, M. S., Mortuza, M. G., & Mohiuddin, M. (2021). Performance Analysis of Mesh Based Enterprise Network Using RIP, EIGRP and OSPF Routing Protocols †. *The 8th International Electronic Conference on Sensors and Applications*, 10(1). <https://doi.org/10.3390/ecsa-8-11285>
- Luis García-Navas, J., García, L., Romero, O., & Lorenz, P. (2021). Comparative Study of RIP, OSPF and EIGRP Protocols to Manage WSN-IoT Traffic vs IPTV Traffic Using Cisco Packet Tracer. *INNOV 2021 : The Tenth International Conference on Communications, Computation, Networks and Technologies*, 8–13.
- Lutfi, M., Hari Trisnawan, P., & Primananda, R. (2021). Implementasi Routing Statis menggunakan Media Komunikasi LoRa dan WebSocket untuk Pengiriman Data dari Sensor ke Cloud pada IoT. *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer*, 5(12), 5339–5348. <http://j-ptiik.ub.ac.id>
- Marfoq, O., Xu, C., Neglia, G., & Vidal, R. (2020). Throughput-Optimal Topology Design for Cross-Silo Federated Learning. *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*.
- Meilisa, L., Jayadi, A., Najib, M., & Satria, D. (2023). ANALISIS PERBANDINGAN METODE ROUTING DISTANCE VECTOR DAN LINK STATE PADA TOPOLOGI MESH DAN TOPOLOGI RING DALAM MENENTUKAN WAKTU KONVERGENSI TERCEPAT. *TELEFORTECH : Journal of Telematics and Information*, 4(1), 7–15.
- Mudhoep, D. I., Linawati, & Oka Saputra. (2021). Kombinasi Protokol Routing OSPF dan BGP dengan VRRP, HSRP, dan GLBP. *Jurnal Nasional Teknik Elektro Dan Teknologi Informasi*, 10(1), 1–10. <https://doi.org/10.22146/jnteti.v10i1.942>
- Novotný, A. (2023, September 1). *What is RTT (Round-Trip Time) and How to Reduce it?* <https://www.stormit.cloud/blog/what-is-round-trip-time-rtt-meaning-calculation/>
- Rifki Wardana, M., & Santoso, D. B. (2023). Analisis Throughput Distribusi Jaringan Nirkabel Pada Politeknik Bumi Akpelni. *Jurnal Riset Sistem Informasi Dan Teknik Informatika (JURASIK)*, 8(2), 558–567. <https://tunasbangsa.ac.id/ejurnal/index.php/jurasik>
- Saiqul Umam, M., Adi Wibowo, S., & Agus Pranoto, Y. (2023). IMPLEMENTASI PROTOKOL MQTT PADA APLIKASI SMART GARDEN BERBASIS IOT (INTERNET OF THINGS). In *Jurnal Mahasiswa Teknik Informatika* (Vol. 7, Issue 1).
- Saputra, E. P., Saryoko, A., Maulidah, M., Hidayati, N., & Dalis, S. (2023). Analisis Quality of Service (QoS) Performa Jaringan Internet Wireless LAN PT. Bhineka Swadaya Pertama. *Jurnal Sains Dan Manajemen*, 11(1).

- Segara, A. J. T., Wijayanto, A., Gustalika, M. A., & Ramadhani, A. D. (2022). Implementasi Mobile Ad-Hoc Network Pada Daerah Pasca Bencana Dengan Protokol DSR. *JURIKOM (Jurnal Riset Komputer)*, 9(4), 834. <https://doi.org/10.30865/jurikom.v9i4.4508>
- Setiawan, R. (2022). ANALISIS KINERJA ROUTING RIP DAN EIGRP PADA TOPOLOGI RING DAN MESH MENGGUNAKAN SIMULATOR GNS 3. *Jurnal Teknologi Pintar*, 2(5), 1–11.
- Sukmandhani, A. A. (2020, June 15). *QoS (Quality of Services)*. <https://online.binus.ac.id/computer-science/2020/06/15/qos-quality-of-services/>

