EMBEDDED SYSTEM NETWORK ANALYZER PADA JARINGAN LAN

SKRIPSI

KONSENTRASI KOMPUTASI BERBASIS JARINGAN

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :

DENNY HERIANTO

NIM. 0810680031

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN

UNIVERSITAS BRAWIJAYA

PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER

PROGRAM STUDI TEKNIK INFORMATIKA

MALANG

2013

LEMBAR PERSETUJUAN

EMBEDDED SYSTEM NETWORK ANALYZER PADA JARINGAN LAN

SKRIPSI

KONSENTRASI KOMPUTASI BERBASIS JARINGAN

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh:

DENNY HERIANTO

NIM. 0810680031

Skripsi ini telah disetujui oleh :

Dosen Pembimbing I	Dosen Pembimbing II
Raden Arief Setyawan, ST., MT.	**Sabriansyah R.A., ST., M.Eng.**
NIP. 19750819 199903 1 001	**NIK. 820809 06 1 1 0084**

LEMBAR PENGESAHAN

EMBEDDED SYSTEM NETWORK ANALYZER PADA JARINGAN LAN

SKRIPSI

KONSENTRASI KOMPUTASI BERBASIS JARINGAN

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :

DENNY HERIANTO

NIM. 0810680031

Skripsi ini telah diuji dan dinyatakan lulus pada
tanggal 17 Juli 2013

Penguji I	Penguji II
Ir. Heru Nurwasito, M.Kom.	**Adharul Muttaqin, ST., MT.**
NIP. 19650402 199002 1 001	**NIP. 197601212005011001**

Penguji III

Eko Sakti Pramukantoro, S.Kom., M.Kom.
NIK. 860805 06 1 1 0252

Mengetahui,
Ketua Program Studi Teknik Informatika

Drs. Marji, MT.
NIP. 19670801 199203 1 001

PERNYATAAN
ORISINALITAS SKRIPSI

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah SKRIPSI ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata didalam naskah SKRIPSI ini dapat dibuktikan terdapat unsur-unsur PLAGIASI, saya bersedia SKRIPSI ini digugurkan dan gelar akademik yang telah saya peroleh (SARJANA) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku. (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, Juli 2013

Mahasiswa,

Denny Herianto

081068003

KATA PENGANTAR

Dengan nama Allah SWT Yang Maha Pengasih dan Penyayang. Segala puji bagi Allah SWT karena atas rahmat dan hidayahNya-lah penulis dapat menyelesaikan Tugas Akhir yang berjudul "Embedded System Network Analyzer Pada Jaringan LAN ". Tugas Akhir ini disusun untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Komputer di Program Studi Teknik Informatika, Program Teknologi Informasi dan Ilmu Komputer Universitas Brawijaya Malang.

Penulis menyadari bahwa tugas akhir ini dapat terselesaikan dengan baik berkat bantuan, bimbingan dan petunjuk dari banyak pihak selama penulisan tugas akhir ini. Oleh karena itu, penulis ingin menyampaikan rasa hormat dan terima kasih penulis kepada :

1. Kedua Orang Tua penulis yang senantiasa tiada henti-hentinya memberikan dukungan moril, materi'il dan do'a demi terselesainya tugas akhir ini.
2. Pakde dan Bude penulis, Bapak Bambang Guritno dan Ibu Rien Samudayati yang telah memberikan nasehat, dukungan, arahan dan tempat tinggal kepada penulis selama kuliah di kota Malang.
3. Bapak Drs. Marji, MT. dan Bapak Issa Arwani S.Kom, MSc. selaku Ketua Program dan Sekretaris Program Studi Teknik Informatika Universitas Brawijaya.
4. Bapak Raden Arief Setyawan ST.,M.T. selaku dosen pembimbing I yang telah banyak memberikan bimbingan, ilmu dan saran dalam penyusunan tugas akhir ini.
5. Bapak Sabriansyah R. Akbar, ST., M.Eng. selaku dosen pembimbing II yang telah banyak memberikan bimbingan, ilmu dan saran dalam penyusunan tugas akhir ini.
6. Seluruh Bapak/Ibu Dosen beserta Staff Laboratorium, Administrasi dan Perpustakaan Program Teknologi Informasi Dan Ilmu Komputer.
7. Semua rekan penulis di Teknik Informatika angkatan 2008 (TIF 08).
8. Semua pihak yang tidak dapat penulis sebutkan satu per satu yang terlibat baik secara langsung maupun tidak langsung demi terselesaikannya tugas akhir ini.

Hanya doa yang bisa penulis berikan dan semoga Allah SWT memberikan pahala serta balasan kebaikan yang berlipat. Amin.

Penulis menyadari bahwa tugas akhir ini masih banyak kekurangan dan jauh dari sempurna, karena keterbatasan materi dan pengetahuan yang dimiliki penulis. Akhirnya semoga tuga akhir ini dapat bermanfaat bagi pembaca dan dapat berguna sebagai acuan bagi penelitian lebih lanjut.

Malang, Juni 2013

Penulis

ABSTRAK

Denny Herianto. 2013. Embedded System Network Analyzer Pada Jaringan LAN. Program Teknologi Informasi dan Ilmu Komputer, Universitas Brawijaya, Malang. Dosen Pembimbing : Raden Arief Setyawan, S.T.,M.T. dan Sabriansyah R.A., S.T., M.Eng.

Jaringan komputer memungkinkan kita menghubungkan sebuah komputer beserta perangkat jaringan lain untuk terhubung antara satu dengan lainnya. Dibutuhkan kondisi jaringan komputer yang baik agar mampu membuat komunikasi antar perangkat tersebut berjalan dengan optimal. Untuk mengukur kualitas jaringan komputer terdapat beberapa parameter yang dapat diukur antara lain *packet loss*, *troughput*, *delay* dan *jitter*. *Embedded system* atau sistem tertanam merupakan sistem komputer, yang dirancang khusus untuk melakukan tugas dan tujuan tertentu dengan menggunakan sumber daya secara minimum dan ditujukan untuk melakukan tugas secara *real time*. Penelitian yang dilakukan bertujuan untuk membuat sebuah *embbedded system network analyzer* yaitu sebuah alat yang dapat melakukan analisa sebuah kondisi jaringan berbasiskan *embedded system* menggunakan perangkat *beagleboard -XM*. Dalam melakukan perhitungan parameter tersebut, sistem dibuat menggunakan metode pemrograman *socket* yang akan melakukan pengiriman paket data dari *client* menuju *server*, hasil *reply* atau balasan yang dikirimkan oleh *server* akan dilakukan perhitungan berdasarkan parameter yang telah ditentukan.

kata kunci : embedded system, network analyzer, beagleboard –XM, pemrograman socket

ABSTRACT

Denny Herianto. 2013. ***Embedded System Network Analyzer in LAN Network.*** *Information Technology and Computer Science Program, Brawijaya University,* Malang. *Advisors*: Raden Arief Setyawan, S.T., M.T. and Sabriansyah R.A., S.T., M.Eng.

Computer networks allow us to connect the computers, servers and other network devices to connect to one another. It takes a good network conditions in order to be able to make communication between the device running with optimal. To measure the quality of computer networks there are some parameters that can be measured include packet loss, delay, jitter and troughput. Embedded systems is a computer system, designed specifically to perform tasks and specific objectives by using the minimum resources and are intended to perform tasks in real time. The research aims to create a network analyzer system embbedded is a tool that can analyze a network condition that is embedded system based using the beagleboard-XM. In performing calculations parameters, the system that we made using the concept programming a socket that will send the data from client toward server, results reply or requital delivered by server will be done calculations based on parameters that has been set.

Keywords *: embedded system, network analyzer, beagleboard-XM, socket programming*

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan kebutuhan pengguna komputer yang memanfaatkan jaringan komputer sebagai media komunikasi data, saat ini tumbuh dengan pesat. Untuk mendukung proses tersebut, dibutuhkan koneksi jaringan komputer yang cepat, stabil dan efisien yang memungkinkan proses komunikasi data dalam sebuah jaringan komputer berjalan dengan baik.

Terdapat beberapa faktor yang dapat mempengaruhi kondisi kualitas jaringan komputer antara lain jumlah pengguna, topologi jaringan dan perangkat keras maupun perangkat lunak yang digunakan, salah satu cara untuk mengetahui kondisi kualitas jaringan adalah dengan cara mengukur nilai parameter *troughtput*, *packet loss*, *delay* dan *jitter* [BAR-11:33-39]. Sebagai contoh dalam proses komunikasi data, komunikasi suara (*VoIP* atau *IP Telephony*) serta video *streaming* yang memanfaatkan jaringan komputer, dapat terganggu ketika paket data dialirkan di atas jaringan komputer dengan *packet loss* yang besar, *troughput* yang kecil serta dengan *delay* dan *jitter* yang berlebih [PET-07:40-43].

Pemanfaatan jaringan komputer sebagai media komunikasi data membutuhkan sebuah perangkat keras dan perangkat lunak untuk melakukan analisis terhadap kinerja jaringan tersebut, yang juga ditunjukan untuk menjaga stabilitas kinerja di dalam penggunaan jaringan komputer. Penilaian dari hasil analisis jaringan tersebut dapat didasarkan pada pengukuran beberapa parameter yang ditentukan sehingga nantinya informasi yang didapat bisa bermanfaat dalam perawatan maupun *troubleshooting* jaringan komputer.

Saat ini telah banyak perangkat keras dan perangkat lunak yang digunakan untuk melakukan analisis kondisi jaringan komputer, namun perangkat lunak tersebut membutuhkan sebuah komputer untuk menjalankan program tersebut, hal ini tentu membutuhkan sumber daya yang cukup besar dari segi penggunaan listrik dan kebutuhan spesifikasi *device* komputer yang digunakan untuk

menjalankan beberapa perangkat lunak yang telah terinstal seperti *service-service* perangkat lunak yang lebih dulu terinstal di komputer tersebut.

Untuk meminimalkan penggunaan sumber daya dan meningkatkan efisiensi pekerjaan, saat ini terdapat beberapa sistem yang mampu digunakan untuk melakukan dan menjalankan *service* maupun perangkat lunak yang sifatnya khusus atau spesifik dengan tujuan penggunaan dan kebutuhan pengguna. Contoh dari sistem yang mampu melakukan pekerjaan tersebut adalah dengan memanfaatkan sistem *embedded system.*

Embedded system adalah sistem yang berorientasi pada aplikasi spesifik dalam skala yang bervariasi, baik pada perangkat lunak maupun perangkat keras. *Embedded system* akan menjadi tren teknologi di masa depan, karena sistem yang kompleks dan handal dapat ditanam dalam sebuah perangkat yang berdaya rendah, berdimensi kecil dan berbiaya minim bila akan diproduksi masal [ARS-09:17-23].

Melihat perkembangan teknologi pada saat ini, *embedded system* akan terus berkembang pada masa depan dikarenakan sistem ini merupakan sebuah sistem yang unik, yang berbeda dari sebuah sistem komputer pada umumnya serta didesain untuk satu tugas dan memakai sumber daya rendah. Contoh penggunaan yang signifikan pada saat ini adalah penggunaan *embedded system* pada *web-server,* sensor suhu dan gerak serta peralatan jaringan seperti *switch* dan perangkat *wireless network.* [BEN-02:12-15].

Berdasarkan uraian diatas, *embedded system* sangat cocok digunakan untuk diimplementasikan pada sebuah perangkat lunak dan perangkat keras untuk melakukan sebuah tugas atau pekerjaan secara khusus guna meminimalkan penggunaan sumber daya. Pada penelitian *embedded system network analyzer* pada jaringan *LAN,* penulis akan membangun sebuah perangkat *network analyzer* berbasiskan *embedded system*. Diharapkan dapat menghasilkan sebuah perangkat *network analyzer* berbasiskan *embedded system* yang bersifat *portable* dan bersumber daya rendah yang mampu melakukan analisis jaringan dengan parameter yang dihitung meliputi *packet loss*, *troughtput*, *delay* dan *jitter* serta dapat berguna dalam melakukan perawatan maupun *troubleshooting* sebuah jaringan komputer.

1.2 Rumusan Masalah

Berdasarkan latar belakang diatas, penulis merumuskan beberapa masalah yaitu sebagai berikut :

1. Merancang dan membangun perangkat *network analyzer* untuk jaringan *Local Area Network (LAN)* berbasiskan *embedded system.*
2. Melakukan perhitungan kondisi sebuah jaringan dengan parameter *packet loss*, *troughtput*, *delay* dan *jitter* pada perangkat *network analyzer* berbasiskan *embedded system* yang dibuat menggunakan program dan perangkat yang telah dibuat.

1.3 Batasan Masalah

Agar diperoleh hasil pembahasan yang sesuai dengan apa yang diharapkan, maka perlu diberikan pembatasan masalah pada pengembangan sistem yang dibuat, yaitu :

- Penelitian *embedded system network analyzer* pada jaringan *LAN*, menganalisis sebuah kondisi jaringan komputer, dengan menghitung parameter *packet loss*, *troughtput*, *delay* dan *jitter* dan menampilkan hasil perhitungan pada sebuah *LCD character* berukuran 20 X 4.
- Analisis jaringan dilakukan dengan menggunakan protokol *TCP/IP*.
- Menggunakan konsep *socket programming* sebagai metode komunikasi untuk pengiriman paket data (*client-server)*.
- Pengujian sistem dilakukan pada satu waktu dan satu segmen jaringan yang sama di wilayah gedung PTIIK UB.
- Sistem operasi pada perangkat di sisi *client* dan *server* menggunakan sistem operasi *linux*.

1.4 Tujuan

Berdasarkan rumusan masalah yang ditulis diatas, tujuan dari penelitian *embedded system network analyzer* pada jaringan *LAN* adalah untuk melakukan analisis kondisi sebuah jaringan dengan melakukan perhitungan parameter *packet loss*, *troughtput*, *delay* dan *jitter* dengan perangkat network analyzer berbasiskan *embedded system* pada sebuah jaringan *Local Area Network (LAN).*

1.5 Manfaat

Berdasarkan rumusan masalah yang ditulis diatas, diharapkan penelitian *embedded system network analyzer* pada jaringan *LAN* dapat bermanfaat untuk berbagai pihak. Manfaat dari penelitian adalah sebagai berikut:

- Memberikan solusi dan alternatif dalam melakukan perhitungan kondisi sebuah jaringan komputer dengan parameter *packet loss, troughtput, delay* dan *jitter* menggunakan perangkat *portable* dan bersumber daya rendah berbasiskan *embedded system*.
- Menjadi bahan referensi penelitian yang berkaitan dengan kualitas layanan jaringan menggunakan berbasiskan *embedded system*.

1.6 Sistematika Pembahasan

Sistematika pembahasan dari penyusunan tugas akhir *embedded system network analyzer* pada jaringan *LAN* direncanakan sebagai berikut :

BAB I PENDAHULUAN

Bab I Pendahuluan terdiri dari latar belakang masalah, batasan masalah, rumusan masalah, tujuan penelitian, manfaat penelitian, dan sistematika pembahasan penelitian. Bab I Pendahuluan menjadi dasar dari keseluruhan pelaksanaan penelitian *embedded system network analyzer* pada jaringan *LAN*.

BAB II KAJIAN TEORI

Bab II membahas teori-teori yang berkaitan dan menunjang dalam penyelesaian penelitian *embedded system network analyzer* pada jaringan *LAN*. Kajian teori yang diambil berasal dari jurnal, buku, dan sumber referensi lainnya yang berhubungan dengan topik yang akan diteliti.

BAB III METODOLOGI PENELITIAN DAN PERANCANGAN

Bab III menjelaskan metode atau langkah-langkah dalam perancangan, implementasi, pengujian dan analisis yang dilakukan. Perancangan membahas analisis kebutuhan sistem seperti, sistem operasi, perangkat lunak dan perangkat keras yang digunakan dalam implementasi sistem serta akan dijelaskan topologi jaringan yang akan digunakan dalam implementasi sistem.

BAB IV IMPLEMENTASI

Pada bab IV implementasi, akan dijelaskan pembuatan sistem *embedded system network analyzer* pada jaringan *LAN* (*Local Area Network*) secara terperinci dengan menampilkan gambar-gambar dari hasil implementasi yang telah dilakukan.

BAB V PENGUJIAN DAN ANALISIS

Bab V berisi pengujian dan analisis terhadap sistem *embedded system network analyzer* yang dibangun. Pengujian tersebut dilakukan secara bertahap sesuai dengan alur yang ditentukan pada Bab III Metodologi Penelitian dan Perancangan.

BAB VI PENUTUP

Bab VI berisi kesimpulan dari pelaksanaan penelitian *embedded system network analyzer* pada jaringan *LAN* yang dibuat setelah dilakukan pengujian dan analisis. Untuk meningkatkan hasil dari kinerja sistem yang telah dibuat dalam penelitian *embedded system network analyzer* pada jaringan *LAN*, maka diberikan saran-saran untuk perbaikan dan penyempurnaan sistem yang telah dibangun.

BAB II
KAJIAN PUSTAKA DAN DASAR TEORI

2.1 Penelitian Terkait

Penelitian yang membahas *network analyzer* berbasiskan *embedded system*, salah satunya adalah penelitian yang dilakukan oleh [RAH-09] yang berjudul "*Web-based Portable Network Traffic Monitoring System based on Embedded Linux and SBC* (*Serial Board Computer*)" tahun 2009, yang dilakukan pada penelitian tersebut adalah pemantauan lalu lintas jaringan menggunakan sebuah program *traffic packet analyzer* yang dibuat oleh peneliti, dibuat berbasiskan program *tcpdump* yang digabungkan dengan sistem *embedded system* dan dibangun menggunakan perangkat *miniboard SBC TS–5400*. Hasil dari pemantauan dan analisis jaringan yang telah dilakukan akan disimpan pada *web server* dan diakses melalui *browser*.

Penelitian lain yang berkaitan dengan *network analyzer* berbasiskan *embedded system* adalah penelitian yang berjudul "*Portable Network Analyzer"* yang dilakukan oleh [WAR-11], tahun 2011. Pada penelitian tersebut, peneliti membangun sebuah alat *network analyzer portable* menggunakan perangkat *miniboard chestnut43-expansionboard*. Perangkat *network analyzer portable* yang dibuat ditujukan untuk melakukan *monitoring* lalu lintas paket data pada sebuah sistem jaringan dengan menggunakan program *wireshark*, peneliti juga melakukan pengujian terhadap performa *resource* yang terdapat pada perangkat *portable network analyzer* yang dibuat.

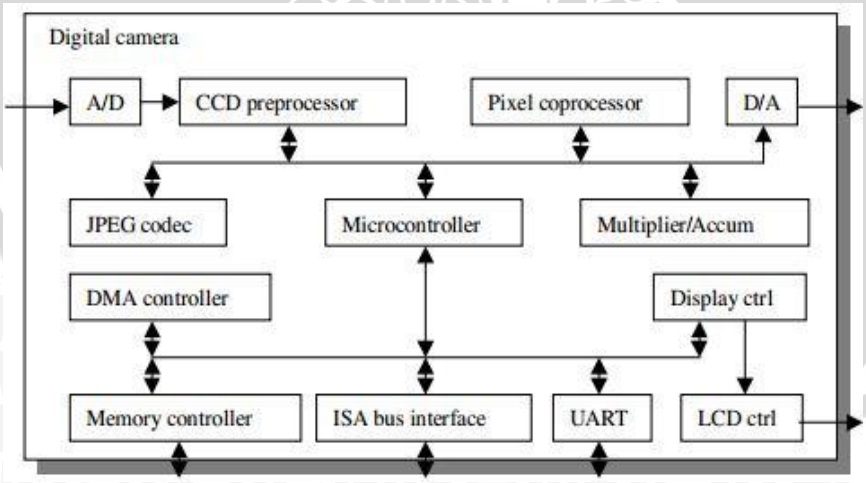
Berdasarkan penelitian yang penulis bahas diatas, pemanfaatan teknologi *embedded system* sudah tepat bila diterapkan pada perangkat *miniboard* dalam menciptakan sebuah *network analyzer* yang *portable*. Pada kedua penelitian tersebut, perangkat-perangkat yang digunakan dalam membangun *network analyzer* berbasiskan *embedded system* adalah perangkat *miniboard SBC (serial board computer) TS-5400* [RAH-09] dan perangkat *miniboard chestnut43-expansionboard*. [WAR-11]. Hal ini membuktikan bahwa penerapan teknologi *embedded system* dapat

berjalan saat diimplementasikan pada perangkat *miniboard* sehingga dapat menggunakan sumber daya yang rendah serta *resource* yang terbatas.

2.2 Embedded System

Embedded Sistem merupakan sistem yang secara khusus dirancang untuk melakukan tugas yang sangat spesifik dan berulang - ulang, serta tidak membutuhkan sumber daya yang sangat tinggi dan hal ini akan menghemat biaya produksi, *Embedded Sistem* diperkenalkan pertama kali pada tahun 1961 yaitu pada sistem yang bernama *Apollo Guidance Computer*, yang dibuat oleh *Charles Stark Draper* di laboratorium *instrument*, MIT (Massachusetts Institute of Technology) [NIM-13].

Sistem ini merupakan sistem komputer yang arsitektur sistemnya berbeda dari jenis komputer *konvensional* seperti komputer *desktop* (*PC*), perbedaannya adalah e*mbedded system* merupakan sistem yang ditanam pada sebuah perangkat yang ditujukan untuk satu pekerjaan tertentu dan melakukan pekerjaan tersebut secara berulang-ulang, perangkat yang sering kali ditanam *embedded system* ini adalah m*icrocontroller, microprocessor* maupun *digital signal processor chip* [NOE-05:6-7]. Salah satu contoh perangkat yang menggunakan *embedded system* adalah *digital camera* yang gambaran arsitekturnya terdapat pada gambar 2.1 :



Gambar 2.1 : Contoh Arsitektur Embedded System

Sumber : [VAH-99:6]

2.2.1 Karateristik Embedded System

Embedded system dirancang untuk tujuan khusus yaitu melakukan satu pekerjaan komputasi atau sistem tersebut dirancang khusus untuk aplikasi yang spesifik sehingga penggunaannya dapat berjalan optimal. Karakteristik mengenai *embedded system*, menurut [LEG-08] adalah komponen-komponen untuk *embedded sistem* dipilih secara khusus agar terbentuk hasil implementasi sistem yang handal tetapi dengan biaya yang minimum dan berjalan secara *real-time* sistem yang prosesnya berjalan secara langsung tanpa waktu tunda serta harus terkondisi tetap stabil namun membutuhkan energi yang sedikit.

Penjelasan tentang karakteristik e*mbedded system* yang membedakan sistem tersebut dengan sistem komputer yang lain, juga dijelaskan oleh [VAH-99:4-8]. Beberapa karakteristik yang dijelaskan pada buku tersebut antara lain bersifat *single function* yaitu hanya menjalankan satu program secara berulang kali, *tightly constrained yaitu* sistem dengan desain pembatasan pada biaya, ukuran dan kekuatan serta r*eactive and real-time* yaitu dapat bekerja dengan kondisi yang berubah-rubah dan *real-time* seperti pada penggunaan sistem sensor kecepatan.

Beberapa peralatan-peralatan dengan karakteristik *embedded system* pada saat ini dapat ditemui pada [ARS-09:17-23] :

- Peralatan elektronik Industri dimana sistem tertanam menjadi komponen penting untuk tugas-tugas produksi dan otomatisasi pabrikasi produk yang kompleks. Contoh aplikasinya terdapat pada *programmable logic controller* (*PLC*) dan sensor cerdas.
- Peralatan *elektromedis* (*medical electronic devices*) seperti peralatan *Ultrasonography* (*USG*), detektor denyut jantung dan alat pengukur gula darah elektroniss erta alat pencitraan medis (*medical imaging*).
- Peralatan jaringan komputer (*computer networks)* seperti pada *routers*, *switch* dan *wireless acces point.*

2.3 Parameter Kualitas Jaringan

Performansi dalam sebuah sistem jaringan komputer mengacu pada tingkat kecepatan dan kehandalan dari sebuah pengiriman paket data di dalam suatu komunikasi berbasis jaringan komputer yang dapat diukur dari beberapa parameter besaran teknis. Pengukuran performansi dan kondisi dari kondisi kualitas jaringan dapat diukur oleh beberapa parameter besaran teknis, [TAN-03:301-311] dalam bukunya menjelaskan sebagai berikut :

1. *Packet loss* adalah suatu parameter yang menggambarkan suatu kondisi yang menunjukkan jumlah paket data yang hilang pada proses pengiriman paket data dalam jaringan, penyebab terjadinya *packet loss* antara lain adalah karena adanya *collision* (tabrakan antar paket data) dan *congestion* (penumpukan paket data) pada jaringan. Rumus perhitungan :

$$Packet\ loss = \frac{\sum Packets\ Lost}{\sum Packets\ Sent} \text{ X } 100\%$$

Sumber : [MEH-12]

Keterangan :

– *Packets Lost* = Paket data yang hilang

– *Packets Sent* = Paket data yang dikirimkan

[ETS-02] menentukan standarisasi kategori penilaian terhadap *paket loss* yang dijelaskan pada tabel 2.1 :

Tabel 2.1 Standarisasi Penilaian *Packet loss*

NO	Kategori	Presentase
1	Sangat Bagus	0%
2	Bagus	3%
3	Sedang	15%
4	Jelek	25%

2. *Throughput* merupakan besaran yang menunjukan seberapa banyak paket data yang berhasil diterima dalam proses pengiriman paket data dalam jaringan. Faktor-faktor yang menentukan nilai *troughput* adalah besaran paket

data yang ditransfer, topologi jaringan, banyaknya pengguna dan spesifikasi perangkat. Rumus perhitungan :

$$Troughput = \frac{\sum Packets\ Delivered}{\sum Packets\ Arrival\ Time - Packets\ Start\ Time}$$

Sumber : [MEH-12]

Keterangan :

- *Packets Delivered* = Paket data yang terkirim
- *Packets Arrival Time* = Waktu kedatangan paket data
- *Packets Start Time* = Waktu awal dikirimkannya paket data

3. *Delay* adalah waktu (*millisecond*) yang dibutuhkan paket data dalam proses pengiriman paket data dari asal menuju tujuan. *Delay* dapat dipengaruhi oleh jarak, media fisik (hardware), topologi jaringan dan besaran paket data yang dikirimkan. Rumus perhitungan :

$$Delay = \sum Packets\ Arrival\ Time - Packet\ Start\ Time$$

Sumber : [MEH-12]

Keterangan :

- *Packets Arrival Time* = Waktu kedatangan paket data.
- *Packets Start Time* = Waktu awal dikirimkannya paket data.

[ETS-02] menentukan standarisasi kategori penilaian terhadap *delay* yang dijelaskan pada tabel 2.2 :

Tabel 2.2 Standarisasi Penilaian *Delay*

No	Kategori	Besar Delay
1	Sangat Bagus	< 150 *ms*
2	Bagus	151 – 300 *ms*
3	Sedang	300- 450 *ms*
4	Jelek	> 450 *ms*

4. *Jitter* adalah variasi waktu proses pengirima paket data (*delay*) yang diakibatkan oleh variasi panjang antrian, waktu yang dibutuhkan dalam pengolahan paket data dan media fisik (*hardware*) yang digunakan. Nilai

Jitter digunakan sebagai patokan parameter kualitas kondisi jaringan untuk melihat kondisi jaringan dalam rentang waktu berbeda. Rumus perhitungan :

$$Jitter = (R_i - S_i) - (R_{i+1} - S_{i+1})$$

Sumber : [ANS - 08]

Keterangan :

- R = Received Time (Waktu paket data saat diterima atau datang)
- S = Sent Start Time (Waktu awal pengiriman paket data)

[ETS-02] menentukan standarisasi kategori penilaian terhadap *paket loss* yang dijelaskan pada tabel 2.3 :

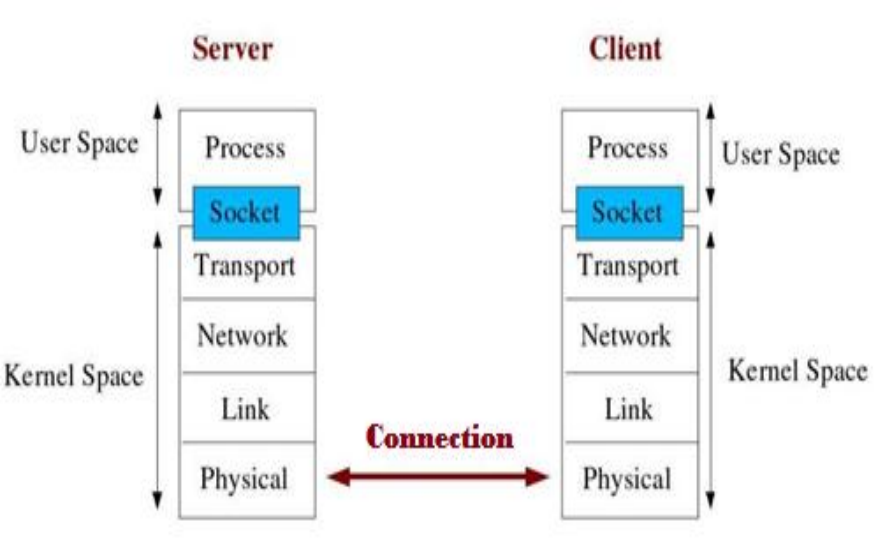
Tabel 2.3 Standarisasi Penilaian *Jitter* [ETS-02]

No	Kategori	Besar *Jitter*
1	Sangat Bagus	0 *ms*
2	Bagus	1 - 75 *ms*
3	Sedang	76 - 125 *ms*
4	Jelek	126 - 225 *ms*

2.4 Pemrograman Socket

Socket adalah mekanisme komunikasi yang memungkinkan terjadinya pertukaran data antar aplikasi atau proses aplikasi baik dalam satu mesin maupun antar mesin yang berbeda. *Socket* memungkinkan suatu aplikasi untuk masuk kedalam jaringan dan berkomunikasi dengan aplikasi lain yang juga masuk kedalam jaringan yang sama. Informasi yang ditulis kedalam *socket* di sebuah aplikasi dalam sebuah mesin dapat dibaca oleh aplikasi lain pada mesin yang berbeda begitu pula sebaliknya [KUR-11].

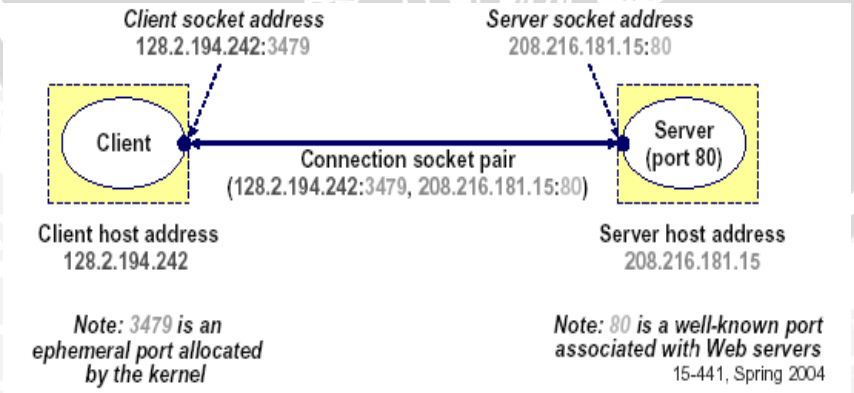
Pemrograman *socket* adalah sebuah aplikasi pemrograman berbasis *socket* yang berada diantara proses sebuah aplikasi dan *transport layer*. Proses aplikasi tersebut dapat mengirim dan menerima pesan dari atau menuju proses aplikasi lainnya lewat sebuah *socket* dan memungkinkan adanya komunikasi antara *client* dan *server* [CHE-12]. Berikut ilustrasi konsep pemrograman *socket* yang dijelaskan pada gambar 2.2 :



Gambar 2.2 Ilustrasi Konsep Pemrograman *Socket*

Sumber : [CHE-12]

Dalam komunikasi jaringan komputer yang memanfaatkan konsep *socket*, dibutuhkan beberapa tahapan dalam prakteknya agar *client* dapat berkomunikasi dengan *server*. Agar *client* dan *server* dapat saling berkomunikasi, *client* harus tahu berapa nomor *IP address server* begitu juga nomor *port* yang dituju. Dalam hal ini *IP address* berperan sebagai alamat dalam jaringan dan nomor *port* menunjukkan *service* yang dijalankan. Contoh program aplikasi di *client* yang meminta *service* di *server* adalah *http, ftp, telnet, ssh* dll. Berikut gambaran konsep komunikasi jaringan berbasis *socket*, yang dijelaskan pada gambar 2.3 [UZI-12] :



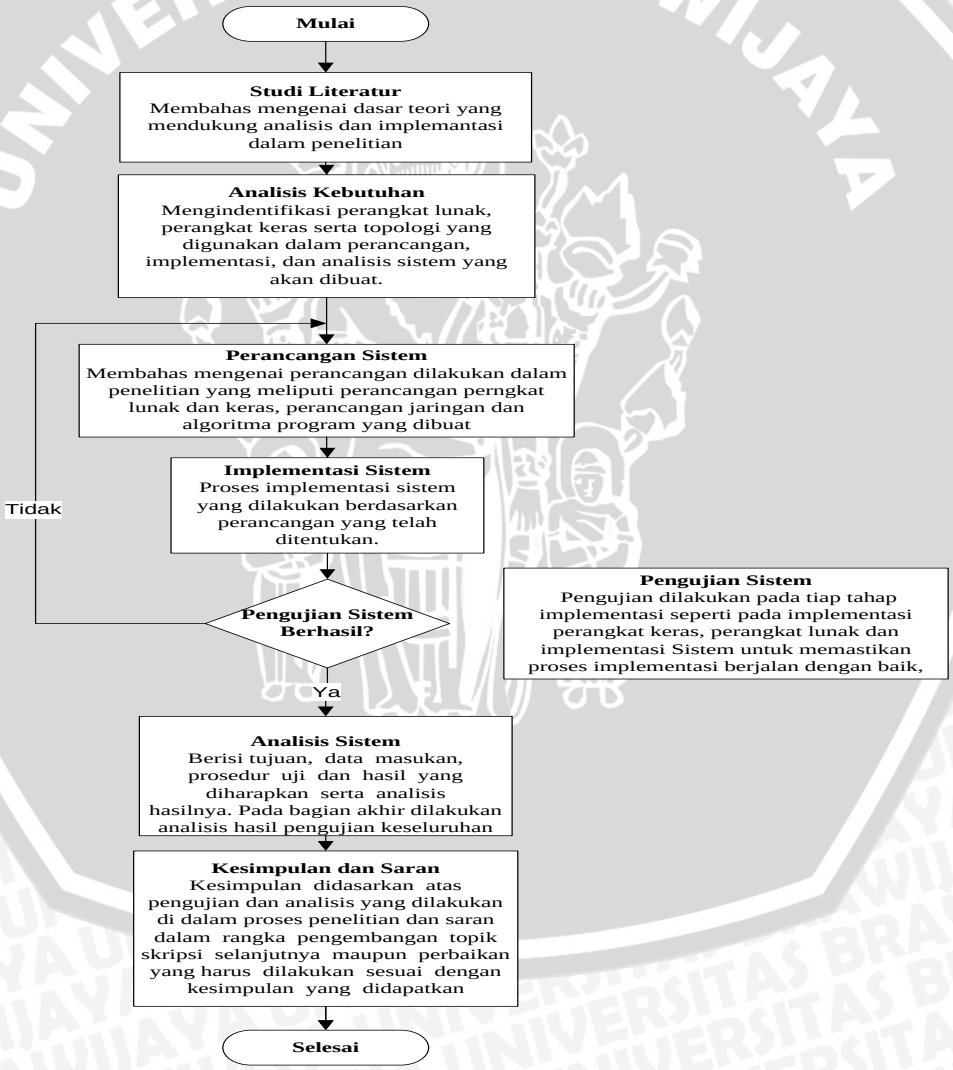
Gambar 2.3 Konsep Komunikasi Jaringan Berbasis *Socket*

Sumber : [UZI-12]

BAB III
METODELOGI PENELITIAN DAN PERANCANGAN

3.1 Metodelogi Penelitian

Pada bab ini dijelaskan langkah-langkah yang akan dilakukan dalam perancangan, implementasi, analisis dan pengujian dari sistem yang akan dibuat. Kesimpulan dan saran disertakan sebagai catatan atas sistem dan kemungkinan arah pengembangan selanjutnya. Diagram alir dari pelaksanaan penelitian secara keseluruhan dapat dilihat pada gambar 3.1 :



Gambar 3.1 Diagram Alir Kesuluruhan Pelaksanaan Penelitian

3.1.1 Studi Literatur

Studi literatur menjelaskan seluruh dasar teori yang mendukung dalam penelitian *embedded system network analyzer* pada jaringan *LAN*, adapun yang dijadikan dalam bahan studi literatur adalah dasar-dasar teori untuk dapat merancang, membangun beserta cara pengujian sistem meliputi :

a) Embedded System
- Karakteristik *Embedded System*

b) Parameter Penilaian Kualitas Jaringan
- *Packet loss*
- *Troughput*
- *Delay*
- *Jitter*

c) Pemrograman *Socket*
- Konsep Pemrograman *Socket*

3.2 Perancangan

Perancangan sistem dilakukan setelah semua kebutuhan sistem didapatkan melalui tahap analisis kebutuhan. Perancangan sistem diperlukan agar peneliti dapat dengan mudah dan tepat dalam melakukan proses implementasi pada tahap selanjutnya. Setelah mendapatkan semua kebutuhan perangkat lunak dan perangkat keras pada tahap analisis kebutuhan, sistem *embedded system network analyzer* ini kemudian dirancang sesuai dengan kebutuhan pelaksanaan penelitian

Pada sistem *embedded system network analyzer*, proses komunikasi paket data antara *client* dan *server* menggunakan metode *socket*. Pada penelitian *embedded system network analyzer* pada jaringan *LAN* terdapat dua perangkat *beagleboard -XM*, sebuah perangkat *beagleboard -XM* berperan sebagai *client*, yang memiliki tugas sebagai pengirim paket data kepada *server*. Sedangkan sebuah perangkat *beagleboard -XM* yang lain berperan sebagai *server*, yang bertugas sebagai pihak yang melakukan *reply* paket data yang dilakukan oleh *client*. Dari proses komunikasi data tersebut parameter *packet loss*, *troughput*, *delay* dan *jitter* yang akan dijadikan ukuran kualitas jaringan pada penelitian *embedded system network analyzer* pada jaringan *LAN*.

Perancangan sistem *embedded system network analyzer* terdiri dari beberapa tahap, yaitu perancangan perangkat keras dan perancangan perangkat lunak, disertai dengan perancangan jaringan sistem dan algoritma program yang dibuat.

3.2.1 Analisis Kebutuhan

Analisis kebutuhan bertujuan untuk mendapatkan semua kebutuhan pada penilitian yang berjudul "*Embedded System Network Analyzer Pada Jaringan LAN*". Pada analisis kebutuhan tindakan yang dilakukan adalah mengindentifikasi perangkat lunak, perangkat keras serta topologi yang digunakan dalam perancangan, implementasi dan analisis sistem. Dengan demikian diharapkan dapat mempermudah dalam mendesain sistem dan hasil analisis terhadap sistem yang dibuat dapat lebih akurat.

3.2.2 Kebutuhan Perangkat Lunak

Kebutuhan perangkat lunak yang digunakan pada penelitian *embedded system network analyzer* pada jaringan *LAN* adalah :

- Operating System : Angstrom Linux
- Pemrograman : C, C++
- Port Scanner : Nmap

3.2.3 Kebutuhan Perangkat Keras

Dalam membangun e*mbedded sytem network analyzer* membutuhkan perangkat berupa *beagleboard -XM* yang akan dijadikan perangkat *portable network analyzer*. Dengan spesifikasi sebagai berikut :

- Processor : Texas Instrument Cortex A 8 1 GHZ
- Data Storage : 4 Gb MDDR SDRAM
- NIC : Ethernet 10/100 Mbps
- Display : LCD Character 20 x 4 - Green Display 5V
- Input : Keyboard

3.2.4 Perancangan Perangkat lunak dan Perangkat Keras

Penelitian *embedded system network analyzer* pada jaringan *LAN* memerlukan beberapa infrastruktur penunjang berupa beberapa perangkat lunak dan perangkat keras. Seluruh perangkat lunak yang akan digunakan telah ditentukan melalui tahap analisis kebutuhan. Pada tahap ini, seluruh perangkat lunak tersebut dirancang agar dapat membentuk sebuah sistem *embedded system network analyzer* sesuai dengan yang dikehendaki. Seluruh perangkat lunak tersebut digunakan sebagai bagian dari proses perancangan, pengujian dan analisis sistem jaringan dari penelitian *embedded system network analyzer* pada jaringan *LAN*.

Untuk perangkat keras yang diperlukan yaitu dua buah perangkat *beagleboard -XM*. Sebuah *beagleboard -XM* digunakan sebagai *client* yang juga berfungsi sebagai pengirim paket data. Sedangkan perangkat *beagleboard -XM* yang lain digunakan sebagai *server* yang berfungsi untuk me-*reply* paket data yang dikirimkan oleh *client*.

- **Perancangan Perangkat Lunak**
  - **Sistem Operasi** : Sistem operasi yang digunakan dalam pembuatan sistem *embedded system network analyzer*, pada sisi *client* dan sisi *server* adalah *Linux Angstrom*. Alasan penulis menggunakan sistem operasi *Linux Angstrom* adalah dikarenakan sistem operasi *Linux Angstrom* telah teruji kehandalannya dan *support* terhadap perangkat *beagleboard -XM*, sistem *embedded system* dan pemrograman *socket.*
  - **Bahasa C, C++ :** Pemrograman bahasa C, C++ digunakan dalam seluruh pembuatan berkas program.
  - **Compiler :** Dalam proses pembuatan program, dibutuhkan program untuk melakukan proses penulisan dan *editing source code* program. Sebagai text editor *source code* program dan *compiler* untuk menjalankan program yang telah dibuat, digunakan program *codebloker* dan *G++*.

- **Perancangan Perangkat Keras**

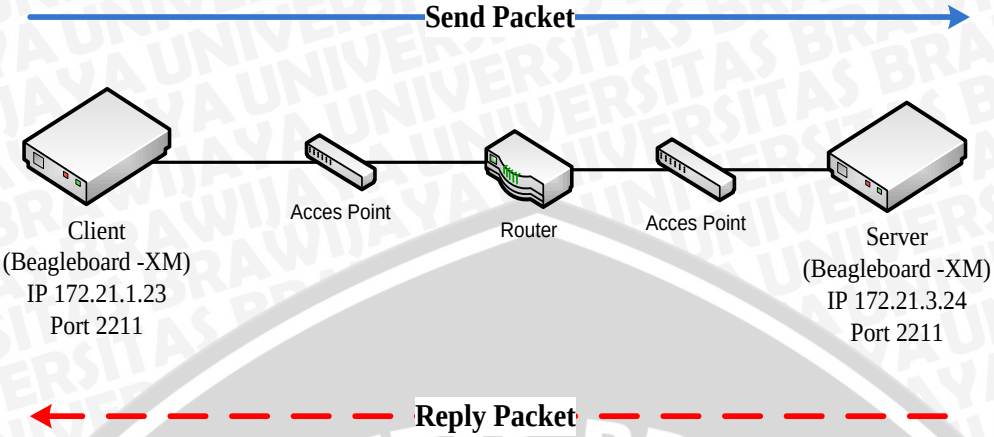
Perancangan perangkat keras diperlukan untuk membangun sistem komunikasi diantara seluruh komponen yang terdapat pada sistem ini. Perangkat *beagleboard -XM* sebagai *client* dan sebuah *beagleboard -XM* sebagai *server*. Berikut alokasi sumber daya yang digunakan pada penelitian *embedded system network analyzer* pada jaringan *LAN* yang dapat dilihat pada tabel 3.1 :

Tabel 3.1 Alokasi Sumber Daya

Server	Client
Beagleboard-XM • OS : Linux Angstrom • NIC : Ethernet 10/100 M*bps*	Beagleboard-XM • OS : Linux Angstrom • NIC : Ethernet 10/100 M*bps* • Display : 20 x 4 Character LCD - Green Display 5V • Input : Keyboard

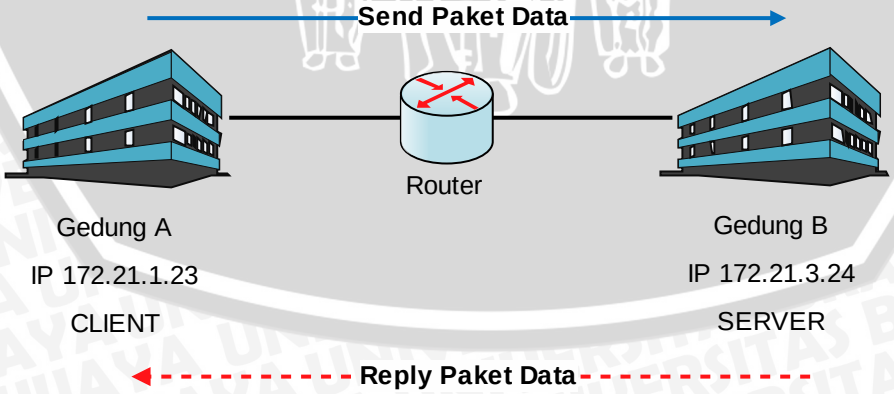
3.2.5 Perancangan Topologi Jaringan

Perancangan topologi jaringan mengarah pada bagaimana perangkat *embedded system network analyzer* bekerja di dalam sistem jaringan. Pada tahap ini menggambarkan bagaimana perangkat *beagleboard –XM* pada sisi c*lient* melakukan pengiriman paket data ke perangkat *beagleboard –XM* pada sisi *server*. Dalam implementasinya sisi *server* tidak harus menggunakan perangkat *beagleboard* –XM juga seperti pada sisi *client*, dikarenakan sisi *server* hanya bertugas melakukan *reply* paket data yang dikirimkan oleh sisi *client*. Untuk sisi *server* dapat digunakan juga *PC* maupun *notebook* yang berperan sebagai sisi *server*. Berikut perancangan topologi jaringan penelitian *embedded system network analyzer* pada *LAN* yang terdapat pada gambar 3.2 :



Gambar 3.2 Perancangan Topologi Jaringan

Penjelasan mengenai perancangan topologi jaringan yang terdapat pada gambar 3.2 adalah pada sisi *client* akan melakukan pengiriman paket data kepada sisi *server*, dengan *IP address client* adalah 172.21.1.23 dan *IP address server* 172.21.3.24. Dikarenakan metode pengiriman paket data pada penelitian *embedded system network analyzer* pada jaringan *LAN* menggunakan konsep *socket*, maka selain *client* membutuhkan *IP address* dibutuhkan juga *port* yang digunakan sebagai *port service* tujuan paket data yang akan dikirimkan. Pada perancangan topologi jaringan ini nomer *port* yang digunakan adalah *port* 2211. Setelah paket data diterima oleh sisi *server*, paket data tersebut akan dikirimkan kembali kepada sisi *client* dan hasil dari pengembalian paket data tersebut akan dilakukan perhitungan parameter *packet loss*, *troughput*, *delay* dan *jitter*.



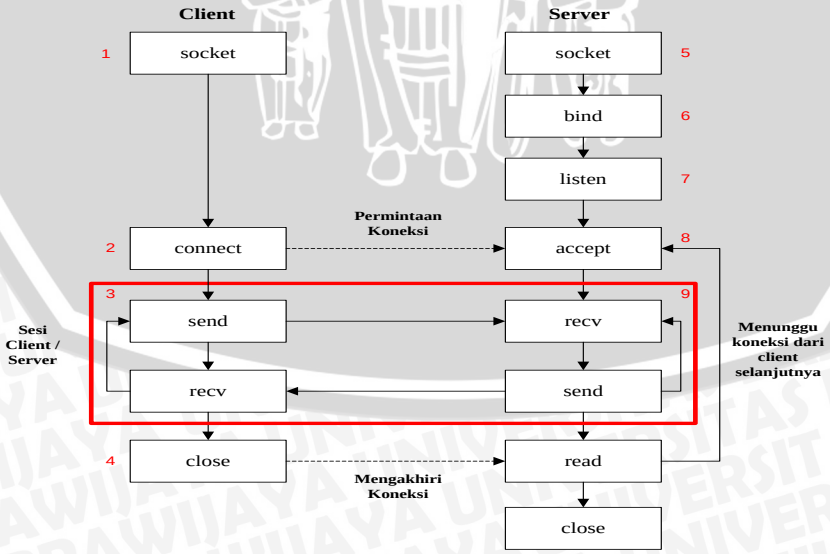
Gambar 3.3 Percancangan Jaringan Pada LAN

Berdasarkan gambar 3.3 dalam proses implementasi pada jaringan LAN, perangkat *network analyzer* berupa perangkat *beagleboard –XM* akan

ditempatkan pada sebuah sistem jaringan yang akan dilakukan perhitungan parameter *packet loss*, *troughput*, *delay* dan *jitter*-nya. Proses implementasi perhitungan parameter pada penelitian *embedded system network analyzer* pada jaringan *LAN* adalah perangkat *network analyzer* yang berupa perangkat *beagleboard –XM* akan ditempatkan pada gedung A dengan *IP address* 172.21.1.23 dan berperan sebagai sisi *client* yang akan melakukan pengiriman paket data sebesar 128 *byte*, 256 *byte*, 512 *byte*, 1024 *byte* dan 1500 *byte* kepada sisi *server* yang terdapat pada gedung B dengan *IP address* 172.21.3.24. Hasil pengembalian paket data yang dilakukan oleh sisi *server* kepada sisi *client* akan dimasukan kedalam rumus perhitungan yang terdapat pada program *network analyzer* di perangkat *beagleboard –XM* untuk dicari nilai dari parameter *packet loss*, *troughput*, *delay* dan *jitter*.

3.3 Perancangan Algoritma dan Pembuatan Program

Untuk dapat melakukan proses perhitungan parameter *packet loss*, *troughput*, *delay* dan *jitter*. Dibutuhkan sebuah program yang ditujukan untuk melakukan perhitungan ke lima parameter tersebut. Dimana sistem kerja perhitungan parameter tersebut berdasarkan proses pengiriman paket data yang dikirimkan dengan metode *socket* dari *client* ke *server*. Berikut gambar ilustrasi perancangan program beserta penjelasannya yang terdapat pada gambar 3.4 :



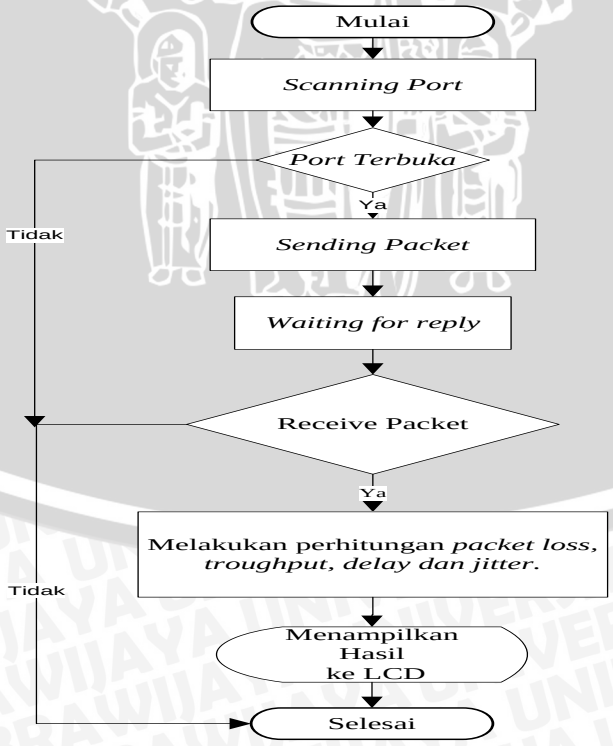
Gambar 3.4 : Ilustrasi Perancangan Pemrograman Socket

Sumber : [DON – 01:111-122]

Berikut penjelasan ilustrasi perancangan program berbasis pemrograman *socket* yang dijelaskan pada gambar 3.4 :

- Proses yang terjadi di sisi *client* :
  1. Membuat *socket* dan melakukan pengalamatan ke *server*.
  2. Menghubungi *server*.
  3. Melakukan komunikasi (mengirim dan menerima data).
  4. Mengakhiri koneksi.
- Langkah – langkah dasar di sisi *server* :
  5. Membuat *socket*.
  6. Mengikatkan *socket* kepada sebuah alamat *network*.
  7. Menyiapkan *socket* untuk menerima koneksi yang masuk.
  8. Menerima koneksi yang masuk ke *server*.
  9. Melakukan komunikasi (mengirim dan menerima data).
  10. Mengakhiri koneksi.

Berikut diagram alir algoritma berdasarkan konsep pemrograman *socket* pada penelitian *embedded system network analyzer* pada jaringan *LAN* yang dapat dilihat pada gambar 3.5 :



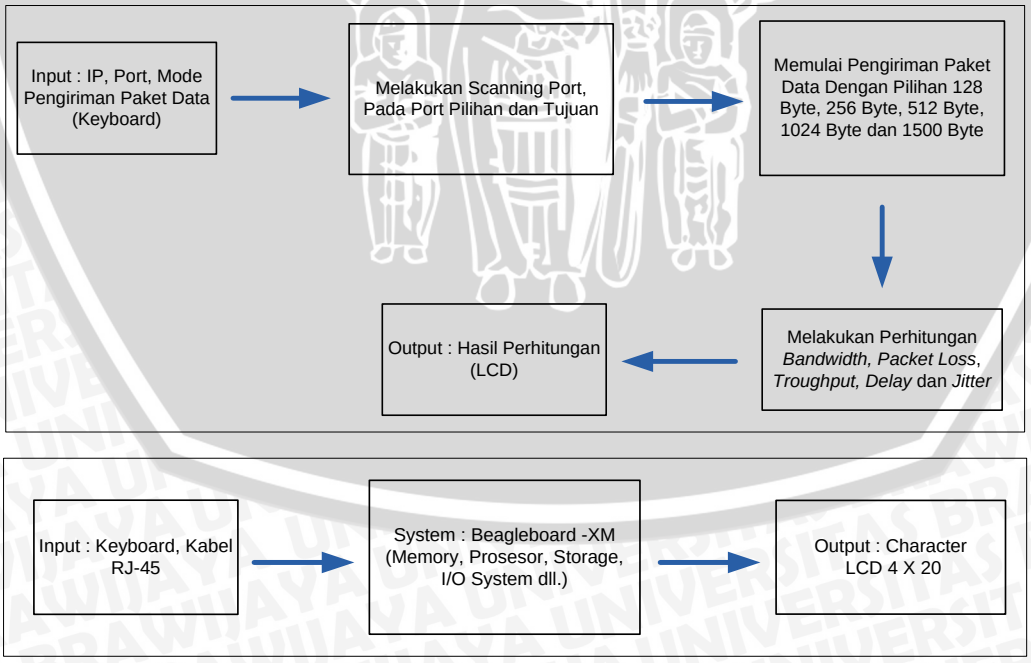
Gambar 3.5 Diagram Alir Algoritma Program

Penjelasan algoritma program :

Proses awal perihtungan parameter *packet loss*, *troughput*, *delay* dan *jitter* dilakukan setelah proses komunikasi antara *client* dan *server* terhubung yaitu antara perangkat *beagleboard –XM* di sisi *client* dan sisi *server*. Kemudian melakukan *scanning port* pada *port* 21 *FTP*, *port* 22 *SSH*, *port* 23 *Telnet*, *port* 80 *HTTP* dan *port* tujuan, setelah itu proses pengiriman paket data dilakukan oleh sisi *client* kepada sisi *server*, paket data yang dikirimkan berupa paket data 128 *byte*, 256 *byte*, 512 *byte*, 1024 *byte*, 1500 *byte* dengan batasan waktu satu detik dan sepuluh kali percobaan di setiap pengiriman paket data .

Setelah mendapat hasil *reply* paket data dari sisi *server* kemudian *reply* paket data tersebut dihitung nilai parameter *packet loss*, *troughput*, *delay* dan *jitter* berdasarkan rumus yang telah ditentukan, kemudian hasil dari perhitungan parameter tersebut akan ditampilkan pada layar *LCD* yang telah terpasang.

Untuk memudahkan proses implementasi keseluruhan perancangan sistem dalam melakukan proses pengkodean kedalam sebuah bahasa pemrograman, akan dibuat terlebih dahulu desain umum perancangan sistem. Berikut desain umum perancangan sistem yang ditampilkan pada gambar 3.6 :



Gambar 3.6 Desain Umum Perancangan Sistem

Berikut penjelasan desain umum perancangan sistem yang terdapat pada gambar 3.6 :

1. Dibutuhkan input berupa *IP Address* tujuan, *Port server* tujuan dan mode pilihan pengiriman paket data menggunakan *keyboard*.
2. Setelah *input* dimasukan, program akan melakukan *scanning port* pada port pilihan dan tujuan.
3. Pengiriman paket data akan dikirimkan oleh sisi *client* kepada sisi *server* sesuai pilihan pengiriman paket data yang dimasukan.
4. Server melakukan *reply* paket data dan proses perhitungan parameter mulai dilakukan.
5. Hasil *output* berupa perhitungan parameter ditampilkan pada layar *LCD* yang terdapat pada sisi *client*.

3.4 Perhitungan Parameter

Pada sub-bab perhitungan parameter, akan ditampilkan proses contoh perhitungan manual untuk mendapat nilai kualitas jaringan yang nantinya proses perhitungan tersebut akan diimplementasikan pada bahasa pemrograman dengan parameter yang dihitung adalah parameter *packet loss*, *troughput*, *delay* dan *jitter*.

1. *Packet loss*, dalam satuan % (Persentase)

   Rumus perhitungan *packet loss* :

$$Packet\ loss = \frac{\sum Packets\ Lost}{\sum Packets\ Sent} \text{ X } 100\% \quad \text{.....(3-1)}$$

Contoh Perhitungan :

- Variabel *Packet loss* =
  - *Packet Lost* = Paket data yang dikirim – paket data yang diterima
    = 128 *Byte* – 128 *Byte* = 0 *Byte*
  - Paket data yang dikirim = 128 *Byte*

Berdasarkan persamaan 3-1, hasil perhitungan *packet loss* adalah :

Packet loss = 0 *Byte* : 128 *Byte* x 100 % = 0 %

2. Troughtput, dalam satuan *Bps* (*Byte* per second)

   Rumus perhitungan *troughput* :

$$Troughput = \frac{\sum Packets\ Delivered}{\sum Packets\ Arrival\ Time - Packets\ Start\ Time} \quad \text{..... (3-2)}$$

Contoh Perhitungan :

- Variabel Troughput =
  - Paket data yang terkirim = 128
  - Total waktu pengiriman = 1,0990

  Berdasarkan persamaan 3-2, hasil perhitungan *troughput* adalah :

  Troughput = 128 *Byte* : 1,0990 x 1000 = 116469,5177434031 *Byte* Per Second

3. Delay, dalam satuan *ms* (milisecond)

   Rumus perhitungan *delay* :

$$Delay = \sum Packets\ Arrival\ Time - Packet\ Start\ Time \quad \text{.....(3-3)}$$

Contoh Perhitungan :

- Variabel Delay =
  - Waktu awal = 0
  - Waktu kedatangan = 1,0990 *ms*

  Berdasarkan persamaan 3-3, hasil perhitungan *delay* adalah :

  Delay = 1,0990 – 0 = 1,0990 *ms*

4. *Jitter*, dalam satuan *ms* (*milisecond*)

   Rumus perhitungan *jitter* :

$$Jitter = (R_i - S_i) - (R_{i+1} - S_{i+1}) \quad \text{.....(3-4)}$$

Contoh Perhitungan :

- Variabel *Jitter* =
  - *Delay* ke -1 = 1,1900
  - *Delay* ke -2 = 1,0070

  Berdasarkan persamaan 3-4, hasil perhitungan *jitter* adalah :

  Jitter = delay ke-2 – delay ke-1 = 1,0990 – 1,0070 = 0,920 *ms*

Untuk nilai *jitter*, hasil akhir perhitungan harus merupakan nilai mutlak, dikarenakan *jitter* merupakan nilai selisih antara nilai delay ke i dan delay i + i [MEH-12].

Contoh perhitungan diatas merupakan contoh perhitungan untuk paket data 128 *Byte*, untuk perhitungan 256 *byte*, 512 *byte*, 1024 *byte* dan 1500 *byte* pada proses perhitungan parameter *packet loss*, *troughput*, *delay* dan *jitter* proses perhitungannya sama hanya diganti varibel besaran paket datanya.

3.5 Perancangan Antarmuka

Tampilan antarmuka aplikasi merupakan gambaran struktur tampilan dari program yang dijalankan. Perancangan antarmuka sangat diperlukan untuk mempermudah *user* dalam menggunakan aplikasi *network analyzer* berbasis *embedded system* yang dibuat dan melihat hasil dari proses eksekusi dan perhitungan dari program tersebut. Tampilan antarmuka yang dibuat akan terdapat pada sisi *server* dan sisi *client*.

3.5.1 Tampilan Antarmuka Sisi Server

Tampilan antarmuka sisi *server*, menampilkan tampilan sisi *server* untuk menjalankan program dan tampilan saat telah terhubung dengan sisi client dan menunjukan *server* dalam proses melakukan *reply* paket data yang dikirimkan oleh *client*. Untuk tampilan awal sisi server bertujuan untuk menjalankan program *network analyzer* pada sisi *server* dengan format ./nama program dan *port* tujuan. Tampilan antarmuka untuk program *network analyzer* pada sisi *server* dapat dilihat pada gambar 3.7 dan 3.8 :

```
./server.out <Port>
```

Gambar 3.7 Tampilan Awal Sisi Server

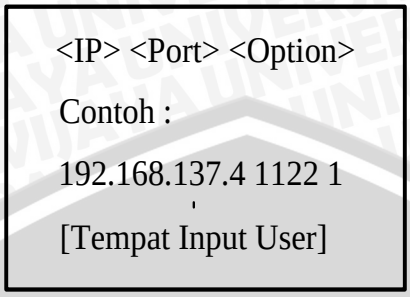
```
Start Listenning...
Client Connected <IP Client>
```

Gambar 3.8 Tampilan Hasil Sisi Server

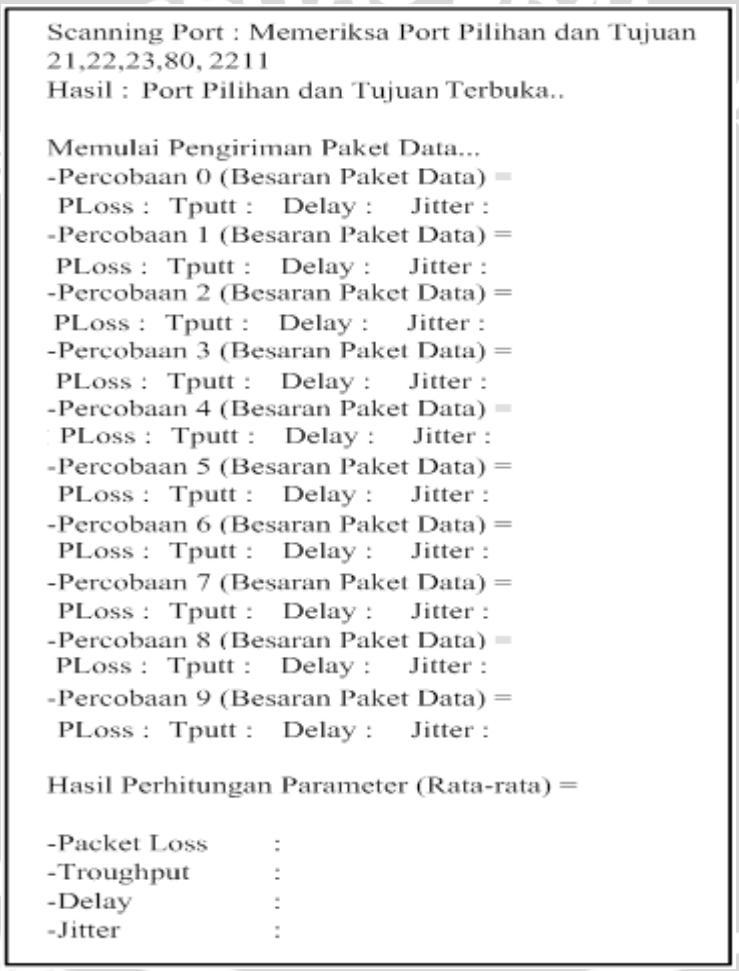
3.5.2 Tampilan Antarmuka Sisi Client

Tampilan antarmuka sisi *client*, menampilkan tampilan awal dari program *network analyzer* yang yang dibuat pada sisi *client*, tampilan ini berfungsi untuk memasukan data dengan mode *IP Address* tujuan, *Port* tujuan dan *Option* yang merupakan pilihan dari besaran paket data yang akan dikirimkan. Tampilan

antarmuka untuk program network analyzer pada sisi client dapat dilihat pada gambar 3.9 dan 3.10



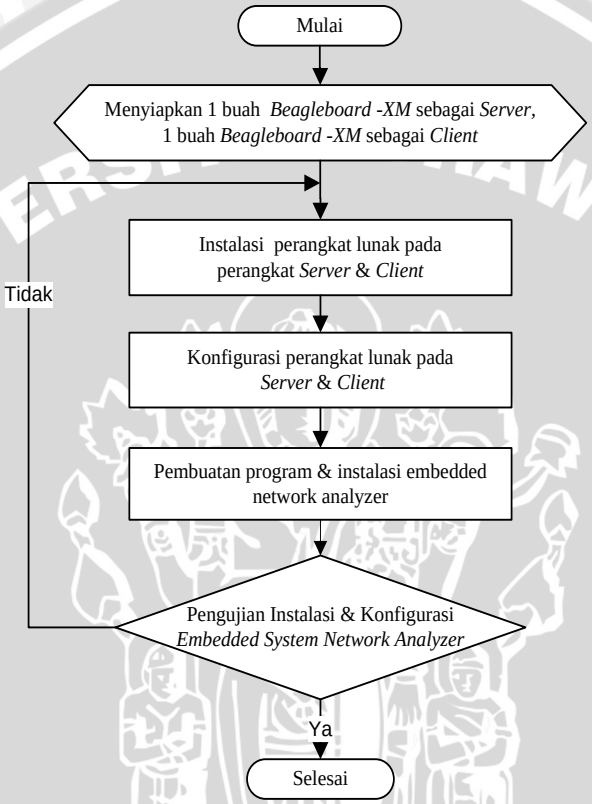
Gambar 3.9 Tampilan Awal Program Sisi Client



Gambar 3.10 Tampilan Hasil Sisi Client

3.6 Implementasi

Implementasi dilakukan berdasarkan perancangan yang telah ditentukan. Proses implementasi dilakukan bertahap agar peneliti dapat memperbaiki kesalahan sistem dengan mudah. Tahapan implementasi pada penelitian *embedded system network analyzer* pada jaringan *LAN* dapat dilihat pada gambar 3.11 berikut:

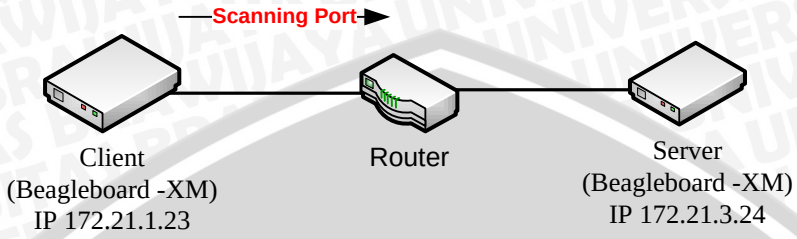


Gambar 3.11 Diagram Alir Implementasi Sistem

3.7 Strategi Pengujian dan Analisis

Pengujian adalah tahap untuk memastikan implementasi yang telah dilakukan. Pengujian dilakukan pada tiap tahap implementasi seperti pada implementasi perangkat keras, perangkat lunak dan implementasi sistem. Tujuan dari pengujian perangkat keras, perangkat lunak dan pengujian sistem adalah untuk memastikan perangkat keras, perangkat lunak dan sistem yang dibangun pada penelitian *embedded system network analyzer* pada jaringan *LAN* dapat bekerja dengan baik. Pada tahapan pengujian, terdapat beberapa tahapan-tahapan pengujian yang dilakukan tahap pertama adalah melakukan setting *port* pada sisi

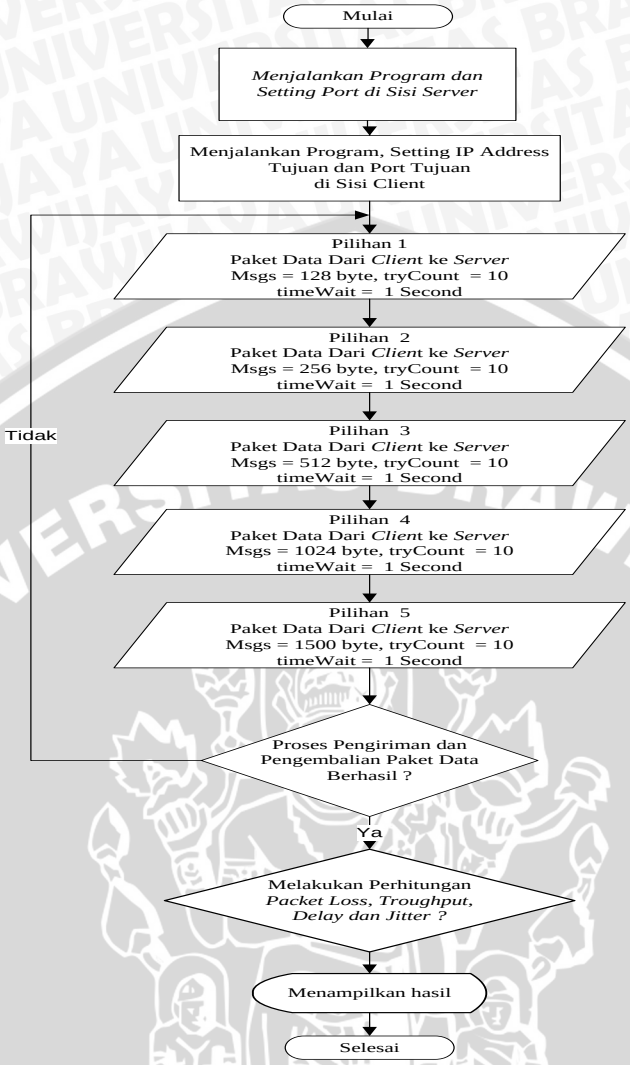
server, selanjutnya pada sisi *client* adalah melakukan pengujian terhadap beberapa *port* pilihan dan tujuan. Ilustrasi dari tahapan *scanning port* dapat dilihat pada gambar 3.12.



Gambar 3.12 Ilustrasi Scanning Port Pada Jaringan

Penjelasan dari gambar 3.12 yang merupakan ilustrasi tahapan *scanning port* adalah bertujuan untuk melakukan pengecekan dan pencarian informasi mengenai kondisi jaringan yang digunakan sebelum melakukan analisis kondisi jaringan. Hal ini ditujukan karena pada sebuah sistem jaringan komputer memiliki kondisi yang berbeda-beda dari segi konfigurasi keamanan jaringan yang dilakukan oleh seorang *administrator* jaringan. Tahapan *scanning port* bertujuan untuk mengetahui nomer *port* berapakah yang dapat digunakan oleh sisi *server* sebagai nomer *port* tujuan untuk sisi *client*. Pada penelitian *embedded system network analyzer* pada jaringan *LAN* penulis menentukan *port* 21 *FTP*, *port* 22 *SSH*, *port* 23 *Telnet*, *port* 80 *HTTP* sebagai contoh *well-known port* pada jaringan dan *port* 2211 sebagai *service por*t yang disediakan oleh sisi *server*.

Tahap ketiga, setelah mengetahui kondisi jaringan dengan melakukan *scanning port* dan mengetahui nomer *port* berapakah yang dapat dipakai sebagai *port* tujuan, barulah dilakukan pengiriman paket data dan melakukan perhitungan parameter kualitas kondisi jaringan. Tahap terakhir adalah menampilkan hasil perhitungan parameter pada layar *LCD* di perangkat *beagleboard –XM* pada sisi *client*. Proses pengujian dan analisis juga dilakukan untuk mengetahui kinerja sistem jaringan, hasil analisis sistem jaringan yang berupa parameter *packet loss*, *troughput*, *delay* dan *jitter* tersebut yang kemudian digunakan untuk menarik kesimpulan dan saran. Tampilan perancangan sistem dapat dilihat pada gambar 3.13 berikut :



Gambar 3.13 Diagram Alir Skenario Pengujian Sistem

3.8 Pengambilan Kesimpulan

Pengambilan kesimpulan dilakukan setelah perancangan, implementasi, pengujian dan analisis dilakukan. Kesimpulan disusun berdasarkan pada hasil pengujian dan analisis terhadap sistem yang dibangun, isi pada kesimpulan diharapkan dapat menjadi acuan untuk pengembangan dan penyempurnaan sistem. Pada akhir penulisan terdapat saran yang bertujuan untuk memperbaiki kesalahan dan melakukan penyempurnaan terhadap sistem yang telah dibuat.

BAB IV
IMPLEMENTASI

Pada bab implementasi, akan dibahas langkah-langkah yang dilakukan dalam pembuatan sistem *embedded system network analyzer*. Sesuai dengan rancangan pada Bab III, sistem *embedded system network analyzer* dibangun menggunakan perangkat *beagleboard –XM*. Pada tahap implementasi, langkah-langkah yang akan dilakukan penulis antara lain adalah instalasi, konfigurasi dan pembuatan berkas program. Dalam hal ini, langkah-langkah tersebut mengacu pada perangkat keras dan perangkat lunak yang digunakan.

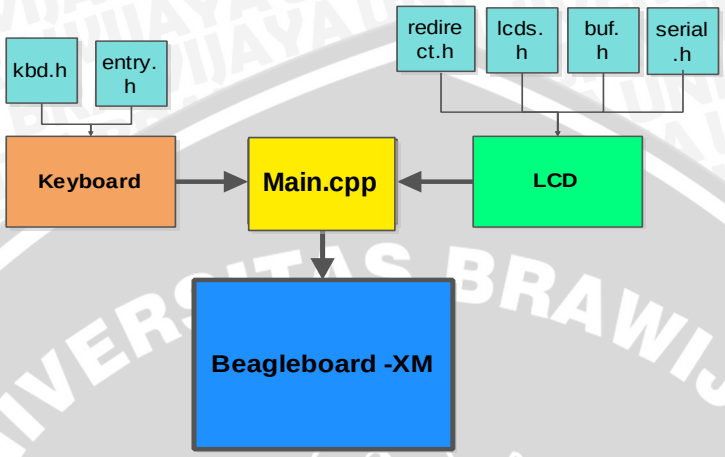
4.1 Implementasi Sistem

Langkah awal yang perlu dilakukan untuk membangun sistem *embedded network analyzer* yaitu menghubungkan seluruh komponen sistem yang ada. Komponen-komponen yang akan dihubungkan adalah perangkat *beagleboard –XM* untuk sisi *client* dan sebuah *beagleboard –XM* untuk sisi *server*, kedua perangkat tersebut dihubungkan dengan menggunakan kabel *UTP*. Langkah selanjutnya adalah proses konfigurasi jaringan dan instalasi perangkat lunak pada perangkat *Beagleboard –XM* pada sisi *client* dan sisi *server*.

Proses instalasi perangkat lunak yang perlu dilakukan adalah instalasi sistem operasi *Linux Angstrom* 2.6.32 dan dilanjutkan dengan instalasi *packet library g++* dan *gcc*, serta program *Nmap*. Setelah proses instalasi perangkat lunak selesai kemudian dilanjutkan pembuatan berkas program untuk sisi *client* dan sisi *server* yang berfungsi untuk melakukan perhitungan parameter *packet loss, troughput, delay* dan *jitter*. Hasil perhitungan parameter nantinya akan ditampilkan pada *LCD* yang terdapat pada perangkat *beagleboard –XM*.

Dalam proses implementasi *network analyzer* berbasiskan *embedded system* dengan menggunakan perangkat *beagleboard –XM* sebagai perangkat yang akan ditanam (*embed*) oleh sebuah program. Terdapat beberapa berkas-berkas program dan perangkat keras yang dijadikan sebagai konfigurasi dan perangkat input serta output yang dibutuhkan untuk membangun sistem *network analyzer*. Agar mendapatkan hasil yang baik, dalam proses implementasi *embedded system*

network analyzer penulis sertakan diagram impelementasi *embedded system* yang akan dibuat beserta penjelasannya. Berikut diagram implementasi *embedded system* yang terdapat pada gambar 4.1 :



Gambar 4.1 Diagram Implementasi Embedded System

Penjelasan gambar 4.1 adalah dalam proses implementasi *embedded system* yang menggunakan perangkat *beagleboard –XM* sebagai perangkat yang ditanam program *network analyzer*, terdapat berkas program Main.cpp yang merupakan program utama yang akan melakukan analisis jaringan dengan melakukan pengiriman paket data kepada sisi server dan melakukan perhitungan parameter *packet loss*, *troughput*, *delay* dan *jitter* dari hasil reply paket data yang diterima. Berkas program ini memiliki *include* program kbd.h dan entry.h, fungsi dari dua berkas program tersebut adalah sebagai program yang melakukan konfigurasi untuk nilai input atau masukan yang berasal dari *Keyboard*, berkas program Main.cpp juga memiliki *include* berkas program redirect.h, lcd.h, buf.h dan serial.h, keempat fungsi dari berkas program tersebut memiliki fungsi untuk hasil nilai output atau keluaran yang akan menampilkan hasil perhitungan parameter *packet loss*, *troughput*, *delay* dan *jitter* pada layar *LCD*.

4.1.1 Konfigurasi Jaringan

Pada proses konfigurasi jaringan yang dilakukan adalah mengubah *berkas* jaringan *Ethernet client* dan *server* pada sistem operasi *Linux Angstrom*. Konfigurasi *berkas* ini diperlukan untuk menentukan alamat *Internet Protocol*

pada kedua perangkat *beagleboard –XM* yang digunakan tersetting pada konfigurasi jaringan *DHCP*. Tujuannya agar mendapatkan alamat *internet protocol* secara otomatis pada saat proses implementasi dan pengujian. Proses konfigurasi jaringan pada sisi *client* dilakukan dengan mengakses *berkas* interface dalam direktori /etc/network/ pada perangkat *beagleboard –XM* yang berfungsi sebagai *client* :

```
  GNU nano 2.2.5         File: /etc/network/interfaces

 auto eth0
 iface eth0 inet dhcp
#iface eth1 inet dhcp

# Ethernet/RNDIS gadget (g_ether) or LAN9514
 auto usb0
# iface usb0 inet dhcp

# Bluetooth networking
#iface bnep0 inet dhcp
```

Gambar 4.2 Konfigurasi Berkas Jaringan Ethernet Sisi Client

Setelah melakukan konfigurasi pada sisi *client*, Proses selanjutnya adalah melakukan konfigurasi pada sisi *server*, proses konfigurasi tersebut hampir sama dengan dengan proses konfigurasi *berkas* pada sisi *client*, Proses konfigurasi jaringan pada sisi *client* dilakukan dengan mengakses berkas *interface* dalam direktori /etc/network/ pada perangkat *beagleboard –XM* yang berfungsi sebagai *server*.

```
  GNU nano 2.2.5         File: /etc/network/interfaces

 auto eth0
 iface eth0 inet dhcp
#iface eth1 inet dhcp

# Ethernet/RNDIS gadget (g_ether) or LAN9514
 auto usb0
# iface usb0 inet dhcp

# Bluetooth networking
#iface bnep0 inet dhcp
```

Gambar 4.3 Konfigurasi Berkas Jaringan Ethernet Sisi Server

4.2 Pembuatan Berkas Program

Setelah melakukan proses konfigurasi jaringan pada sisi *client* dan server selesai dilakukan, proses selanjutnya adalah pembuatan berkas program. Pada penelitian *embedded system network analyzer* pada jaringan *LAN* proses komunikasi data dibuat berbasiskan komunikasi *socket*, sehingga dibutuhkan dua buah program utama yang berfungsi sebagai *client* dan *server*. Selain itu dibuat juga berkas program untuk menampilkan hasil perhitungan parameter *packet loss, troughput, delay* dan *jitter*. Berikut seluruh berkas program yang dibuat penulis beserta daftar berkas yang dibuat beserta penjelasan singkat dari masing-masing berkas yang terdapat pada tabel 4.1 :

Tabel 4.1 Penjelasan Berkas Program

No	Nama Berkas Program	Fungsi
1	server.cpp	Berkas Program server.cpp akan berjalan pada sisi server, program ini akan melakukan *reply* paket data yang dilakukan oleh sisi client. Sebelumnya berkas program server.cpp akan di-*compile* terlebih dahulu dengan perintah g++ server.cpp –o server.out
2	main.cpp	Berkas program main.cpp akan berjalan pada sisi *client*, program ini berfungsi sebagai pengirim paket data ke *server*. Sebelumnya *berkas* program client.cpp di-*compile* terlebih dahulu dengan perintah g++ main.cpp –o client.out
3	kbd.h	Berkas program ini digunakan untuk konfigurasi fungsi *keyboard* yang berupa karakter, arah (atas, bawah, kanan, kiri), *spasi*, *backspace*, *delete* dan *enter* di LCD

4	entry.h	Berkas program ini digunakan untuk menyediakan fungsi inputan *IP Address*, nomer *Port* dan mode pilihan besaran paket data serta arah (atas, bawah, kanan, kiri), edit, delete dan enter pada LCD
5	lcds.h	Berkas program ini digunakan untuk konfigurasi driver dan tampilan inputan pada LCD
6	buf.h	Berkas program ini digunakan untuk menyediakan dan menampung buffer karakter, yang akan ditampilkan pada LCD
7	redirect.h	Berkas program ini berfungsi sebagai pengubah perintah tampilan (perintah printf) pada program yang terdapat pada perangkat beagleboard –XM menjadi bentuk karakter pada LCD
8	serial.h	Berkas program ini digunakan sebagai media komunikasi dari beagleboard –XM menuju LCD yang menggunakan output serial

Dalam pembuatan berkas program server.cpp dan main.cpp (*server* dan *client*), penulis sesuaikan dengan konsep pemrograman *socket* yang terdapat pada buku "TCP/IP Socket C – Practical Guide for Programmers" karya Michael j. Donahoo dan Kenneth L. Calvert [DON – 01:111-122], Berikut penulis menjabarkan perintah-perintah (fungsi) pada berkas program tersebut :

1. Berkas program server.cpp

a) Fungsi Socket :

Fungsi *socket* pada *server* digunakan untuk membuat koneksi *socket*, yang kemudian digunakan sebagai titik awal dan akhir komunikasi untuk mengirim dan menerima data menggunakan protokol tertentu.

```
If ((Server_Sockets = socket(PF_INET, SOCK_STREAM,
IPPROTO_TCP)) < 0)
```

Dari kode diatas PF_NET menunjukan bahwa *socket* ditunjukan untuk koneksi internet, SOCK_STREAM dan IPPROTO_TCP menunjukan bahwa program ini menggunakan stream socket TCP yang berarti *connection oriented*.

b) Fungsi Bind :

Digunakan untuk memberikan alamat atau nomor *port* untuk *socket*. *Server* menggunakan fungsi *bind* untuk mengikat koneksi *client* yang masuk pada *server* yang sebelumnya telah menunggu sinyal masuk dari koneksi yang akan terhubung dengan memberikan koneksi *socket*.

```
if (bind(Server_Sockets, (struct sockaddr *)
&Server_IP, sizeof(Server_IP)) < 0)
```

Dari kode diatas bind(Server_Sockets, (struct sockaddr *) menunujukan alamat *port* yang akan ditentukan oleh *server*, &Server_IP,sizeof(Server_IP) menunjukan alamat IP yang ditentukan oleh *server*.

c) Fungsi Listen :

Menyiapkan *socket* untuk menerima koneksi yang masuk, fungsi *listen* menunjukkan bahwa *socket* yang diberikan siap untuk menerima koneksi masuk.

```
if (listen(Server_Sockets, MAX_CONNECTION) < 0)
```

Dari kode diatas *listen* (Server_Sockets, Max Connection) < 0 menunjukan *server* siap menerima koneksi dari *client*, dengan kondisi *client* memasukan *IP Address* dan *port* sesuai dengan *IP Address* dan *port* yang ditentukan oleh *server*.

d) Fungsi Accept :

Fungsi *accept* bertugas untuk menunjukan bahwa *server* siap untuk melakukan koneksi antar *socket* (*client* dan *server*) dan siap untuk menerima koneksi yang masuk dari *client*, dan *server* memunculkan keterangan *IP Address* yang telah terhubung dengan *server*.

```
if ((Client_Sockets = accept(Server_Sockets, (struct
sockaddr *) &Client_IP,  &Addres_Length)) < 0)
```

Dari kode diatas terdapat baris Client_Socket = accept (Server_Socket) (struct socket *) &Client_IP, &Address_Length)) < 0) baris ini menunjukan server menerima koneksi dari *client* berupa *port* dan *IP Adrress* beserta besaran paket data yang dikirimkan oleh *client*. Dan program *server* akan menampilkan baris *Client connected* bila koneksi antara *client* dan *server* telah tersambung.

e) Fungsi Recv:

Fungsi *recv* digunakan *server* untuk menerima data. Fungsi *recv* digunakan *server* untuk menerima data sesuai dengan konsep pemrograman *socket*, pada sisi *server* fungsi *recv* digunakan untuk mengirimkan *response* pada *client* hasil sinyal *request* yang *client* kirimkan sebelumnya.

```
if ((Recv_Msg_Length = recv(Client_Socket, Buffer_Msgs,
BUFFER_SIZE , 0)) < 0)
```

Dari kode di atas terdapat peintah Recv_Msg_Length yang menunjukan dimana paket data yang diterima adalah recv (Client_Socket, Buffer_*Ms*gs, BUFFER_SIZE) atau menunjukan paket data yang diterima berupa socket dari *client* dan Buffer_Msgs yang merupakan alokasi buffer yang disediakan untuk menampung data di *server*.

f) Fungsi Send :

Fungsi send pada sisi *server* digunakan untuk mengirimkan informasi. Prosesnya, *server* mengirimkan sinyal *response* kepada *client* setelah *client* mengirimkan sinyal *request*. Jika *socket* sudah terhubung, fungsi send dapat digunakan untuk mengirimkan data. Ketika terjadi panggilan kembali, data yang telah antri ditransmisikan melalui koneksi *socket* yang telah terhubung sebelumnya.

```
if (send(Client_Socket, Buffer_Msgs, Recv_Msg_Length,
0) != Recv_Msg_Length)
```

Dari kode di atas terdapat peintah send Client_Socket, Buffer_*Ms*gs, Recv_Msg_Length. Client_Socket menunjukan port dan IP Address masukan dari client, `Buffer_Msgs`, menunjukan besaran alokasi buffer yang disediakan untuk

menampung data yang dikirim oleh *client* dan Recv_Msg_Length menunjukan besaran paket data yang diterima oleh *server*.

2. Berkas program main.cpp

a) Fungsi Socket

Berkas program ini digunakan untuk Membuat *TCP* atau *UDP socket*, yang kemudian dapat digunakan sebagai titik awal komunikasi untuk mengirim dan menerima data menggunakan protokol protocol yang telah ditentukan tersebut.

```
if ((sock = socket( PF_INET,  SOCK_STREAM,
IPPROTO_TCP))<0) perror("socket() command failed");
```

Dari kode diatas PF_NET menunjukan bahwa *socket* ditunjukan untuk koneksi internet, SOCK_STREAM dan IPPROTO_TCP menunjukan bahwa program ini menggunakan stream socket *TCP* yang berarti *connection oriented*.

b) Fungsi Connect :

Membentuk koneksi antara *socket* tujuan (*server*) dan *socket remote* (*client*) dan mengembalikan nilai *character* (*message*) dari sisi *client*. Untuk *TCP sockets*, perintah connect() akan mengembalikan nilai *character* (*message*) setelah tahapan proses *three ways handshaking* berhasil dilakukan.

```
if (connect(sock, (struct sockaddr*) &Server_IP,
sizeof(Server_IP)) <0)
printf("\n\nMemulai Pengiriman Paket Data...\n  ") ;
```

Dari kode diatas struck sockaddr menunjukan *port* tujuan pada *server*, &Server_IP menunjukan *IP Address* tujuan yaitu *IP Address server*, sizeof menunjukan besaran paket data yang akan dikirimkan *client* pada *server*.

c) Fungsi Send :

Baik *server* maupun *client* keduanya membutuhkan fungsi ini untuk mengirimkan informasi. Prosesnya, *client* mengirimkan sinyal *request*, kemudian *server* mengirimkan sinyal *response*. Jika *socket* sudah terhubung, *fungsi send* dapat digunakan untuk mengirimkan paket data.

```
if (mode==1) {
      Msgs_Length = strlen(Msgs128);
      printf("Percobaan-%d (128Byte) = ", steps);
      if (send(sock, Msgs128, Msgs_Length, 0) !=
Msgs_Length)
                  perror("send() command error");

}else if (mode==2) {
      Msgs_Length = strlen(Msgs256);
      printf("Percobaan-%d (256Byte) = ", steps);
      if (send(sock, Msgs256, Msgs_Length, 0) !=
Msgs_Length)
                  perror("send() command error");

}else if (mode==3) {
      Msgs_Length = strlen(Msgs512);
      printf("Percobaan-%d (512Byte) = ", steps);
      if (send(sock, Msgs512, Msgs_Length, 0) !=
Msgs_Length)
                  perror("send() command error");

}else if (mode==4) {
      Msgs_Length = strlen(Msgs1024);
      printf("Percobaan-%d (1024Byte) = ", steps);
      if (send(sock, Msgs1024, Msgs_Length, 0) !=
Msgs_Length)
            perror("send() command error");
}else  {Msgs_Length = strlen(Msgs1500);
      printf("Percobaan-%d (1280Byte) = ", steps);
      if (send(sock, Msgs1280, Msgs_Length, 0) !=
Msgs_Length)
            perror("send() command error");
```

Dari kode di atas terdapat perintah *Ms*gs_Length yang menunujukan besaran paket data (*byte*) yang dikirimkan, perintah strlen (Msgs 128, 256, 512,

1024, 1500) yang menunjukan jenis besaran paket data (*byte*) yang akan dikirimkan. Pada program *embedded network analyzer* yang penulis buat, penulis membuat pilihan kondisi besaran jenis paket data yang akan dikirimkan. Besaran paket data yang dapat dikirmkan adalah 128 *byte*, 256, *byte*, 512 *byte*, 1024 *byte* dan 1500 *byte*.

d) Fungsi Recv :

Fungsi recv digunakan oleh *client* untuk menerima data dari *server* yang berupa sinyal *response* hasil sinyal *request* yang *client* kirimkan sebelumnya kepada *server*.

```
if ((Bytes_Rcvd = recv(sock, Buffer_Msgs, Msgs_Length -
1, 0)) <= 0)
```

Dari kode di atas terdapat peintah *Byte*s_recv yang menunjukan dimana paket data diterima, recv (sock, Buffer_Msgs, Msgs_Length) menunjukan paket data yang diterima berupa *socket*, *buffer* dan panjang data atau ukuran paket data.

e) Fungsi Close :

Fungsi *close* digunakan untuk memerintahkan sistem untuk menutup penggunaan *socket*.

```
close(sock);
    exit(0);
```

Setelah mendefinisikan masing-masing fungsi fungsi yang dipakai pada pemrograman *socket* untuk sisi *server* dan *client*, berikut penulis menampilkan juga baris *berkas* yang menunjukan *variable* dan fungsi yang dipakai dalam melakukan *scanning port* dan perhitungan parameter *packet loss*, *troughput*, *delay* dan *jitter*.

- Fungsi *Scanning port*

```
sprintf(cmd,"nmap%s-
np21,22,23,80,%d>log.txt",servIP,Server_PORT);
```

Penjelasan dari baris kode program diatas adalah fungsi untuk menjalankan program nmap yang sebelumnya telah terinstal pada sistem operasi yang digunakan. Nantinya program nmap akan melakukan *scanning port* pada *port* 21, 22, 23, 80 dan *port* tujuan yang nantinya akan dipakai oleh sisi *server*.

- Variabel metode percobaan dalam program

```
int steps = 0;
int tryCount = 10;
int timeWait = 1 ;
```

Penjelasan :

- int steps = 0; digunakan untuk memberikan fungsi setiap tahap percobaan yang dilakukan.
- int tryCount = 10; digunakan untuk memberi fungsi percobaan pengiriman paket data yang dilakukan oleh client. Pada penelitian *embedded system network analyzer* pada jaringan *LAN* penulis memberikan nilai 10, yang berarti akan dilakukan 10 x percobaan di setiap skenario pengiriman paket data.
- Int timeWait 1; digunakan untuk memberi fungsi batasan waktu sebesar 1 detik pada setiap kali dilakukan percobaan pengiriman paket data oleh *client* ke *server*.

- Variable dan fungsi Perhitungan parameter

Variable dan fungsi yang penulis gunakan dalam membuat *berkas* program pada penelitian *embedded system network analyzer* pada jaringan *LAN* disesuaikan dengan rumus perhitungan parameter *packet loss*, *troughput*, *delay* dan *jitter* yang terdapat pada BAB II.

1. Rumus perhitungan *packet loss* pada program :

- Perhitungan *packet loss* per percobaan :

```
double ploss = (((Msgs_Length -Total_Bytes_Rcvd) / Msgs_Length) /
elapsedTime) * 0.001;
```

Baris berkas ini menunjukan perhitungan *packet loss*, yaitu jumlah paket data yang diterima dibagi jumlah paket data yang dikirimkan kemudian dikali 100%, Perhitungan *packet loss* per percobaan dilakukan untuk mengetahui berapa jumlah paket data yang hilang pada setiap percobaan yang dilakukan.

- Perhitungan *packet loss* rata-rata :

```
double avgpacketloss = totPloss / tryCount;
```

Baris berkas ini menunjukan perhitungan *packet loss* yang dihitung berdasarkan total *packet loss* dari sepuluh kali percobaan yang dilakukan. Perhitungan *packet loss* rata-rata bertujuan untuk mengetahui berapa jumlah rata-rata paket data yang hilang dari total sepuluh kali percobaan yang dilakukan.

2. Rumus perhitungan *Troughput* pada program :

- Perhitungan *troughput* per percobaan :

```
(Total_Bytes_Rcvd / elapsedTime) *1000.0);
```

Baris berkas ini menunjukan perhitungan *troughput*, yaitu jumlah paket data yang diterima dibagi waktu yang digunakan. Perhitungan *troughput* per percobaan bertujuan untuk mengetahui jumlah paket data yang berhasil diterima pada setiap percobaan yang dilakukan.

```
double avgtroughput = totPut / tryCount;
```

Baris berkas ini menunjukan perhitungan *troughput* rata-rata dari total penjumlahan *troughput* dibagi sepuluh kali percobaan yang dilakukan. Perhitungan *troughput* rata-rata bertujuan untuk mengetahui berapa rata-rata jumlah paket data yang berhasil diterima dari sepuluh kali percobaan yang dilakukan.

3. Rumus perhitungan *Delay* pada program :

- Perhitungan *delay* per percobaan :

```
elapsedTime = (t2.tv_sec - t1.tv_sec) * 1000.0;
```

```
elapsedTime += (t2.tv_usec - t1.tv_usec) / 1000.0;
delay = elapsedTime
```

Baris berkas ini menunjukan perhitungan *delay*, Sebelumnya penulis memberikan fungsi bahwa *delay* adalah durasi pengiriman paket data, sehingga rumus *delay* adalah yaitu waktu akhir saat paket data diterima atau datang dikurangi waktu awal saat pengiriman paket data dilakukan.

- Perhitungan delay rata-rata :

```
double delay = timeDuration ;
     double averagesDelay = delay / tryCount;
     double delayCount = 0;
```

Baris berkas ini menunjukan perhitungan *delay* rata-rata dari total penjumlahan *delay* dibagi sepuluh kali percobaan yang dilakukan. Perhitungan *delay* rata-rata bertujuan untuk mengetahui berapa rata-rata waktu yang digunakan dalam proses pengiriman dan penerimaan paket data.

4. Rumus perhitungan *Jitter* pada program:

- Perhitungan *jitter* per percobaan :

```
timeDuration += elapsedTime;
          arrayDelay[steps] = elapsedTime;
          if (steps != 0) {
               diffDelay = ( arrayDelay[steps] -
arrayDelay[steps-1]);
               if(diffDelay<0){
               diffDelay = -diffDelay;
               }
               sumJitter += diffDelay;
          }
```

Baris program ini menunjukan perhitungan *jitter* per percobaan dimana pada BAB II diterangkan bahwa *jitter* adalah variasi nilai *delay*. Sehingga rumus *jitter* pada program adalah nilai *delay* ke i dikurangi nilai *delay* ke i+1,

maksudnya adalah nilai delay yang didapat pada iterasi ke – i akan dikurangi nilai delay iterasi setelahnya atau i+1.

- Perhitungan *jitter* rata-rata :

```
double avgJitter = (sumJitter/tryCount);
if(avgJitter<0){
avgJitter = -avgJitter;
}
```

Baris berkas ini menunjukan perhitungan *jitter* rata-rata dari total penjumlahan *jitter* dibagi sepuluh kali percobaan yang dilakukan. Perhitungan *jitter* rata-rata bertujuan untuk mengetahui berapa rata-rata nilai *jitter* dari sepuluh percobaan kali percobaan yang dilakukan.

3. Berkas program kbd.h

Berkas program kbd.h berfungsi untuk konfigurasi input atau masukan data berupa karakter, tanda titik dan spasi, serta fungsi hapus dan arah (atas, bawah, kiri dan kanan) saat program ditampilkan pada layar *LCD*.

```
if (ev.type == EV_KEY && ev.value==up_down)
    {
        switch (ev.code)
        {
        case KEY_0:
        case KEY_1:
        case KEY_2:
        case KEY_3:
        case KEY_4:
        case KEY_5:
        case KEY_6:
        case KEY_7:
        case KEY_8:
        case KEY_9:
        case KEY_DOT:
```

```
        case KEY_SPACE:
        case KEY_ENTER:
        case KEY_BACKSPACE:
        case KEY_UP: //eUp:
        case KEY_RIGHT: //eRight:
        case KEY_LEFT: //eLeft:
        case KEY_DOWN: //eDown:
            code = ev.code;
            break;
        default:
            break;
        }
    }
```

4. Berkas Program entry.h

Berkas program entry.h berfungsi sebagai konfigurasi data masukan dari keyboard yang berupa karakter, tanda titik dan spasi, serta fungsi hapus dan arah (atas, bawah, kiri dan kanan) untuk ditampilakn di layar *LCD*.

```
    lcds_puts(entry_buffer); //tampilkan entry terakhir
    #endif

while(1){
        key = kbd_get(KBD_DOWN); //baca tombol keyboard
pada saat ditekan

        switch(key){
            case KEY_0:
            case KEY_1:
            case KEY_2:
            case KEY_3:
            case KEY_4:
            case KEY_5:
            case KEY_6:
            case KEY_7:
```

```
            case KEY_8:
            case KEY_9:
            case KEY_DOT:
            case KEY_SPACE:
                //cegah overwrite outside buffer
                if(i<ENTRY_BUFFER_LEN){
                    ch = entry_key_to_char(key);
                    entry_buffer[i++] = ch;
                    lcds_putch(ch);
                }
                break;
            case KEY_BACKSPACE:
                //hapus karakter sekarang dan kembali
ke posisi sebelumnya
                if(i>=0){
                    ch = ' ';
                    entry_buffer[i] = ch;
                    if(i>0) i--;
                    lcds_goto(ENTRY_ROW,i);
                    lcds_putch(ch);
                    lcds_goto(ENTRY_ROW,i);
                }
                break;
           case KEY_ENTER:
                entry_buffer[i++]='   ';  //to  enable
scanning data through loop
                entry_buffer[i]   =   0;   //end   of
string/entry
                break;
        }
```

5. Berkas program lcd.h

Berkas program lcd.h digunakan sebagai konfigurasi antara driver pada *LCD* dan setingan fungsi program yang lain sehinga dapat ditampilkan pada layar *LCD* tersebut.

```
int lcds_init(unsigned char col){
   int status = -1;
       status = uart_9600_init();
       col_num = col;
       _delay_ms(500); //wait after power up
       return status;
}
```

6. Berkas Program buf.h

Berkas program buf.h digunakan sebagai digunakan untuk menyediakan dan menampung buffer karakter, yang akan ditampilkan pada *LCD*.

```
void buf_view(char rowv, char colv){
    unsigned int r, c, rr, cc;
    char chr;
    for(r=0;    rr    =    r    +    (rowv*LCD_ROW_NUM),
r<LCD_ROW_NUM; r++){
        //printf("%d:",rr);
        for(c=0;     cc     =     c+(colv*LCD_COL_NUM),
c<LCD_COL_NUM; c++){
            //printf("%d,",cc);
            if(rr>=BUF_ROW_NUM || cc>=BUF_COL_NUM){
                chr = 'x';
            }
            else
                chr = buf[rr][cc];
            lcd_buf[r][c] = chr==0?' ':chr;
            //printf("%c",chr);
        }
```

```
        lcd_buf[r][c] = 0;
        //printf("\n");
```

7. Berkas Program redirect.h

Berkas program ini berfungsi sebagai pengubah perintah tampilan (perintah printf) pada program yang terdapat perangkat *beagleboard –xm* menjadi bentuk karakter pada *LCD*.

```
//menginisialisasi redirect supaya printf
int redirect_init(){
    //lcds_puts("redirect_init begin\n");
    saved_stdout = dup(STDOUT_FILENO);  /* save stdout
for display later */
    if( pipe(out_pipe) != 0 ) {          /* make a pipe
*/
        return 0;
        //exit(1);
    }
    dup2(out_pipe[1], STDOUT_FILENO);      /* redirect
stdout to the pipe */
    close(out_pipe[1]);
```

8. Berkas Program serial.h

Berkas program ini digunakan sebagai konfigurasi komunikasi antara *beagleboard –XM* dengan layar *LCD* yang menggunakan *output serial*.

```
signal(SIGINT,(sighandler_t)sigint_handler);    //attach
handler
        tcgetattr(serial_port,&options_original);
        tcgetattr(serial_port, &options);

        cfsetispeed(&options, BAUDRATE); //B115200
        cfsetospeed(&options, BAUDRATE);

        options.c_cflag |= (CLOCAL | CREAD); //CS8
        //set 8N1
        options.c_cflag &= ~PARENB;
        options.c_cflag &= ~CSTOPB;
        options.c_cflag &= ~CSIZE;
        options.c_cflag |= CS8;
        options.c_cflag &= ~CRTSCTS;
```

4.3 Implementasi Perangkat Lunak

Setelah melakukan pembuatan berkas program, tahapan selanjutnya adalah implementasi perangkat lunak hasil dari *compile* berkas program yang telah dibuat. Pada tahapan implementasi perangkat lunak, terdapat dua *berkas* program utama yaitu *berkas* program untuk sisi *server* dan berkas program untuk sisi *client*. Pada sub-bab impelementasi perangkat lunak penulis menampilkan tampilan hasil implementasi program yang telah dibuat dan diimplementasikan pada perangkat *beagleboard –XM* menggunakan akses *remote ssh* dikarenakan untuk memudahkan tampilan keseluruhan hasil program. Berikut tampilan hasil implementasi dari *berkas* program yang telah penulis buat :

4.3.1 Sisi Server

Perangkat lunak yang berjalan pada sisi server, berfungsi untuk melakukan *reply* paket data yang dikirimkan oleh sisi *client*. Berikut tampilan hasil implementasi perangkat lunak pada sisi server :

```
root@beagleboard:~# ./server.out 2211

Start listening...

Client connected 192.168.137.4
```

Gambar 4.4 Implementasi Sisi Server

Gambar 4.4 menunjukan hasil implementasi perangkat lunak server.out, telah dijalankan pada port 2211 dan setelah program server.out dijalankan terdapat tulisan *start listening* yang menunjukan sisi *server* telah aktif dan siap melakukan *reply* seandainya ada *client* yang melakukan *request* kepada sisi *server*. Format konfigurasi untuk menjalankan program server.out adalah ./server.out <Server Port>.

4.3.2 Sisi Client

Perangkat lunak yang berada pada sisi *client* berfungsi sebagai pengirim paket data kepada sisi *server*. Pada penelitian *embedded system network analyzer* pada jaringan *LAN* penulis memberikan lima pilihan besaran paket data yang dapat dikirimkan oleh perangkat lunak pada sisi *client* menuju sisi *server*. Berikut tampilan hasil implementasi perangkat lunak pada sisi *client* :

- Implementasi Scanning Port

Implementasi *scanning port* menampilkan hasil dari implementasi saat program melakukan *scanning port* sebelum melakukan pengiriman paket data. antara lain *port* 21 *FTP*, *port* 22 *SSH*, *port* 23 *Telnet*, *port* 80 *HTTP*, dan *scanning port* tujuan. Hal ini dilakukan untuk mengetahui apakah kondisi jaringan yang digunakan terdapat *firewall* maupun pemblokiran *port-port* tujuan sehingga proses pengiriman paket data tidak dapat dilakukan. Berikut tampilan program saat melakukan proses *scanning port* :

```
Scanning Port : Memeriksa Port Pilihan dan Tujuan 21,22,23,80 2211

Starting Nmap 6.25 ( http://nmap.org ) at 2013-03-11 14:26 GMT
Nmap scan report for 192.168.137.4
Host is up (0.0014s latency).
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
23/tcp   open  telnet
80/tcp   open  http
2211/tcp open  unknown
MAC Address: 08:00:27:A3:7A:35 (Cadmus Computer Systems)

Nmap done: 1 IP address (1 host up) scanned in 0.45 seconds
```

Gambar 4.5 Implementasi Scanning Port Berhasil

Gambar 4.5 merupakan implementasi saat kondisi jaringan tidak terdapat *firewall* maupun pemblokiran *port-port* tujuan di jaringan yang ditunjukan oleh status pada tampilan hasil eksekusi program menunjukan *open*. Sehingga proses pengiriman paket data dari sisi *client* menuju sisi *server* siap untuk dilakukan.

```
Scanning Port : Memeriksa Port Pilihan dan Tujuan 21,22,23,80 2211

Starting Nmap 6.25 ( http://nmap.org ) at 2013-03-11 14:33 GMT
Nmap scan report for 192.168.137.4
Host is up (0.0014s latency).
PORT     STATE    SERVICE
21/tcp   filtered ftp
22/tcp   filtered ssh
23/tcp   filtered telnet
80/tcp   filtered http
2211/tcp filtered unknown
MAC Address: 08:00:27:A3:7A:35 (Cadmus Computer Systems)

Nmap done: 1 IP address (1 host up) scanned in 3.55 seconds
```

Gambar 4.6 Implementasi Scanning Port Terblokir

Gambar 4.6 merupakan implementasi saat kondisi jaringan terdapat *firewall* atau pemblokiran pada *port-port* pilihan dan *port* tujuan yang ditunjukan oleh status tampilan hasil eksekusi program menunjukan *filtered.* Sehingga proses pengiriman paket data dari sisi *client* menuju sisi *server* tidak dapat untuk dilakukan.

- Impelementasi Hasil Perhitungan :

Implementasi hasil perhitungan menampilkan hasil perhitungan pada setiap percobaan dan hasil perhitungan rata-rata beradasarkan rumus perhitungan yang telah ditentukan dan diimplementasikan di dalam bahasa pemrograman, berikut tampilan hasil dari hasil perhitungan parameter kualitas jaringan berupa *packet loss*, *troughput*, *delay* dan *jitter* yang telah dilakukan.

```
Percobaan-0 (128Byte) =
 PLoss:0.00 %; Tput:606635.07 KBps; delay:0.2110 ms; J:0.0000 ms
Percobaan-1 (128Byte) =
 PLoss:0.00 %; Tput:624390.24 KBps; delay:0.2050 ms; J:0.0060 ms
Percobaan-2 (128Byte) =
 PLoss:0.00 %; Tput:633663.37 KBps; delay:0.2020 ms; J:0.0030 ms
Percobaan-3 (128Byte) =
 PLoss:0.00 %; Tput:524590.16 KBps; delay:0.2440 ms; J:0.0420 ms
Percobaan-4 (128Byte) =
 PLoss:0.00 %; Tput:630541.87 KBps; delay:0.2030 ms; J:0.0410 ms
Percobaan-5 (128Byte) =
 PLoss:0.00 %; Tput:455516.01 KBps; delay:0.2810 ms; J:0.0780 ms
Percobaan-6 (128Byte) =
 PLoss:0.00 %; Tput:568888.89 KBps; delay:0.2250 ms; J:0.0560 ms
Percobaan-7 (128Byte) =
 PLoss:0.00 %; Tput:612440.19 KBps; delay:0.2090 ms; J:0.0160 ms
Percobaan-8 (128Byte) =
 PLoss:0.00 %; Tput:547008.55 KBps; delay:0.2340 ms; J:0.0250 ms
Percobaan-9 (128Byte) =
 PLoss:0.00 %; Tput:609523.81 KBps; delay:0.2100 ms; J:0.0240 ms

Hasil Perhitungan Parameter(Rata-rata) =

-Packet Loss      : 0.00 %
-Throughput       : 581319.82 Bps
-Delay            : 0.2224 ms
-Jitter           : 0.0291 ms
```

Gambar 4.7 Implementasi Hasil Perhitungan 128 Byte

```
Percobaan-0 (256Byte) =
 PLoss:0.00 %; Tput:548179.87 KBps; delay:0.4670 ms; J:0.0000 ms
Percobaan-1 (256Byte) =
 PLoss:0.00 %; Tput:535564.85 KBps; delay:0.4780 ms; J:0.0110 ms
Percobaan-2 (256Byte) =
 PLoss:0.00 %; Tput:664935.06 KBps; delay:0.3850 ms; J:0.0930 ms
Percobaan-3 (256Byte) =
 PLoss:0.00 %; Tput:570155.90 KBps; delay:0.4490 ms; J:0.0640 ms
Percobaan-4 (256Byte) =
 PLoss:0.00 %; Tput:579185.52 KBps; delay:0.4420 ms; J:0.0070 ms
Percobaan-5 (256Byte) =
 PLoss:0.00 %; Tput:570155.90 KBps; delay:0.4490 ms; J:0.0070 ms
Percobaan-6 (256Byte) =
 PLoss:0.00 %; Tput:536687.63 KBps; delay:0.4770 ms; J:0.0280 ms
Percobaan-7 (256Byte) =
 PLoss:0.00 %; Tput:457960.64 KBps; delay:0.5590 ms; J:0.0820 ms
Percobaan-8 (256Byte) =
 PLoss:0.00 %; Tput:436115.84 KBps; delay:0.5870 ms; J:0.0280 ms
Percobaan-9 (256Byte) =
 PLoss:0.00 %; Tput:636815.92 KBps; delay:0.4020 ms; J:0.1850 ms

Hasil Perhitungan Parameter(Rata-rata) =

-Packet Loss      : 0.00 %
-Throughput       : 553575.72 Bps
-Delay            : 0.4695 ms
-Jitter           : 0.0505 ms
```

Gambar 4.8 Implementasi Hasil Perhitungan 256 Byte

```
Percobaan-0 (512Byte) =
 PLoss:0.00 %; Tput:2031746.03 KBps; delay:0.2520 ms; J:0.0000 ms
Percobaan-1 (512Byte) =
 PLoss:0.00 %; Tput:1358090.19 KBps; delay:0.3770 ms; J:0.1250 ms
Percobaan-2 (512Byte) =
 PLoss:0.00 %; Tput:1239709.44 KBps; delay:0.4130 ms; J:0.0360 ms
Percobaan-3 (512Byte) =
 PLoss:0.00 %; Tput:1242718.45 KBps; delay:0.4120 ms; J:0.0010 ms
Percobaan-4 (512Byte) =
 PLoss:0.00 %; Tput:1264197.53 KBps; delay:0.4050 ms; J:0.0070 ms
Percobaan-5 (512Byte) =
 PLoss:0.00 %; Tput:1166287.02 KBps; delay:0.4390 ms; J:0.0340 ms
Percobaan-6 (512Byte) =
 PLoss:0.00 %; Tput:1051334.70 KBps; delay:0.4870 ms; J:0.0480 ms
Percobaan-7 (512Byte) =
 PLoss:0.00 %; Tput:1199063.23 KBps; delay:0.4270 ms; J:0.0600 ms
Percobaan-8 (512Byte) =
 PLoss:0.00 %; Tput:1044897.96 KBps; delay:0.4900 ms; J:0.0630 ms
Percobaan-9 (512Byte) =
 PLoss:0.00 %; Tput:1120350.11 KBps; delay:0.4570 ms; J:0.0330 ms

Hasil Perhitungan Parameter(Rata-rata) =

-Packet Loss      : 0.00 %
-Throughput       : 1271839.47 Bps
-Delay            : 0.4159 ms
-Jitter           : 0.0407 ms
```

Gambar 4.9 Implementasi Hasil Perhitungan 512 Byte

```
Percobaan-0 (1024Byte) =
 PLoss:0.00 %; Tput:4762790.70 KBps; delay:0.2150 ms; J:0.0000 ms
Percobaan-1 (1024Byte) =
 PLoss:0.00 %; Tput:4899521.53 KBps; delay:0.2090 ms; J:0.0060 ms
Percobaan-2 (1024Byte) =
 PLoss:0.00 %; Tput:4995121.95 KBps; delay:0.2050 ms; J:0.0040 ms
Percobaan-3 (1024Byte) =
 PLoss:0.00 %; Tput:4946859.90 KBps; delay:0.2070 ms; J:0.0020 ms
Percobaan-4 (1024Byte) =
 PLoss:0.00 %; Tput:4995121.95 KBps; delay:0.2050 ms; J:0.0020 ms
Percobaan-5 (1024Byte) =
 PLoss:0.00 %; Tput:4452173.91 KBps; delay:0.2300 ms; J:0.0250 ms
Percobaan-6 (1024Byte) =
 PLoss:0.00 %; Tput:4785046.73 KBps; delay:0.2140 ms; J:0.0160 ms
Percobaan-7 (1024Byte) =
 PLoss:0.00 %; Tput:4785046.73 KBps; delay:0.2140 ms; J:0.0000 ms
Percobaan-8 (1024Byte) =
 PLoss:0.00 %; Tput:4740740.74 KBps; delay:0.2160 ms; J:0.0020 ms
Percobaan-9 (1024Byte) =
 PLoss:0.00 %; Tput:4785046.73 KBps; delay:0.2140 ms; J:0.0020 ms

Hasil Perhitungan Parameter(Rata-rata) =

-Packet Loss      : 0.00 %
-Throughput       : 4814747.09 Bps
-Delay            : 0.2129 ms
-Jitter           : 0.0059 ms
```

Gambar 4.10 Implementasi Hasil Perhitungan 1024 Byte

```
Percobaan-0 (1500Byte) =
 PLoss:0.00 %; Tput:2757352.94 KBps; delay:0.5440 ms; J:0.0000 ms
Percobaan-1 (1500Byte) =
 PLoss:0.00 %; Tput:1391465.68 KBps; delay:1.0780 ms; J:0.5340 ms
Percobaan-2 (1500Byte) =
 PLoss:0.00 %; Tput:1704545.45 KBps; delay:0.8800 ms; J:0.1980 ms
Percobaan-3 (1500Byte) =
 PLoss:0.00 %; Tput:1544799.18 KBps; delay:0.9710 ms; J:0.0910 ms
Percobaan-4 (1500Byte) =
 PLoss:0.00 %; Tput:3658536.59 KBps; delay:0.4100 ms; J:0.5610 ms
Percobaan-5 (1500Byte) =
 PLoss:0.00 %; Tput:2835538.75 KBps; delay:0.5290 ms; J:0.1190 ms
Percobaan-6 (1500Byte) =
 PLoss:0.00 %; Tput:2423263.33 KBps; delay:0.6190 ms; J:0.0900 ms
Percobaan-7 (1500Byte) =
 PLoss:0.00 %; Tput:3246753.25 KBps; delay:0.4620 ms; J:0.1570 ms
Percobaan-8 (1500Byte) =
 PLoss:0.00 %; Tput:3318584.07 KBps; delay:0.4520 ms; J:0.0100 ms
Percobaan-9 (1500Byte) =
 PLoss:0.00 %; Tput:3000000.00 KBps; delay:0.5000 ms; J:0.0480 ms

Hasil Perhitungan Parameter(Rata-rata) =

-Packet Loss      : 0.00 %
-Throughput       : 2588083.92 Bps
-Delay            : 0.6445 ms
-Jitter           : 0.1808 ms
```

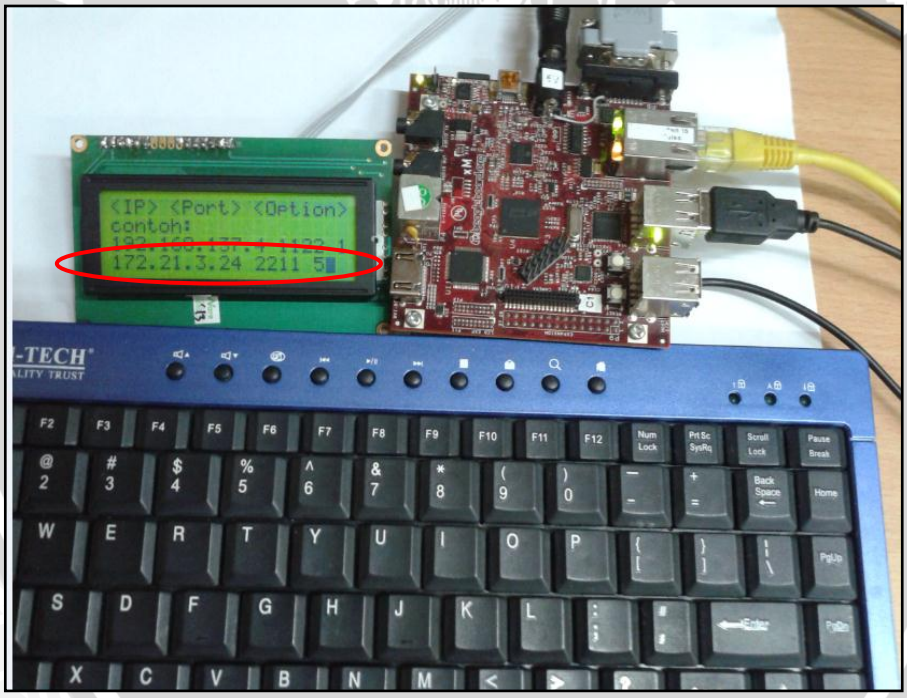
Gambar 4.11 Implementasi Hasil Perhitungan 1500 Byte

Gambar 4.7, 4.8, 4.9, 4.10 dan 4.11 menunjukan implementasi hasil perhitungan menggunakan program *network analyzer* untuk melakukan perhitungan parameter *packet loss, troughput, delay* dan *jitter*. Gambar-gambar tersebut menunjukan hasil perhitungan setiap percobaan sebanyak sepuluh kali dan hasil rata-rata dari seluruh percobaan yang telah dilakukan pada percobaan pengiriman paket data 128 *byte*, 256 *byte*, 512 *byte*, 1024 *bye* dan 1500 *byte*.

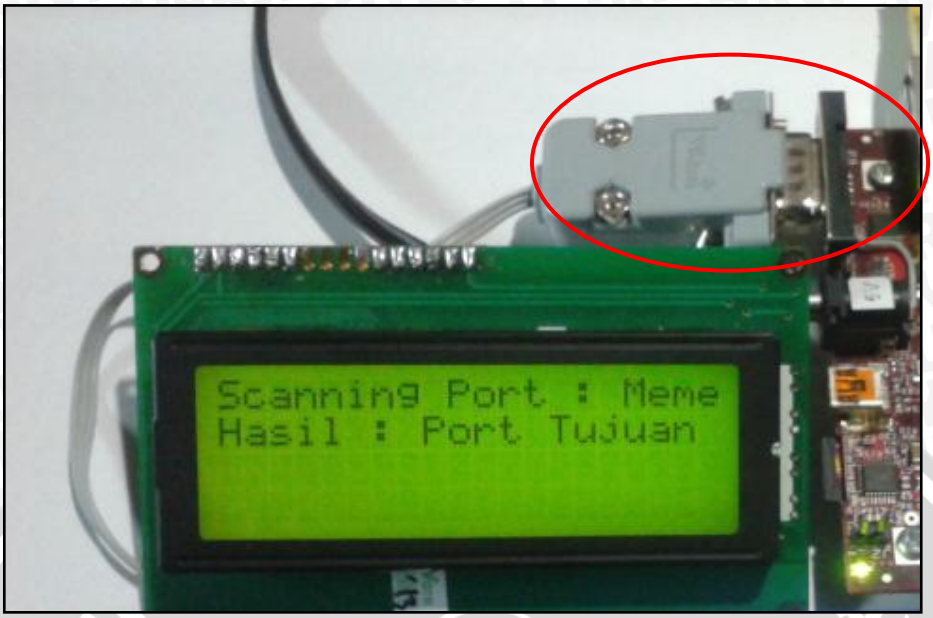
4.4 Implementasi Perangkat Keras

Implementasi perangkat keras mengarah pada hasil implementasi perangkat lunak yang dibuat, kemudian diimplementasikan pada perangkat keras yang digunakan. Pada sub-bab implementasi perangkat keras ini, juga merupakan hasil implementasi dari sistem *embedded system* yang dibangun. Hasil – hasil implementasi yang ditampilkan adalah hasil dari masing - masing fungsi yang ditujukan untuk melakukan tugas tertentu seperti fungsi *output* pada *LCD*, fungsi inputan dari *keyboard*, fungsi untuk konfigurasi kapasitas *buffer* dan fungsi konfigurasi untuk komunikasi *serial* yang menghubungkan perangkat *beagleboard –XM* dan *LCD*. Berikut tampilan dari hasil-hasil implementasi perangkat keras yang juga termasuk hasil implementasi dari sistem *embedded system*.

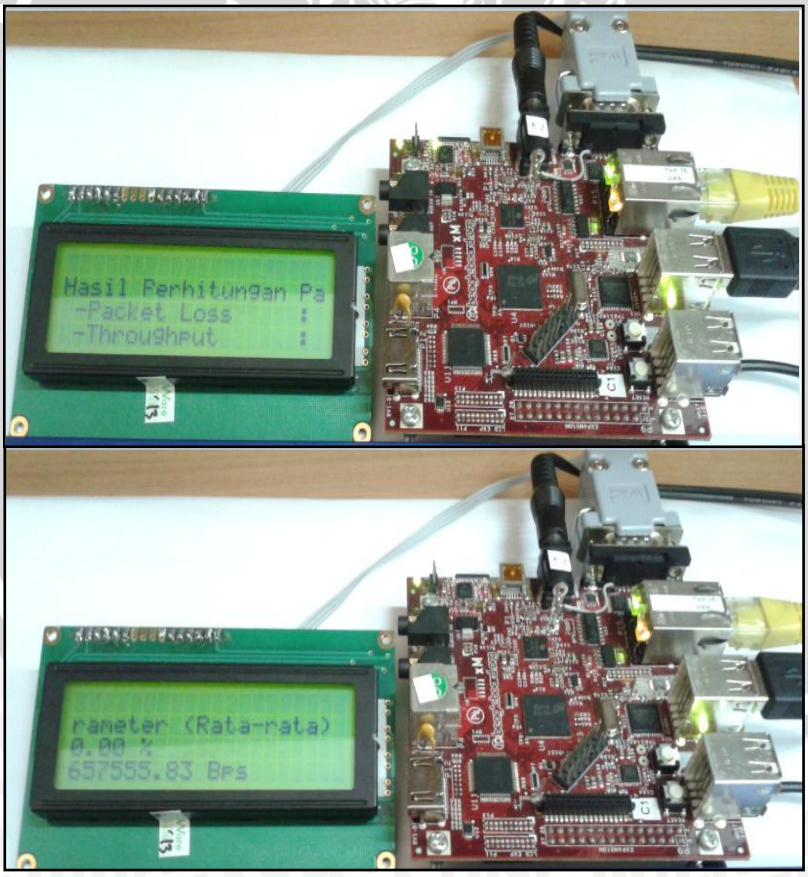
Gambar 4.12 Tampilan Awal Program Network Analyzer



Gambar 4.13 Tampilan Fungsi Input Keyboard



Gambar 4.14 Tampilan Fungsi Komunikasi Serial



Gambar 4.15 Tampilan Hasil Perhitungan Parameter

Gambar 4.11, 4.12, 4.13, 4.14 dan 4.15 menunjukan hasil eksekusi perangkat lunak client.out beserta program penunjang yang dibutuhkan sebagai *library* yaitu berkas program kbd.h, entry.h, lcds.h, buf.h, redirect.h dan serial.h yang memiliki fungi antara antara lain konfigurasi *LCD, buffer*, komunikasi serial dan konfigurasi input dari *keyboard* dan merupakan hasil eksekusi client.out beserta program penunjang lainnya yang telah diimplementasikan pada perangkat beagleboard –*XM* dan ditampilkan pada layar *LCD* di perangkat *beagleboard - XM*, format konfigurasi untuk menjalankan perangkat lunak ./client.out di layar *LCD* adalah <IP> adalah alamat *internet protocol server*, <Port> adalah nomer *port* yang dipakai oleh *server* dan <Option> adalah pilihan pengiriman paket data yaitu 128*byte*, 256 *byte*, 512 *byte*, 1024 *byte*, dan 1500 *byte*.

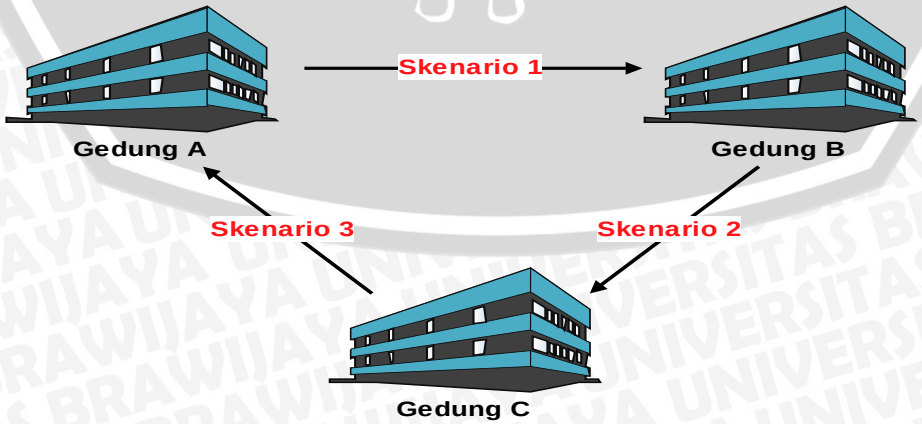
BAB V
PENGUJIAN DAN ANALISIS

Pada bab ini memuat hasil pengujian dan analisis kondisi sistem jaringan menggunakan perangkat lunak yang telah dibuat dan dijalankan menggunakan dua perangkat *beagleboard–XM*, pada sisi *client* dan sisi *server*. Hasil pengujian dan analisis berupa pengukuran besaran nilai parameter *packet loss*, *troughtput*, *delay* dan *jitter*. Pengujian dan analisis sistem perlu dilakukan untuk mengetahui apakah sistem yang telah dirancang dan diimplementasikan telah bekerja sesuai dengan tujuan dan dapat melakukan analisis terhadap kondisi jaringan yang digunakan untuk pengujian.

5.1 Pengujian

Dalam penelitian *embedded system network analyzer* pada jaringan *LAN*, pengujian dilakukan dengan tiga skenario pengujian, yang mana pada setiap skenario pengujian dilakukan pengiriman paket data antara *client* dan *server* dengan lokasi gedung di lingkungan PTIIK. Gedung-gedung yang digunakan untuk pengujian ini antara lain gedung A, gedung B dan gedung gedung C. Untuk skenario pengujian yang akan dilakukan adalah sebagai berikut

- Skenario 1 : Gedung A sebagai client, gedung B sebagai server.
- Skenario 2 : Gedung B sebagai client, gedung C sebagai server.
- Skenario 3 : Gedung C sebagai client, gedung A sebagai server.



Gambar 5.1 Skenario Pengujian

Proses pengujian dilakukan dengan percobaan pengiriman paket data sepuluh kali, besaran paket data 128 *byte*, 256 *byte*, 512 *byte*, 1024 *byte* dan 1500 *byte* dan batasan waktu sebesar 1 detik.

Berikut variabel kontrol yang ditentukan penulis pada setiap skenario percobaan yang akan dilakukan :

- Jumlah percobaan = 10
- Besaran paket data = 128 *byte*, 256 *byte*, 512 *byte*, 1024 *byte*, dan1500 *byte*
- Batasan waktu setiap pengiriman paket data = 1 detik

Berikut parameter kualitas jaringan yang ditentukan penulis untuk diketahui nilainya dalam pengujian :

1. *Packet loss*, dalam satuan % (Persentase)

   Rumus perhitungan *packet loss* :

$$Packet\ loss = \frac{\sum Packets\ Lost}{\sum Packets\ Sent}\ \text{X}\ 100\% \quad \text{.....(5-1)}$$

2. *Troughtput*, dalam satuan *Bps* (*Byte per second*)

   Rumus perhitungan *troughput* :

$$Troughput = \frac{\sum Packets\ Delivered}{\sum Packets\ Arrival\ Time - Packets\ Start\ Time} \quad \text{.....(5-2)}$$

3. *Delay*, dalam satuan *ms* (*milisecond*)

   Rumus perhitungan *delay* :

$$Delay = \sum Packets\ Arrival\ Time - Packet\ Start\ Time \quad \text{.....(5-3)}$$

4. *Jitter*, dalam satuan *ms* (milisecond)

   Rumus perhitungan *jitter* :

$$Jitter = (R_i - S_i) - (R_{i+1} - S_{i+1}) \quad \text{.....(5-4)}$$

Besaran variabel kontrol (jumlah percobaan, besaran paket data dan batasan waktu) diatas ditentukan untuk mendapatkan nilai parameter kualitas jaringan yang variatif sehingga nantinya dapat menghasilkan nilai parameter kualitas jaringan yang lebih akurat. Pengujian yang akan dilakukan diharapkan

penulis dapat menghasilkan nilai parameter kualitas jaringan pada lokasi pengujian yaitu pada jaringan *local area network* yang berlokasi pada gedung A, gedung B dan gedung C PTIIK. Untuk konfigurasi jaringan, sesuai pada BAB IV penulis menggunakan konfigurasi *DHCP*, sehingga perangkat *beagleboard–XM* yang penulis gunakan sebagai perangkat *embedded system network analyzer* langsung mendapatkan alamat *internet protocol* secara otomatis.

Dari variable kontrol yang penulis tentukan dapat diartikan bahwa terdapat sepuluh kali percobaan pengiriman paket data pada setiap pilihan pengiriman besaran paket data (128 *byte*, 256 *byte*, 512 *byte*, 1024 *byte*, dan 1500 *byte*) terdapat batasan waktu sebesar satu detik di setiap percobaan pengiriman paket data. Proses pengujian yang dilakukan adalah sisi *client* mengirimkan paket data kepada sisi *server* kemudian hasil *reply* dari kiriman paket data tersebut dihitung kualitas layanan jaringannya berdasarkan parameter yang akan dihitung oleh program *network analyzer* yang terdapat pada sisi *client*.

Seluruh data hasil pengujian yang berupa nilai parameter kualitas jaringan, penulis dapatkan pada layar *LCD character* yang terdapat pada perangkat *beagleboard –XM* yang terdapat pada *sisi client*. Seluruh data yang diperoleh kemudian dianalisis untuk selanjutnya digunakan untuk menarik kesimpulan dan saran. Analisis data menggunakan metode pengujian deskripstif berupa nilai rata-rata, nilai maksimum dan nilai minimum. Hasil analisis akan digunakan untuk menarik kesimpulan terhadap kondisi jaringan pada skenario satu, skenario dua dan skenario tiga. Berikut penjelasan detail dari hasil pengujian beserta hasil analisis yang dilakukan pada skenario satu, skenario dua dan skenario tiga.

5.1.1 Skenario Satu

Pada pengujian skenario satu, percobaan yang dilakukan adalah mengirimkan paket data dari gedung A menuju gedung B dengan *client* berada pada sisi gedung A dan *server* berada pada sisi gedung B. Berikut konfigurasi jaringan pada skenario satu :

Tabel 5.1 Konfigurasi jaringan skenario satu

Status	Alamat IP	Port
Client	172.21.1.54	2211
Server	172.21.3.30	2211

Tabel 5.1 menunjukan konfigurasi jaringan pada skenario satu, dengan konfigurasi alamat *IP client* 172.21.1.54 dan *server* 172.21.3.30 beserta *port* yang digunakan adalah 2211. Tujuan pengujian pada skenario satu adalah untuk mengetahui besaran nilai parameter *packet loss*, *troughtput*, *delay* dan *jitter* pada jaringan *local area network* antara gedung A dan gedung B. Berikut tampilan dari hasil pengujian dengan pengiriman paket data sebesar 128 *byte*, 256 *byte*, 512 *byte*, 1024 *byte*, 1024 *byte* dan 1500 *byte* dengan sepuluh kali percobaan pengiriman paket data dan batasan waktu sebesar satu detik pada setiap percobaan:

5.1.1.1 Paket Data 128 Byte

Tabel 5.2 Hasil Pengujian 128 Byte

Percobaan	Paket Data 128 Byte			
	Packet Loss (%)	Trough put (Bps)	Delay (ms)	Jitter (ms)
1	0	107563,03	1,1900	0,0000
2	0	110440,03	1,1590	0,0310
3	0	116575,59	1,0980	0,0610
4	0	123314,07	1,0380	0,0600
5	0	139890,71	0,9150	0,1230
6	0	74897,60	1,7090	0,7940
7	0	113374,67	1,1290	0,5800
8	0	119850,19	1,0680	0,0610
9	0	123314,07	1,0380	0,0300
10	0	123432,98	1,0370	0,0010
Rata-rata	0	115265,29	1,1381	0,1741

Manual Perhitungan :

Tabel 5.3 Perhitungan Manual Skenario 1 Paket Data 128 *Byte*

Percobaan	Paket Data 128 Byte			
	Packet Loss (%)	**Troughput (Bps)**	**Delay (ms)**	**Jitter (ms)**
	Rumus Perhitungan : Persamaan 5.1	**Rumus Perhitungan : Persamaan 5.2**	**Rumus Perhitungan : Persamaan 5.3**	**Rumus Perhitungan : Persamaan 5.4**
1	0	107563,0252	1,1900	0,0000
2	0	110440,0345	1,1590	0,0310
3	0	116575,5920	1,0980	0,0610
4	0	123314,0655	1,0380	0,0600
5	0	139890,7104	0,9150	0,1230
6	0	74897,6009	1,7090	0,7940
7	0	113374,6678	1,1290	0,5800
8	0	119850,1873	1,0680	0,0610
9	0	123314,0655	1,0380	0,0300
10	0	123432,9797	1,0370	0,0010
Rata-Rata	0	115265,2929	1,1381	0,1741

Berdasarkan tabel 5.2 pada pengiriman paket data sebesar 128 *byte* di skenario satu didapatkan nilai *packet loss* terbesar dan terkecil 0 % yang menunjukan bahwa dari sepuluh kali percobaan paket data berhasil terkirim seluruhnya. Untuk nilai *troughput* terbesar adalah 139890,71 *Bps* dan nilai *troughput* terkecil adalah 74897,60 *Bps*, nilai *delay* terbesar adalah 1,7090 *ms* dan nilai *delay* terkecil adalah 0,9150 *ms*, nilai *jitter* terbesar dalah 0,7940 dan nilai *jitter* terkecil adalah 0,0010 *ms*. Tabel 5.3 merupakan proses perhitungan secara manual untuk mendapatkan nilai dari parameter *packet loss*, *troughput*, *delay* dan *jitter* pada skenario satu, dengan besaran paket data 128 *byte*.

5.1.1.2 Paket Data 256 Byte

Tabel 5.4 Hasil Pengujian 256 Byte

Percobaan	Paket Data 256 Byte			
	Packet Loss (%)	Trough put (Bps)	Delay (ms)	Jitter (ms)
1	0	178397,21	1,4350	0,0000
2	0	204472,84	1,2520	0,1830
3	0	226749,34	1,1290	0,1230
4	0	214945,42	1,1910	0,0620
5	0	129097,33	1,9830	0,7920
6	0	204636,29	1,2510	0,7320
7	0	215126,05	1,1900	0,0610
8	0	226749,34	1,1290	0,0610
9	0	215126,05	1,1900	0,0610
10	0	215126,05	1,1900	0,0000
Rata-rata	0	203042,59	1,2940	0,2075

Manual Perhitungan :

Tabel 5.5 Perhitungan Manual Skenario 1 Paket Data 256 Byte

Percobaan	Paket Data 256 Byte			
	Packet Loss (%)	Troughput (Bps)	Delay (ms)	Jitter (ms)
	Rumus Perhitungan : Persamaan 5.1	Rumus Perhitungan : Persamaan 5.2	Rumus Perhitungan : Persamaan 5.3	Rumus Perhitungan : Persamaan 5.4
1	0	178397,2125	1,4350	0,0000
2	0	204472,8435	1,2520	0,1830
3	0	226749,3357	1,1290	0,1230
4	0	214945,4240	1,1910	0,0620
5	0	129097,3273	1,9830	0,7920
6	0	204636,2910	1,2510	0,7320
7	0	215126,0504	1,1900	0,0610
8	0	226749,3357	1,1290	0,0610
9	0	215126,0504	1,1900	0,0610

10	0	215126,0504	1,1900	0,0000
Rata-Rata	0	203042,5921	1,2940	0,2075

Berdasarkan tabel 5.4 pada pengiriman paket data sebesar 256 *byte* di skenario satu didapatkan nilai *packet loss* terbesar dan terkecil 0 % yang menunjukan bahwa dari sepuluh kali percobaan paket data berhasil terkirim seluruhnya. Untuk nilai *troughput* terbesar adalah 226749,34 *Bps* dan nilai *troughput* terkecil adalah 129097,33 *Bps*, nilai *delay* terbesar adalah 1,9830 *ms* dan nilai *delay* terkecil adalah 1,1900 *ms*, nilai *jitter* terbesar dalah 0,7920 dan nilai *jitter* terkecil adalah 0,0610 *ms*. Tabel 5.5 merupakan proses perhitungan secara manual untuk mendapatkan nilai dari parameter *packet loss*, *troughput*, *delay* dan *jitter* pada skenario satu, dengan besaran paket data 256 *byte*.

5.1.1.3 Paket Data 512 Byte

Tabel 5.6 Hasil Pengujian 512 Byte

Percobaan	Paket Data 512 Byte			
	Packet Loss (%)	Trough put (Bps)	Delay (ms)	Jitter (ms)
1	0	322621,30	1,5870	0,0000
2	0	349726,78	1,4640	0,1230
3	0	364672,36	1,4040	0,0600
4	0	215126,05	2,3800	0,9760
5	0	349488,05	1,4650	0,9150
6	0	364672,36	1,4040	0,0610
7	0	357043,24	1,4340	0,0300
8	0	356794,43	1,4350	0,0010
9	0	357043,24	1,4340	0,0010
10	0	390243,90	1,3120	0,1220
Rata-rata	0	342743,17	1,5319	0,2289

Manual Perhitungan :

Tabel 5.7 Perhitungan Manual Skenario 1 Paket Data 512 Byte

Percobaan	Paket Data 512 Byte			
	Packet Loss (%)	**Troughput (Bps)**	**Delay (ms)**	**Jitter (ms)**
	Rumus Perhitungan : Persamaan 5.1	**Rumus Perhitungan : Persamaan 5.2**	**Rumus Perhitungan : Persamaan 5.3**	**Rumus Perhitungan : Persamaan 5.4**
1	0	322621,2980	1.5870	0,0000
2	0	349726,7760	1.4640	0,1230
3	0	364672,3647	1.4040	0,0600
4	0	215126,0504	2.3800	0,9760
5	0	349488,0546	1.4650	0,9150
6	0	364672,3647	1.4040	0,0610
7	0	357043,2357	1.4340	0,0300
8	0	356794,4251	1.4350	0,0010
9	0	357043,2357	1.4340	0,0010
10	0	390243,9024	1.3120	0,1220
Rata-Rata	0	342743,1707	1,5319	0,2289

Berdasarkan tabel 5.6 pada pengiriman paket data sebesar 512 *byte* di skenario satu didapatkan nilai *packet loss* terbesar dan terkecil 0 % yang menunjukan bahwa dari sepuluh kali percobaan paket data berhasil terkirim seluruhnya. Untuk nilai troughput terbesar adalah 390243,90 *Bps* dan nilai troughput terkecil adalah 215126,05 *Bps*, nilai delay terbesar adalah 2,3800 *ms* dan nilai delay terkecil adalah 1,3120 *ms*, nilai *jitter* terbesar dalah 0,9760 dan nilai *jitter* terkecil adalah 0,0010 *ms*. Tabel 5.7 merupakan proses perhitungan secara manual untuk mendapatkan nilai dari parameter *packet loss*, *troughput*, *delay* dan *jitter* pada skenario satu, dengan besaran paket data 512 *byte*.

5.1.1.4 Paket Data 1024 Byte

Tabel 5.8 Hasil Pengujian 1024 Byte

Percobaan	Paket Data 1024 Byte			
	Packet Loss (%)	Trough put (Bps)	Delay (ms)	Jitter (ms)
1	0	493493,98	2,0750	0,0000
2	0	435744,68	2,3500	0,2750
3	0	381378,03	2,6850	0,3350
4	0	532501,30	1,9230	0,7620
5	0	549946,29	1,8620	0,0610
6	0	541226,22	1,8920	0,0300
7	0	568573,01	1,8010	0,0910
8	0	568573,01	1,8010	0,0000
9	0	568573,01	1,8010	0,0000
10	0	559257,24	1,8310	0,0300
Rata-rata	0	519926,68	2,0021	0,1584

Manual Perhitungan :

Tabel 5.9 Perhitungan Manual Skenario 1 Paket Data 1024 Byte

Percobaan	Paket Data 1024 Byte			
	Packet Loss (%)	Troughput (Bps)	Delay (ms)	Jitter (ms)
	Rumus Perhitungan : Persamaan 5.1	Rumus Perhitungan : Persamaan 5.2	Rumus Perhitungan : Persamaan 5.3	Rumus Perhitungan : Persamaan 5.4
1	0	493493,9759	2,0750	0,0000
2	0	435744,6809	2,3500	0,2750
3	0	381378,0261	2,6850	0,3350
4	0	532501,3001	1,9230	0,7620
5	0	549946,2943	1,8620	0,0610
6	0	541226,2156	1,8920	0,0300
7	0	568573,0150	1,8010	0,0910
8	0	568573,0150	1,8010	0,0000
9	0	568573,0150	1,8010	0,0000
10	0	559257,2365	1,8310	0,0300

Rata-Rata	0	519926,6774	2,0021	0,1584

Berdasarkan tabel 5.8 pada pengiriman paket data sebesar 1024 *byte* di skenario satu didapatkan nilai *packet loss* terbesar dan terkecil 0 % yang menunjukan bahwa dari sepuluh kali percobaan paket data berhasil terkirim seluruhnya. Untuk nilai *troughput* terbesar adalah 568573,01 *Bps* dan nilai *troughput* terkecil adalah 381378,03 *Bps*, nilai delay terbesar adalah 2,6850 *ms* dan nilai *delay* terkecil adalah 1,8010 *ms*, nilai *jitter* terbesar dalah 0,7620 dan nilai *jitter* terkecil adalah 0 *ms*. Tabel 5.9 merupakan proses perhitungan secara manual untuk mendapatkan nilai dari parameter *packet loss*, *troughput*, *delay* dan *jitter* pada skenario satu, dengan besaran paket data 1024 *byte*.

5.1.1.5 Paket Data 1500 Byte

Tabel 5.10 Hasil Pengujian 1500 Byte

Percobaan	Paket Data 1500 Byte			
	Packet Loss (%)	Trough put (Bps)	Delay (ms)	Jitter (ms)
1	0	599520,38	2,5020	0,0000
2	0	655307,99	2,2890	0,2130
3	0	592183,18	2,5330	0,2440
4	0	673249,55	2,2280	0,3050
5	0	664304,69	2,2580	0,0300
6	0	702247,19	2,1360	0,1220
7	0	692201,20	2,1670	0,0310
8	0	655594,41	2,2880	0,1210
9	0	655307,99	2,2890	0,0010
10	0	692201,20	2,1670	0,1220
Rata-rata	0	658211,78	2,2857	0,1189

Manual Perhitungan :

Tabel 5.11 Perhitungan Manual Skenario 1 Paket Data 1500 Byte

Percobaan	Paket Data 1500 Byte			
	Packet Loss (%)	**Troughput (Bps)**	**Delay (ms)**	**Jitter (ms)**
	Rumus Perhitungan : Persamaan 5.1	**Rumus Perhitungan : Persamaan 5.2**	**Rumus Perhitungan : Persamaan 5.3**	**Rumus Perhitungan : Persamaan 5.4**
1	0	599520,3837	2,5020	0,0000
2	0	655307,9948	2,2890	0,2130
3	0	592183,1820	2,5330	0,2440
4	0	673249,5512	2,2280	0,3050
5	0	664304,6944	2,2580	0,0300
6	0	702247,1910	2,1360	0,1220
7	0	692201,1998	2,1670	0,0310
8	0	655594,4056	2,2880	0,1210
9	0	655307,9948	2,2890	0,0010
10	0	692201,1998	2,1670	0,1220
Rata-Rata	0	658211,7797	2,2857	0,1189

Berdasarkan tabel 5.10 pada pengiriman paket data sebesar 1500 *byte* di skenario satu didapatkan nilai nilai *packet loss* terbesar dan terkecil 0 % yang menunjukan bahwa dari sepuluh kali percobaan paket data berhasil terkirim seluruhnya. Untuk nilai *troughput* terbesar adalah 702247,19 *Bps* dan nilai *troughput* terkecil adalah 592183,18 *Bps*, nilai *delay* terbesar adalah 2,5330 *ms* dan nilai *delay* terkecil adalah 2,1360 *ms*, nilai *jitter* terbesar dalah 0,3050 dan nilai *jitter* terkecil adalah 0,0010 *ms*. Tabel 5.11 merupakan proses perhitungan secara manual untuk mendapatkan nilai dari parameter *packet loss*, *troughput*, *delay* dan *jitter* pada skenario satu, dengan besaran paket data 1500 *byte*.

5.1.2 Skenario Dua

Pada pengujian skenario dua, percobaan yang dilakukan adalah mengirimkan paket data dari gedung B menuju gedung C dengan *client* berada pada sisi gedung B dan *server* berada pada sisi gedung C. Berikut konfigurasi jaringan pada skenario dua :

Tabel 5.12 Konfigurasi jaringan skenario dua

Status	Alamat IP	Port
Client	172.21.3.46	2211
Server	172.21.4.20	2211

Tabel 5.12 menunjukan konfigurasi jaringan pada skenario dua, dengan konfigurasi alamat *IP client* 172.21.3.46 dan *server* 172.21.4.20 beserta *port* yang digunakan adalah 2211. Tujuan pengujian adalah untuk mengetahui besaran nilai parameter *packet loss*, *troughtput*, *delay* dan *jitter* pada jaringan *local area network* antara gedung B dan gedung C. Berikut tampilan dari hasil pengujian dengan pengiriman paket data sebesar 128 *byte*, 256 *byte*, 512 *byte*, 1024 *byte*, 1024 *byte* dan 1500 *byte* dengan sepuluh kali percobaan pengiriman paket data dan batasan waktu sebesar satu detik pada setiap percobaan:

5.1.2.1 Paket Data 128 Byte

Tabel 5.13 Hasil Pengujian 128 Byte

Percobaan	Paket Data 128 Byte			
	Packet Loss (%)	Trough put (Bps)	Delay (ms)	Jitter (ms)
1	0	91168,09	1,4040	0,0000
2	0	102318,15	1,2510	0,1530
3	0	97560,98	1,3120	0,0610
4	0	104832,1	1,2210	0,0910
5	0	91168,09	1,4040	0,1830
6	0	80655,32	1,5870	0,1830
7	0	93158,66	1,3740	0,2130
8	0	99843,99	1,2820	0,0920
9	0	107563,03	1,1900	0,0920
10	0	99843,99	1,2820	0,0920
Rata-	0	96811,24	1,3307	0,1160

rata				

Manual Perhitungan :

Tabel 5.14 Perhitungan Manual Skenario 2 Paket Data 128 *Byte*

Percobaan	Paket Data 128 Byte			
	Packet Loss (%)	**Troughput (Bps)**	**Delay (ms)**	**Jitter (ms)**
	Rumus Perhitungan : Persamaan 5.1	**Rumus Perhitungan : Persamaan 5.2**	**Rumus Perhitungan : Persamaan 5.3**	**Rumus Perhitungan : Persamaan 5.4**
1	0	91168,0912	1,4040	0,0000
2	0	102318,1455	1,2510	0,1530
3	0	97560,9756	1,3120	0,0610
4	0	104832,1048	1,2210	0,0910
5	0	91168,0912	1,4040	0,1830
6	0	80655,3245	1,5870	0,1830
7	0	93158,6608	1,3740	0,2130
8	0	99843,9938	1,2820	0,0920
9	0	107563,0252	1,1900	0,0920
10	0	99843,9938	1,2820	0,0920
Rata-Rata	0	96811,2406	1,3307	0,1160

Berdasarkan tabel 5.13 pada pengiriman paket data sebesar 128 *byte* di skenario dua didapatkan nilai *packet loss* terbesar dan terkecil 0 % yang menunjukan bahwa dari sepuluh kali percobaan paket data berhasil terkirim seluruhnya. Untuk nilai troughput terbesar adalah 107563,03 *Bps* dan nilai *troughput* terkecil adalah 80655,32 *Bps*, nilai *delay* terbesar adalah 1,4040 *ms* dan nilai *delay* terkecil adalah 1,1900 *ms*, nilai *jitter* terbesar dalah 0,2130 dan nilai *jitter* terkecil adalah 0,0610 *ms*. Tabel 5.14 merupakan proses perhitungan secara manual untuk mendapatkan nilai dari parameter *packet loss*, *troughput*, *delay* dan *jitter* pada skenario dua, dengan besaran paket data 128 *byte*.

5.1.2.2 Paket Data 256 Byte

Tabel 5.15 Hasil Pengujian 256 Byte

Percobaan	Paket Data 256 *Byte*			
	Packet Loss (%)	Trough put (*Bps*)	Delay (*ms*)	*Jitter* (*ms*)
1	0	158317,87	1,6170	0,0000
2	0	186453,02	1,3730	0,2440
3	0	186317,32	1,3740	0,0010
4	0	113374,67	2,2580	0,8840
5	0	178521,62	1,4340	0,8240
6	0	182336,18	1,4040	0,0300
7	0	186453,02	1,3730	0,0310
8	0	195121,95	1,3120	0,0610
9	0	178397,21	1,4350	0,1230
10	0	178521,62	1,4340	0,0010
Rata-rata	0	174381,45	1,5014	0,2199

Manual Perhitungan :

Gambar 5.16 Perhitungan Manual Skenario 2 Paket Data 256 *Byte*

Percobaan	Paket Data 256 Byte			
	Packet Loss (%)	Troughput (Bps)	Delay (ms)	Jitter (ms)
	Rumus Perhitungan : Persamaan 5.1	Rumus Perhitungan : Persamaan 5.2	Rumus Perhitungan : Persamaan 5.3	Rumus Perhitungan : Persamaan 5.4
1	0	158317,8726	1,6170	0,0000
2	0	186453,0226	1,3730	0,2440
3	0	186317,3217	1,3740	0,0010
4	0	113374,6678	2,2580	0,8840
5	0	178521,6179	1,4340	0,8240
6	0	182336,1823	1,4040	0,0300
7	0	186453,0226	1,3730	0,0310
8	0	195121,9512	1,3120	0,0610
9	0	178397,2125	1,4350	0,1230
10	0	178521,6179	1,4340	0,0010

Rata-Rata	0	174381,4489	1,5014	0,2199

Berdasarkan tabel 5.15 pada pengiriman paket data sebesar 256 *byte* di skenario dua didapatkan nilai *packet loss* terbesar dan terkecil 0 % yang menunjukan bahwa dari sepuluh kali percobaan paket data berhasil terkirim seluruhnya. Untuk nilai *troughput* terbesar adalah 195121,95 *Bps* dan nilai troughput terkecil adalah 113374,67 *Bps*, nilai delay terbesar adalah 2,2580 *ms* dan nilai delay terkecil adalah 1,3120 *ms*, nilai *jitter* terbesar dalah 0,8840 dan nilai *jitter* terkecil adalah 0,0010 *ms*. Tabel 5.16 merupakan proses perhitungan secara manual untuk mendapatkan nilai dari parameter *packet loss, troughput, delay* dan *jitter* pada skenario dua, dengan besaran paket data 256 *byte*.

5.1.2.3 Paket Data 512 Byte

Tabel 5.17 Hasil Pengujian 512 Byte

Percobaan	Paket Data 512 *Byte*			
	Packet Loss (%)	Trough put (*Bps*)	Delay (*ms*)	*Jitter* (*ms*)
1	0	289265,54	1,7700	0,0000
2	0	322621,30	1,5870	0,1830
3	0	316635,75	1,6170	0,0300
4	0	250489,24	2,0440	0,4270
5	0	310679,61	1,6480	0,3960
6	0	335517,69	1,5260	0,1220
7	0	335517,69	1,5260	0,0000
8	0	357043,24	1,4340	0,0920
9	0	329048,84	1,5560	0,1220
10	0	329048,84	1,5560	0,0000
Rata-rata	0	317586,77	1,6264	0,1372

Manual Perhitungan :

Gambar 5.18 Perhitungan Manual Skenario 2 Paket Data 512 Byte

Percobaan	Paket Data 512 Byte			
	Packet Loss (%)	**Troughput (Bps)**	**Delay (ms)**	**Jitter (ms)**
	Rumus Perhitungan : Persamaan 5.1	**Rumus Perhitungan : Persamaan 5.2**	**Rumus Perhitungan : Persamaan 5.3**	**Rumus Perhitungan : Persamaan 5.4**
1	0	289265,5367	1,7700	0,0000
2	0	322621,2980	1,5870	0,1830
3	0	316635,7452	1,6170	0,0300
4	0	250489,2368	2,0440	0,4270
5	0	310679,6117	1,6480	0,3960
6	0	335517,6933	1,5260	0,1220
7	0	335517,6933	1,5260	0,0000
8	0	357043,2357	1,4340	0,0920
9	0	329048,8432	1,5560	0,1220
10	0	329048,8432	1,5560	0,0000
Rata-Rata	0	317586,7737	1,6264	0,1372

Berdasarkan tabel 5.17 pada pengiriman paket data sebesar 512 *byte* di skenario satu didapatkan nilai nilai *packet loss* terbesar dan terkecil 0 % yang menunjukan bahwa dari sepuluh kali percobaan paket data berhasil terkirim seluruhnya. Untuk nilai *troughput* terbesar adalah 357043,24 *Bps* dan nilai troughput terkecil adalah 250489,24 *Bps*, nilai *delay* terbesar 2,0440 *ms* dan nilai delay terkecil adalah 1,4340 *ms*, nilai *jitter* terbesar dalah 0,4270 *ms* dan nilai *jitter* terkecil adalah 0 *ms*. Tabel 5.18 merupakan proses perhitungan secara manual untuk mendapatkan nilai dari parameter, *packet loss*, *troughput*, *delay* dan *jitter* pada skenario dua, dengan besaran paket data 512 *byte*.

5.1.2.4 Paket Data 1024 Byte

Tabel 5.19 Hasil Pengujian 1024 Byte

Percobaan	Paket Data 1024 *Byte*			
	Packet Loss (%)	Trough put (*Bps*)	Delay (*ms*)	*Jitter* (*ms*)
1	0	466090,12	2,1970	0,0000
2	0	524321,56	1,9530	0,2440
3	0	364802,28	2,8070	0,8540
4	0	524321,56	1,9530	0,8540
5	0	516389,31	1,9830	0,0300
6	0	516129,03	1,9840	0,0010
7	0	550241,81	1,8610	0,1230
8	0	532501,30	1,9230	0,0620
9	0	559257,24	1,8310	0,0920
10	0	549946,29	1,8620	0,0310
Rata-rata	0	510400,05	2,0354	0,2291

Manual Perhitungan :

5.20 Perhitungan Manual Skenario 2 Paket Data 1024 Byte

Percobaan	Paket Data 1024 Byte			
	Packet Loss (%)	Troughput (Bps)	Delay (ms)	Jitter (ms)
	Rumus Perhitungan : Persamaan 5.1	Rumus Perhitungan : Persamaan 5.2	Rumus Perhitungan : Persamaan 5.3	Rumus Perhitungan : Persamaan 5.4
1	0	466090,1229	2,1970	0,0000
2	0	524321,5566	1,9530	0,2440
3	0	364802,2800	2,8070	0,8540
4	0	524321,5566	1,9530	0,8540
5	0	516389,3091	1,9830	0,0300
6	0	516129,0323	1,9840	0,0010
7	0	550241,8055	1,8610	0,1230
8	0	532501,3001	1,9230	0,0620
9	0	559257,2365	1,8310	0,0920
10	0	549946,2943	1,8620	0,0310

Rata-Rata	0	510400,0494	2,0354	0,2291

Berdasarkan tabel 5.19 pada pengiriman paket data sebesar 1024 *byte* di skenario dua didapatkan nilai *packet loss* terbesar dan terkecil 0 % yang menunjukan bahwa dari sepuluh kali percobaan paket data berhasil terkirim seluruhnya. Untuk nilai *troughput* terbesar adalah 559257,25 *Bps* dan nilai troughput terkecil adalah 364802,28 *Bps*, nilai *delay* terbesar adalah 2,8070 *ms* dan nilai delay terkecil adalah 1,8310 *ms*, nilai *jitter* terbesar 0,8540 dan nilai *jitter* terkecil adalah 0,0010 *ms*. Tabel 5.20 merupakan proses perhitungan secara manual untuk mendapatkan nilai dari parameter *packet loss*, *troughput*, *delay* dan *jitter* pada skenario dua, dengan besaran paket data 1024 *byte*.

5.1.2.5 Paket Data 1500 Byte

Tabel 5.21 Hasil Pengujian 1500 Byte

Percobaan	Paket Data 1500 Byte			
	Packet Loss (%)	Trough put (Bps)	Delay (ms)	Jitter (ms)
1	0	599520,38	2,5020	0,0000
2	0	655307,99	2,2890	0,2130
3	0	646830,53	2,3190	0,0300
4	0	655307,99	2,2890	0,0300
5	0	712589,07	2,1050	0,1840
6	0	733496,33	2,0450	0,0600
7	0	655307,99	2,2890	0,2440
8	0	664010,62	2,2590	0,0300
9	0	655307,99	2,2890	0,0300
10	0	673551,86	2,2270	0,0620
Rata-rata	0	665123,08	2,2613	0,0883

Manual Perhitungan :

Tabel 5.22 Perhitungan Manual Skenario 2 Paket Data 1500 Byte

Percobaan	Paket Data 1500 Byte			
	Packet Loss (%)	**Troughput (Bps)**	**Delay (ms)**	**Jitter (ms)**
	Rumus Perhitungan : Persamaan 5.1	**Rumus Perhitungan : Persamaan 5.2**	**Rumus Perhitungan : Persamaan 5.3**	**Rumus Perhitungan : Persamaan 5.4**
1	0	599520,3837	2,5020	0,0000
2	0	655307,9948	2,2890	0,2130
3	0	646830,5304	2,3190	0,0300
4	0	655307,9948	2,2890	0,0300
5	0	712589,0736	2,1050	0,1840
6	0	733496,3325	2,0450	0,0600
7	0	655307,9948	2,2890	0,2440
8	0	664010,6242	2,2590	0,0300
9	0	655307,9948	2,2890	0,0300
10	0	673551,8635	2,2270	0,0620
Rata-Rata	0	665123,0787	2,2613	0,0883

Berdasarkan tabel 5.21 pada pengiriman paket data sebesar 1500 *byte* di skenario satu didapatkan nilai *packet loss* terbesar dan terkecil 0 % yang menunjukan bahwa dari sepuluh kali percobaan paket data berhasil terkirim seluruhnya. Untuk nilai *troughput* terbesar adalah 733496,33 *Bps* dan nilai troughput terkecil adalah 599520,38 *Bps*, nilai *delay* terbesar adalah 2,502 *ms* dan nilai delay terkecil adalah 2,0450 *ms*, nilai *jitter* terbesar dalah 0,2440 dan nilai *jitter* terkecil adalah 0,0300 *ms*. Tabel 5.22 merupakan proses perhitungan secara manual untuk mendapatkan nilai dari parameter *packet loss*, *troughput*, *delay* dan *jitter* pada skenario dua, dengan besaran paket data 1500 *byte*.

5.1.3 Skenario Tiga

Pada pengujian skenario tiga, percobaan yang dilakukan adalah mengirimkan paket data dari gedung C menuju gedung A dengan *client* berada pada sisi gedung C dan *server* berada pada sisi gedung A. Berikut konfigurasi jaringan pada skenario tiga :

Tabel 5.23 Konfigurasi jaringan skenario Tiga

Status	Alamat IP	Port
Client	172.21.4.8	2211
Server	172.21.1.44	2211

Tujuan pengujian adalah untuk mengetahui besaran nilai parameter *packet loss, troughtput, delay* dan *jitter* pada jaringan *local area network* antara gedung B dan gedung C. Berikut tampilan dari hasil pengujian dengan pengiriman paket data sebesar 128 *byte*, 256 *byte*, 512 *byte*, 1024 *byte*, 1024 *byte* dan 1500 *byte* dengan sepuluh kali percobaan pengiriman paket data dan batasan waktu sebesar satu detik pada setiap percobaan:

5.1.3.1 Paket Data 128 Byte

Tabel 5.24 Hasil Pengujian 128 Byte

Percobaan	Paket Data 128 *Byte*			
	Packet Loss (%)	Trough put (*Bps*)	Delay (*ms*)	*Jitter* (*ms*)
1	0	91233,07	1,4030	0,0000
2	0	102318,15	1,2510	0,1520
3	0	102318,15	1,2510	0,0000
4	0	99921,94	1,2810	0,0300
5	0	83879,42	1,5260	0,2450
6	0	99921,94	1,2810	0,2450
7	0	99921,94	1,2810	0,0000
8	0	107563,03	1,1900	0,0910
9	0	102318,15	1,2510	0,0610
10	0	102318,15	1,2510	0,0000
Rata-rata	0	99171,39	1,2966	0,0824

Manual Perhitungan :

Tabel 5.25 Perhitungan Manual Skenario 3 Paket Data 128 *Byte*

Percobaan	Paket Data 128 Byte			
	Packet Loss (%)	**Troughput (Bps)**	**Delay (ms)**	**Jitter (ms)**
	Rumus Perhitungan : Persamaan 5.1	**Rumus Perhitungan : Persamaan 5.2**	**Rumus Perhitungan : Persamaan 5.3**	**Rumus Perhitungan : Persamaan 5.4**
1	0	91233,0720	1,4030	0,0000
2	0	102318,1455	1,2510	0,1520
3	0	102318,1455	1,2510	0,0000
4	0	99921,9360	1,2810	0,0300
5	0	83879,4233	1,5260	0,2450
6	0	99921,9360	1,2810	0,2450
7	0	99921,9360	1,2810	0,0000
8	0	107563,0252	1,1900	0,0910
9	0	102318,1455	1,2510	0,0610
10	0	102318,1455	1,2510	0,0000
Rata-Rata	0	99171,3910	1,2966	0,0824

Berdasarkan tabel 5.24 pada pengiriman paket data sebesar 128 *byte* di skenario tiga didapatkan nilai *packet loss* terbesar dan terkecil 0 % yang menunjukan bahwa dari sepuluh kali percobaan paket data berhasil terkirim seluruhnya. Untuk nilai troughput terbesar adalah 107563,03 *Bps* dan nilai troughput terkecil adalah 83879,42 *Bps*, nilai delay terbesar adalah 1,4030 *ms* dan nilai delay terkecil adalah 1,2510 *ms*, nilai *jitter* terbesar dalah 0,0910 dan nilai *jitter* terkecil adalah 0 *ms*. Tabel 5.25 merupakan proses perhitungan secara manual untuk mendapatkan nilai dari parameter *packet loss*, *troughput*, *delay* dan *jitter* pada skenario tiga, dengan besaran paket data 128 *byte*.

5.1.3.2 Paket Data 256 Byte

Tabel 5.26 Hasil Pengujian 256 Byte

Percobaan	Paket Data 256 Byte			
	Packet Loss (%)	Trough put (Bps)	Delay (ms)	Jitter (ms)
1	0	171122,99	1,4960	0,0000
2	0	186453,02	1,3730	0,1230
3	0	186453,02	1,3730	0,0000
4	0	129032,26	1,9840	0,6110
5	0	174744,03	1,4650	0,5190
6	0	186453,02	1,3730	0,0920
7	0	199687,99	1,2820	0,0910
8	0	195121,95	1,3120	0,0300
9	0	178521,62	1,4340	0,1220
10	0	199687,99	1,2820	0,1520
Rata-rata	0	180727,79	1,4374	0,1740

Manual Perhitungan :

Tabel 5.27 Perhitungan Manual Skenario 3 Paket Data 256 *Byte*

Percobaan	Paket Data 256 Byte			
	Packet Loss (%)	Troughput (Bps)	Delay (ms)	Jitter (ms)
	Rumus Perhitungan : Persamaan 5.1	Rumus Perhitungan : Persamaan 5.2	Rumus Perhitungan : Persamaan 5.3	Rumus Perhitungan : Persamaan 5.4
1	0	171122,9947	1,4960	0,0000
2	0	186453,0226	1,3730	0,1230
3	0	186453,0226	1,3730	0,0000
4	0	129032,2581	1,9840	0,6110
5	0	174744,0273	1,4650	0,5190
6	0	186453,0226	1,3730	0,0920
7	0	199687,9875	1,2820	0,0910
8	0	195121,9512	1,3120	0,0300
9	0	178521,6179	1,4340	0,1220
10	0	199687,9875	1,2820	0,1520

Rata-Rata	0	180727,7892	1,4374	0,1740

Berdasarkan tabel 5.26 pada pengiriman paket data sebesar 256 *byte* di skenario satu didapatkan nilai *packet loss* terbesar dan terkecil 0 % yang menunjukan bahwa dari sepuluh kali percobaan paket data berhasil terkirim seluruhnya. Untuk nilai troughput terbesar adalah 199687,99 *Bps* dan nilai troughput terkecil adalah 129032,26 *Bps*, nilai *delay* terbesar adalah 1,9840 *ms* dan nilai delay terkecil adalah 1,282 *ms*, nilai *jitter* terbesar dalah 0,6110 dan nilai *jitter* terkecil adalah 0 *ms*. Tabel 5.27 merupakan proses perhitungan secara manual untuk mendapatkan nilai dari parameter *packet loss*, *troughput*, *delay* dan *jitter* pada skenario tiga, dengan besaran paket data 256 *byte*.

5.1.3.3 Paket Data 512 Byte

Tabel 5.28 Hasil Pengujian 512 Byte

Percobaan	Paket Data 512 *Byte*			
	Packet Loss (%)	Trough put (*Bps*)	Delay (*ms*)	*Jitter* (*ms*)
1	0	310679,61	1,6480	0,0000
2	0	349488,05	1,4650	0,1830
3	0	335517,69	1,5260	0,0610
4	0	270613,11	1,8920	0,3660
5	0	335517,69	1,5260	0,3660
6	0	342245,99	1,4960	0,0300
7	0	335517,69	1,5260	0,0300
8	0	329048,84	1,5560	0,0300
9	0	356794,43	1,4350	0,1210
10	0	357043,24	1,4340	0,0010
Rata-rata	0	332246,63	1,5504	0,1188

Manual Perhitungan :

Tabel 5.29 Perhitungan Manual Skenario 3 Paket Data 512 Byte

Percobaan	Paket Data 512 Byte			
	Packet Loss (%)	**Troughput (Bps)**	**Delay (ms)**	**Jitter (ms)**
	Rumus Perhitungan : Persamaan 5.1	**Rumus Perhitungan : Persamaan 5.2**	**Rumus Perhitungan : Persamaan 5.3**	**Rumus Perhitungan : Persamaan 5.4**
1	0	310679,6117	1,6480	0,0000
2	0	349488,0546	1,4650	0,1830
3	0	335517,6933	1,5260	0,0610
4	0	270613,1078	1,8920	0,3660
5	0	335517,6933	1,5260	0,3660
6	0	342245,9893	1,4960	0,0300
7	0	335517,6933	1,5260	0,0300
8	0	329048,8432	1,5560	0,0300
9	0	356794,4251	1,4350	0,1210
10	0	357043,2357	1,4340	0,0010
Rata-Rata	0	332246.6347	1,5504	0,1188

Berdasarkan tabel 5.28 pada pengiriman paket data sebesar 512 *byte* di skenario tiga didapatkan nilai *packet loss* terbesar dan terkecil 0 % yang menunjukan bahwa dari sepuluh kali percobaan paket data berhasil terkirim seluruhnya. Untuk nilai troughput terbesar adalah 357043,24 *Bps* dan nilai troughput terkecil adalah 270613,11 *Bps*, nilai delay terbesar adalah 1,8920 *ms* dan nilai delay terkecil adalah 1,434 *ms*, nilai *jitter* terbesar dalah 0,3660 dan nilai *jitter* terkecil adalah 0,0010 *ms*. Tabel 5.29 merupakan proses perhitungan secara manual untuk mendapatkan nilai dari parameter *packet loss*, *troughput*, *delay* dan *jitter* pada skenario tiga, dengan besaran paket data 512 *byte*.

5.1.3.4 Paket Data 1024 Byte

Tabel 5.30 Hasil Pengujian 1024 Byte

Percobaan	Paket Data 1024 *Byte*			
	Packet Loss (%)	Trough put (*Bps*)	Delay (*ms*)	*Jitter* (*ms*)
1	0	493493,98	2,0750	0,0000
2	0	486229,82	2,1060	0,0310
3	0	424720,03	2,4110	0,3050
4	0	549946,29	1,8620	0,5490
5	0	541226,29	1,8920	0,0300
6	0	541226,22	1,8010	0,0910
7	0	568573,01	1,8010	0,0000
8	0	568573,01	1,8010	0,0000
9	0	568888,89	1,8000	0,0010
10	0	559257,24	1,8310	0,0310
Rata-rata	0	530213,48	1,9380	0,1038

Manual Perhitungan :

Tabel 5.31 Perhitungan Manual Skenario 3 Paket Data 1024 Byte

Percobaan	Paket Data 1024 Byte			
	Packet Loss (%)	Troughput (Bps)	Delay (ms)	Jitter (ms)
	Rumus Perhitungan : Persamaan 5.1	Rumus Perhitungan : Persamaan 5.2	Rumus Perhitungan : Persamaan 5.3	Rumus Perhitungan : Persamaan 5.4
1	0	493493,9759	2,0750	0,0000
2	0	486229,8196	2,1060	0,0310
3	0	424720,0332	2,4110	0,3050
4	0	549946,2943	1,8620	0,5490
5	0	541226,2156	1,8920	0,0300
6	0	568573,0150	1,8010	0,0910
7	0	568573,0150	1,8010	0,0000
8	0	568573,0150	1,8010	0,0000
9	0	568888,8889	1,8000	0,0010
10	0	559257,2365	1,8310	0,0310

Rata-Rata	0	532948,1509	1,9380	0,1038

Berdasarkan tabel 5.30 pada pengiriman paket data sebesar 1024 *byte* di skenario tiga didapatkan nilai *packet loss* terbesar dan terkecil 0 % yang menunjukan bahwa dari sepuluh kali percobaan paket data berhasil terkirim seluruhnya. Untuk nilai *troughput* terbesar adalah 568888,89 *Bps* dan nilai troughput terkecil adalah 424720,03 *Bps*, nilai *delay* terbesar adalah 2,4110 *ms* dan nilai delay terkecil adalah 1,800 *ms*, nilai *jitter* terbesar dalah 0,3050 dan nilai *jitter* terkecil adalah 0,0010 *ms*. Tabel 5.31 merupakan proses perhitungan secara manual untuk mendapatkan nilai dari parameter *packet loss*, *troughput*, *delay* dan *jitter* pada skenario tiga, dengan besaran paket data 1024 *byte*.

5.1.3.5 Paket Data 1500 Byte

Tabel 5.32 Hasil Pengujian 1500 Byte

Percobaan	Paket Data 1500 Byte			
	Packet Loss (%)	Trough put (Bps)	Delay (ms)	Jitter (ms)
1	0	614502,25	2,4410	0,0000
2	0	673249,55	2,2280	0,2130
3	0	655594,41	2,2880	0,0600
4	0	622148,49	2,4110	0,1230
5	0	692201,20	2,1670	0,2440
6	0	664304,69	2,2580	0,0910
7	0	692520,78	2,1660	0,0920
8	0	702247,19	2,1360	0,0300
9	0	692520,78	2,1660	0,0300
10	0	692201,20	2,1670	0,0010
Rata-rata	0	670149,05	2,2428	0,0884

Manual Perhitungan :

Tabel 5.33 Perhitungan Manual Skenario 3 Paket Data 1500 Byte

Percobaan	Paket Data 128 Byte			
	Packet Loss (%)	**Troughput (Bps)**	**Delay (ms)**	**Jitter (ms)**
	Rumus Perhitungan : Persamaan 5.1	**Rumus Perhitungan : Persamaan 5.2**	**Rumus Perhitungan : Persamaan 5.3**	**Rumus Perhitungan : Persamaan 5.4**
1	0	614502,2532	2,4410	0,0000
2	0	673249,5512	2,2280	0,2130
3	0	655594,4056	2,2880	0,0600
4	0	622148,4861	2,4110	0,1230
5	0	692201,1998	2,1670	0,2440
6	0	664304,6944	2,2580	0,0910
7	0	692520,7756	2,1660	0,0920
8	0	702247,191	2,1360	0,0300
9	0	692520,7756	2,1660	0,0300
10	0	692201,1998	2,1670	0,0010
Rata-Rata	0	670149,0532	2,2428	0,0884

Berdasarkan tabel 5.32 pada pengiriman paket data sebesar 1500 *byte* di skenario tiga didapatkan nilai *packet loss* terbesar dan terkecil 0 % yang menunjukan bahwa dari sepuluh kali percobaan paket data berhasil terkirim seluruhnya. Untuk nilai *troughput* terbesar adalah 702247,19 *Bps* dan nilai troughput terkecil adalah 614502,25 *Bps*, nilai *delay* terbesar adalah 2,4410 *ms* dan nilai delay terkecil adalah 2,1360 *ms*, nilai *jitter* terbesar dalah 0,2440 dan nilai *jitter* terkecil adalah 0,0010 *ms*. Tabel 5.33 merupakan proses perhitungan secara manual untuk mendapatkan nilai dari parameter *packet loss*, *troughput*, *delay* dan *jitter* pada skenario tiga, dengan besaran paket data 1500 *byte*.

5.2 Analisis

Pada bagian analisis, akan dilakukan komparasi atau perbandingan dari nilai setiap percobaan yang dilakukan dan nilai rata-rata parameter *packet loss*, *packet loss*, *delay* dan *jitter* yang didapat pada pengujian di skenario satu, skenario dua dan skenario tiga yang telah dilakukan. Hal ini bertujuan untuk mengetahui bagaimana kondisi kualitas jaringan pada jaringan di skenario satu yaitu kondisi jaringan antara gedung A dan gedung B, skenario dua yaitu kondisi jaringan antara gedung B dan gedung C dan skenario tiga yaitu kondisi jaringan antara gedeung C dan gedung A. Hal ini bertujuan untuk dapat mengetahui antara jaringan di skenario satu, skenario dua dan skenario tiga manakah yang memiliki kondisi kualitas jaringan terbaik.

5.2.1 Komparasi Nilai Setiap Percobaan

Pada sub-bab komparasi nilai setiap percobaan akan dilakukan perbandingan untuk melihat hasil yang didapat pada hasil setiap percobaan yang dilakukan sebanyak sepuluh kali, dengan pengiriman paket data 128 *byte*, 256 *byte*, 512 *byte*, 1024 *byte* dan 1500 *byte* yang didapat dari pengujian yang telah dilakukan pada sekenario satu yaitu pengujian kondisi jaringan antara gedung A dan gedung B, skenario dua yaitu pengujian kondisi jaringan antara gedung B dan gedung C dan skenario tiga yaitu pengujian kondisi jaringan antara gedung C dan gedung A.

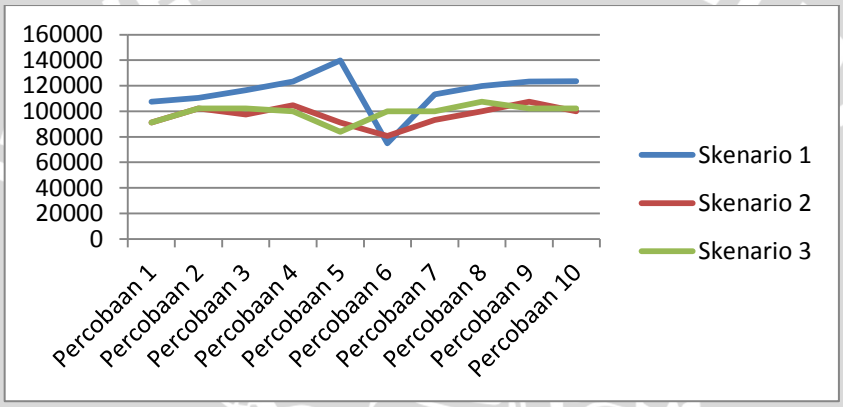
5.2.1.1 Komparasi Nilai Packet Loss

Dari hasil percobaan pengiriman paket data sebesar 128 *byte*, 256 *byte*, 512 *byte*, 1024 *byte* dan 1500 *byte* yang dilakukan sebanyak sepuluh kali pada masing-masing paket data yang dikirimkan tersebut, dan dilakukan pada pengujian sekenario satu yaitu pengujian kondisi jaringan antara gedung A dan gedung B, skenario dua yaitu pengujian kondisi jaringan antara gedung B dan gedung C dan skenario tiga yaitu pengujian kondisi jaringan antara gedung C dan gedung A didapatkan hasil yang sama yaitu sebesar 0 % untuk seluruh percobaan yang dilakukan, hal ini disebabkan pada pengujian ini menggunakan *protocol TCP* yang bersifat *connection oriented* dan bersifat *reliable* yang artinya adalah terjadi proses *handshaking* antara *client* dan *server* dan menerapkan deteksi

kesalahan pada saat pengiriman paket data sehingga paket data dapat dikirimkan dengan baik tanpa terjadi hilangnya paket data pada proses pengiriman.

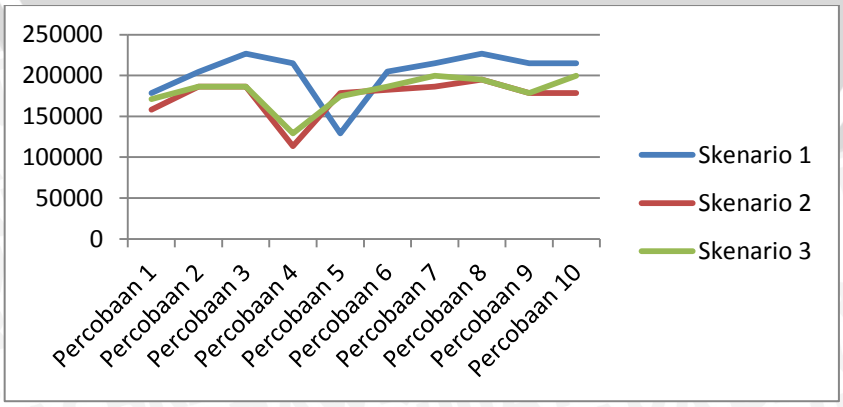
5.2.1.2 Komparasi Nilai Troughput

Pada komparasi nilai *troughput* akan dibandingkan nilai yang didapat dari setiap percobaan pengiriman paket data yaitu sebesar 128 *byte*, 256 *byte*, 512 *byte*, 1024 *byte* dan 1500 *byte* pada pengujian di skenario satu, skenario dua dan skenario tiga dan dicari nilai *troughput* terbesar dari seluruh percobaan tersebut. Berikut grafik komparasi nilai *troughput* untuk setiap percobaan yang ditunjukan pada gambar 5.2, 5.3, 5.4, 5.5 dan 5.6.



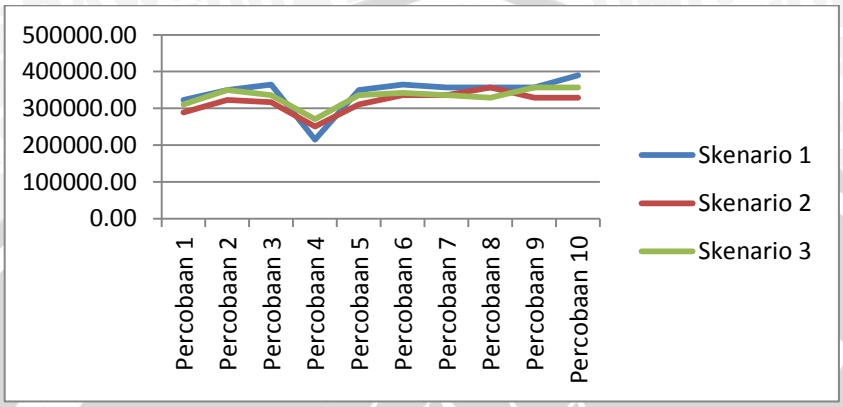
Gambar 5.2 Komparasi Troughput Paket Data 128 Byte

Hasil komparasi nilai troughput untuk besaran paket data 128 *byte* yang terdapat pada gambar 5.2 menunjukan dari setiap sepuluh kali percobaan yang dilakukan pada masing-masing skenario percobaan didapatkan nilai *troughput* terbesar 139890,71 *Kbps* yang terdapat pada percobaan ke lima di skenario satu.



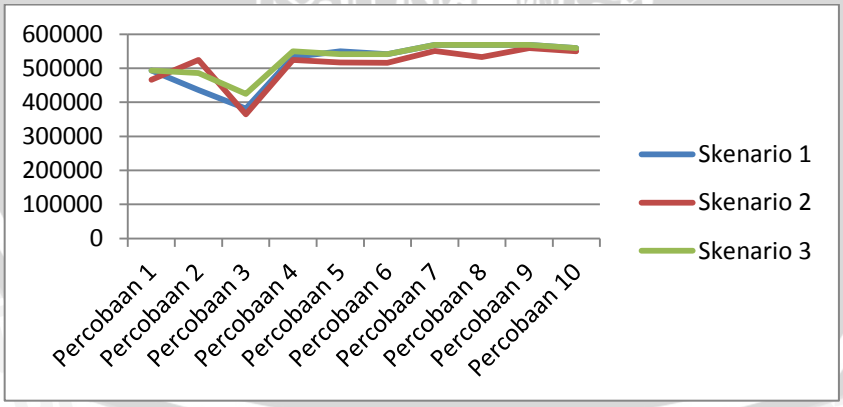
Gambar 5.3 Komparasi Troughput Paket Data 256 Byte

Hasil komparasi nilai troughput untuk besaran paket data 256 *byte* yang terdapat pada gambar 5.3 menunjukan dari setiap sepuluh kali percobaan yang dilakukan pada masing-masing skenario percobaan didapatkan nilai *troughput* terbesar 226749.34 *Kbps* yang terdapat pada percobaan ke tiga di skenario satu.



Gambar 5.4 Komparasi Troughput Paket Data 512 Byte

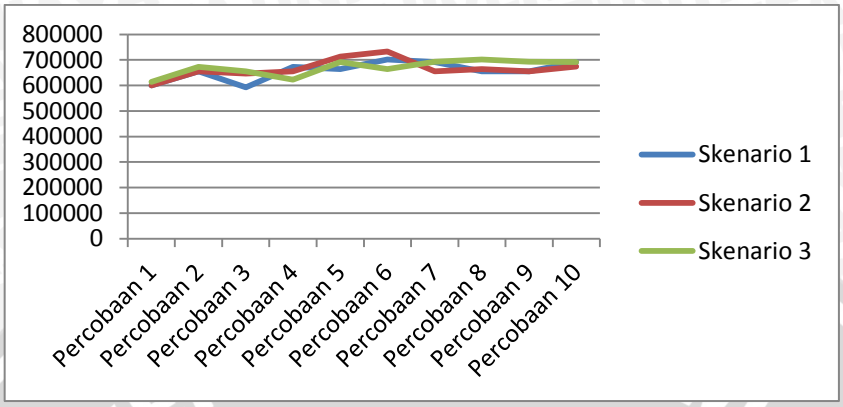
Hasil komparasi nilai troughput untuk besaran paket data 512 *byte* yang terdapat pada gambar 5.4 menunjukan dari setiap sepuluh kali percobaan yang dilakukan pada masing-masing skenario percobaan didapatkan nilai *troughput* terbesar 390243.90 *Kbps* yang terdapat pada percobaan ke sepuluh di skenario satu.



Gambar 5.5 Komparasi Troughput Paket Data 1024 Byte

Hasil komparasi nilai troughput untuk besaran paket data 1024 *byte* yang terdapat pada gambar 5.5 menunjukan dari setiap sepuluh kali percobaan yang dilakukan pada masing-masing skenario percobaan didapatkan nilai *troughput*

terbesar 357043,24 *Kbps* yang terdapat pada percobaan ke sepuluh di skenario tiga.

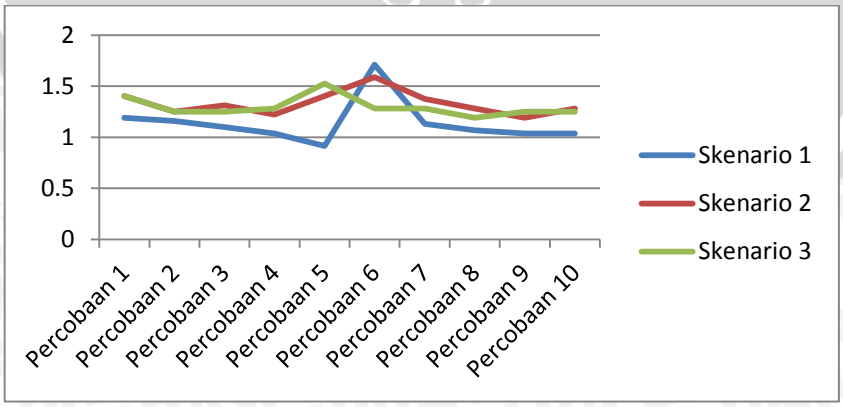


Gambar 5.6 Komparasi Troughput Paket Data 1500 Byte

Hasil komparasi nilai troughput untuk besaran paket data 1500 *byte* yang terdapat pada gambar 5.6 menunjukan dari setiap sepuluh kali percobaan yang dilakukan pada masing-masing skenario percobaan didapatkan nilai *troughput* terbesar 733496,33 *Kbps* yang terdapat pada percobaan ke enam di skenario dua.

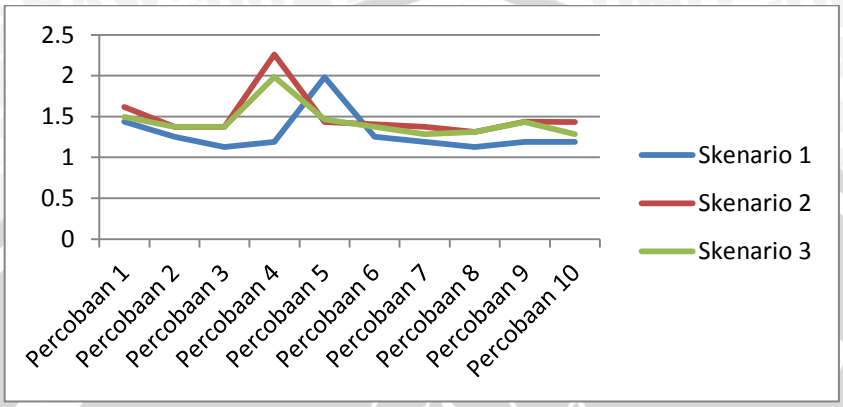
5.2.1.3 Komparasi Nilai Delay

Pada komparasi nilai *delay* akan dibandingkan nilai yang didapat dari setiap percobaan pengiriman paket data yaitu sebesar 128 *byte*, 256 *byte*, 512 *byte*, 1024 *byte* dan 1500 *byte* pada pengujian di skenario satu, skenario dua dan skenario tiga dan dicari nilai *delay* terkecil dari seluruh percobaan tersebut. Berikut grafik komparasi nilai *delay* untuk setiap percobaan yang ditunjukan pada gambar 5.7, 5.8, 5.9, 5.10 dan 5.11.



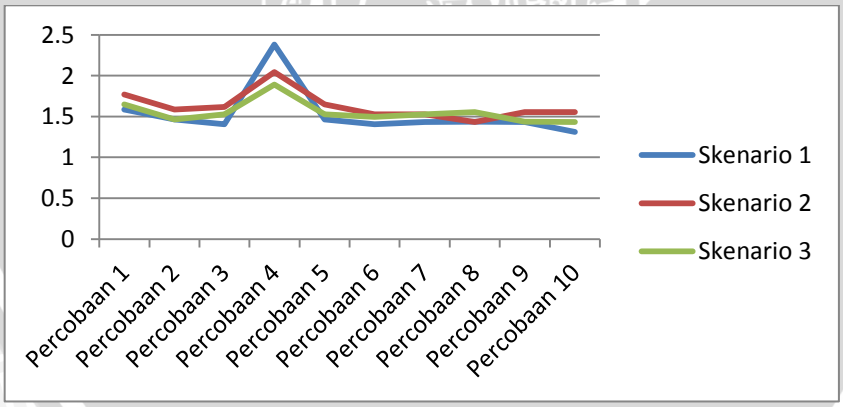
Gambar 5.7 Komparasi Delay Paket Data 128 Byte

Hasil komparasi nilai *delay* untuk besaran paket data 128 *byte* yang terdapat pada gambar 5.7 menunjukan dari setiap sepuluh kali percobaan yang dilakukan pada masing-masing skenario percobaan didapatkan nilai *delay* terkecil 0,915 *ms* yang terdapat pada percobaan ke lima di skenario satu.



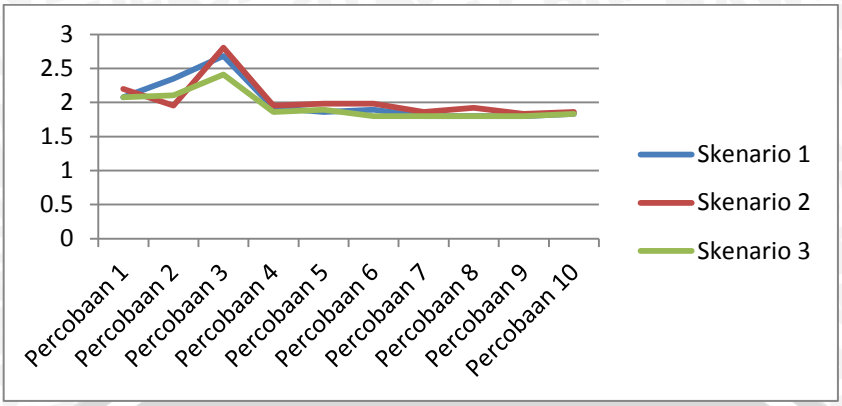
Gambar 5.8 Komparasi Delay Paket Data 256 Byte

Hasil komparasi nilai *delay* untuk besaran paket data 256 *byte* yang terdapat pada gambar 5.8 menunjukan dari setiap sepuluh kali percobaan yang dilakukan pada masing-masing skenario percobaan didapatkan nilai *delay* terkecil 1.1290 *ms* yang terdapat pada percobaan ke tiga dan delapan di skenario satu.



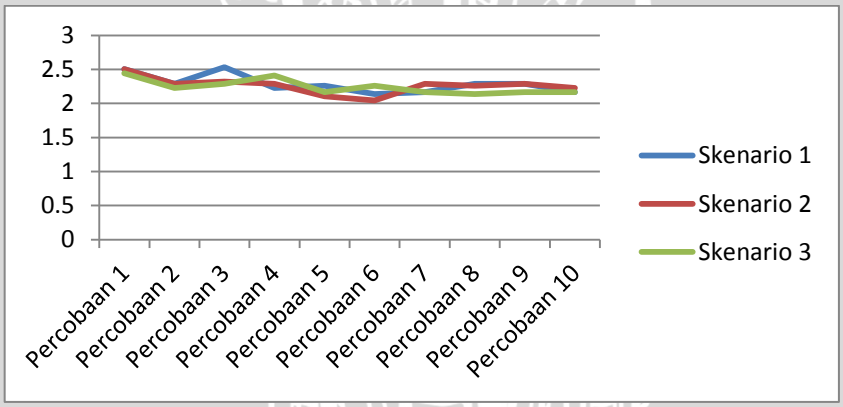
Gambar 5.9 Komparasi Delay Paket Data 512 Byte

Hasil komparasi nilai *delay* untuk besaran paket data 512 *byte* yang terdapat pada gambar 5.9 menunjukan dari setiap sepuluh kali percobaan yang dilakukan pada masing-masing skenario percobaan didapatkan nilai *delay* terkecil 1,312 *ms* yang terdapat pada percobaan ke sepuluh di skenario satu.



Gambar 5.10 Komparasi Delay Paket Data 1024 Byte

Hasil komparasi nilai *delay* untuk besaran paket data 1024 *byte* yang terdapat pada gambar 5.10 menunjukan dari setiap sepuluh kali percobaan yang dilakukan pada masing-masing skenario percobaan didapatkan nilai *delay* terkecil 1,800 *ms* yang terdapat pada percobaan ke sembilan di skenario tiga.



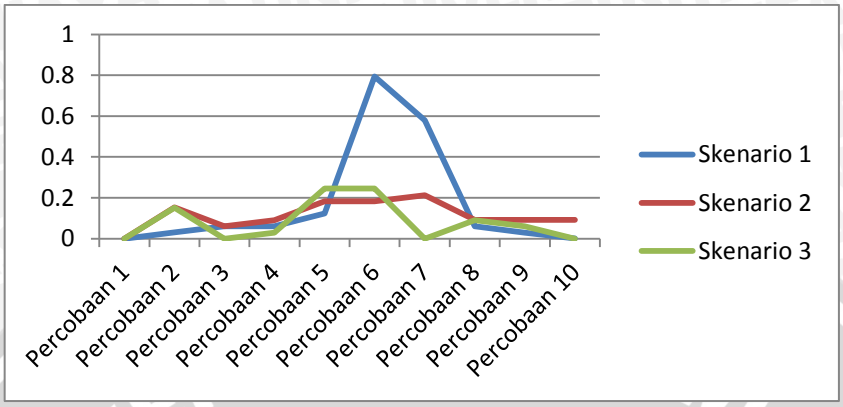
Gambar 5.11 Komparasi Delay Paket Data 1500 Byte

Hasil komparasi nilai *delay* untuk besaran paket data 1500 *byte* yang terdapat pada gambar 5.11 menunjukan dari setiap sepuluh kali percobaan yang dilakukan pada masing-masing skenario percobaan didapatkan nilai *delay* terkecil 2,045 *ms* yang terdapat pada percobaan ke enam di skenario dua.

5.2.1.4 Komparasi Nilai Jitter

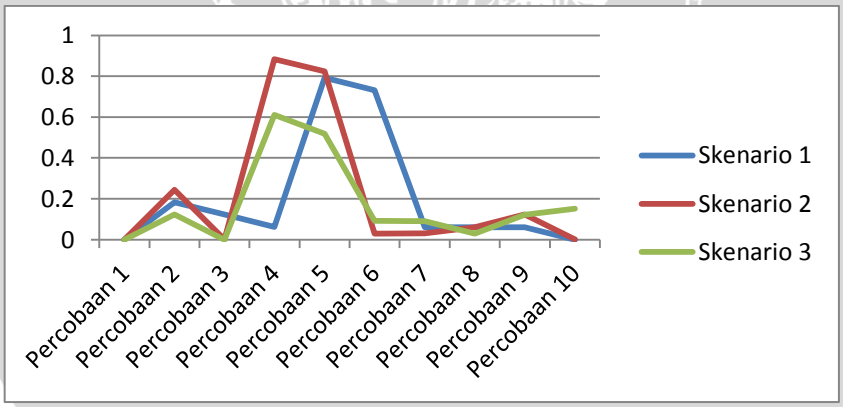
Pada komparasi nilai *jitter* akan dibandingkan nilai dari setiap percobaan pengiriman paket data yaitu sebesar 128 *byte*, 256 *byte*, 512 *byte*, 1024 *byte* dan 1500 *byte* pada pengujian di skenario satu, skenario dua dan skenario tiga dan dicari nilai *jitter* terkecil dari seluruh percobaan tersebut. Berikut grafik

komparasi nilai *jitter* untuk setiap percobaan yang ditunjukan pada gambar 5.12, 5.13, 5.14, 5.15 dan 5.16.



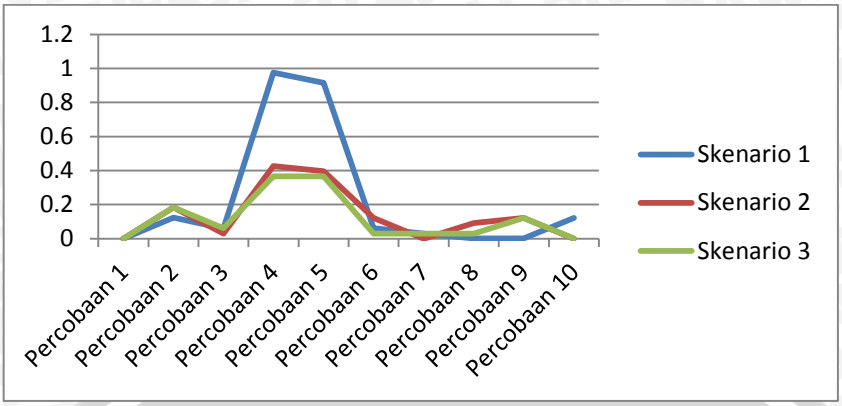
Gambar 5.12 Komparasi Jitter Paket Data 128 Byte

Hasil komparasi nilai *jitter* untuk besaran paket data 128 *byte* yang terdapat pada gambar 5.12 menunjukan dari setiap sepuluh kali percobaan yang dilakukan pada masing-masing skenario percobaan didapatkan nilai *jitter* terkecil 0 *ms* yang terdapat pada percobaan ke tiga dan sepuluh di skenario tiga.



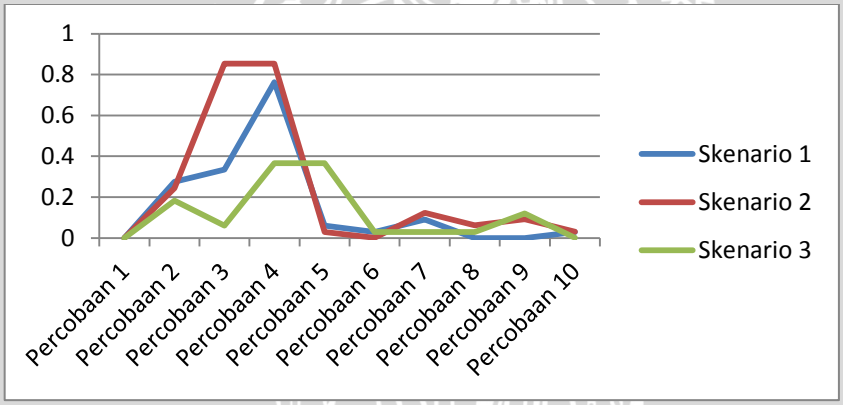
Gambar 5.13 Komparasi Jitter Paket Data 256 Byte

Hasil komparasi nilai *jiiter* untuk besaran paket data 256 *byte* yang terdapat pada gambar 5.13 menunjukan dari setiap sepuluh kali percobaan yang dilakukan pada masing-masing skenario percobaan didapatkan dua nilai *jitter* terkecil 0 *ms* yang terdapat pada percobaan ke sepuluh di skenario satu dan percobaan ke tiga di skenario tiga.



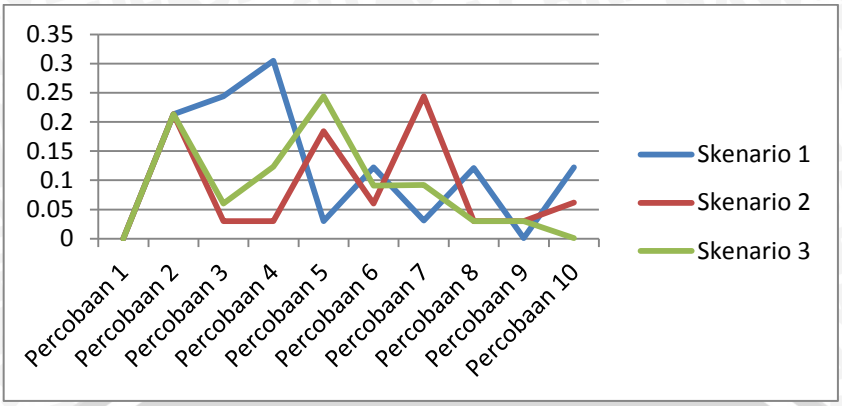
Gambar 5.14 Komparasi Jitter Paket Data 512 Byte

Hasil komparasi nilai *jiiter* untuk besaran paket data 512 *byte* yang terdapat pada gambar 5.14 menunjukan dari setiap sepuluh kali percobaan yang dilakukan pada masing-masing skenario percobaan didapatkan tiga nilai *jitter* terkecil yaitu 0 *ms* yang terdapat pada percobaan ke delapan dan sembilan di skenario satu dan pada percobaan ke tujuh dan sepuluh di skenario dua.



Gambar 5.15 Komparasi Jitter Paket Data 1024 Byte

Hasil komparasi nilai *jitter* untuk besaran paket data 1024 *byte* yang terdapat pada gambar 5.15 menunjukan dari setiap sepuluh kali percobaan yang dilakukan pada masing-masing skenario percobaan didapatkan nilai *jitter* terkecil 0 *ms* yang terdapat pada percobaan ke delapan dan sembilan di skenario satu.



Gambar 5.16 Komparasi Jitter Paket Data 1500 Byte

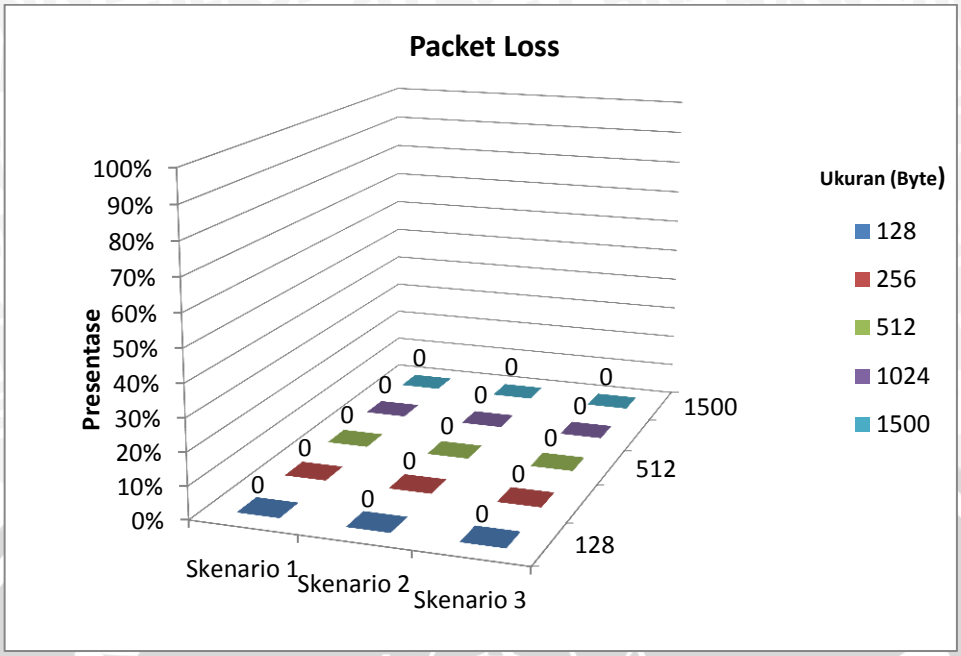
Hasil komparasi nilai *jitter* untuk besaran paket data 1500 *byte* yang terdapat pada gambar 5.16 menunjukan dari setiap sepuluh kali percobaan yang dilakukan pada masing-masing skenario percobaan didapatkan nilai *jitter* terkecil 0,0010 *ms* yang terdapat pada percobaan ke sembilan di skenario satu dan pada percobaan ke sepuluh di skenario tiga.

5.2.2 Komparasi Nilai Rata-rata

Pada sub-bab komparasi nilai rata-rata akan dilakukan perbandingan untuk melihat hasil terbesar nilai rata-rata yang didapat dari pengujian, berdasarkan hasil yang didapat dari skenario satu yaitu pengujian kondisi jaringan antara gedung A dan gedung B, skenario dua yaitu pengujian kondisi jaringan antara gedung B dan gedung C dan skenario tiga yaitu pengujian kondisi jaringan antara gedung C dan gedung A.

5.2.2.1 Komparasi Nilai Rata-rata Packet loss

Dalam komparasi nilai rata-rata *packet loss* yang dilakukan, nilai *packet loss* terkecilah yang merupakan nilai yang terbaik. Semakin kecil nilai *packet loss* maka semakin baik kondisi kualitas sebuah jaringan. Berikut gambar grafik yang menunjukan hasil komparasi nilai rata-rata *packet loss* dari ketiga skenario yang ditunjukan pada gambar 5.17.

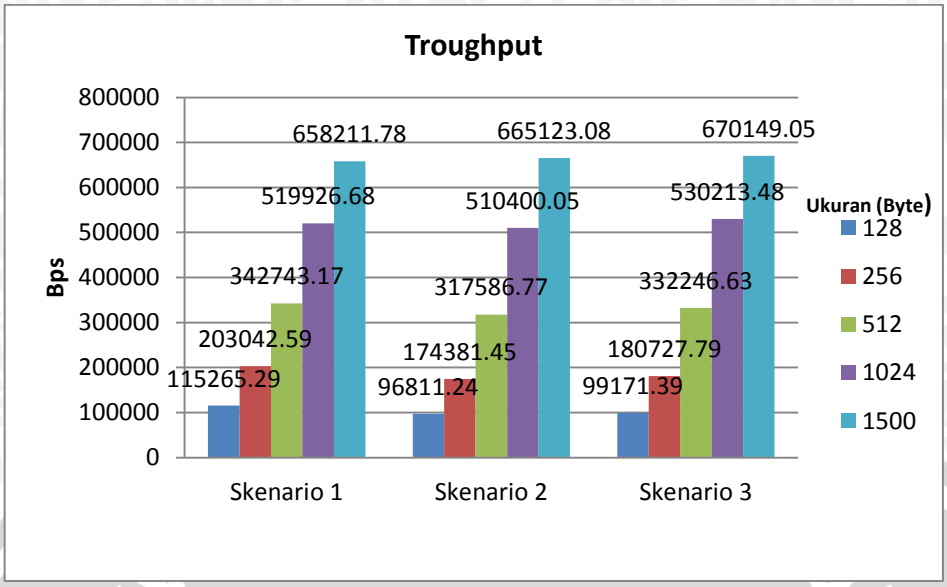


Gambar 5.17 Grafik Nilai Rata-rata *Packet loss*

Pada gambar 5.17 dapat dilihat bahwa nilai *packet loss* dari seluruh skenario, yaitu skenario satu, skenario dua dan skenario tiga didapatkan nilai *packet loss* 0 %. Hal ini menunjukan pada setiap percobaan yang dilakukan, pada skenario satu, skenario dua dan skenario tiga paket data yang dikirimkan berhasil terkirim tanpa adanya paket data yang hilang pada proses pengiriman dan penerimaan data antara *client* dan *server*. Dari semua skenario percobaan, yaitu skenario satu dengan pengujian kondisi jaringan antara gedung A dan gedung B, skenario dua dengan pengujian kondisi jaringan antara gedung B dan gedung C dan skenario tiga dengan pengujian kondisi jaringan antara gedung C dan gedung A didapatkan nilai *packet loss* 0% yang menunjukan nilai yang sangat bagus menurut [ETS-02].

5.2.2.2 Komparasi Nilai Rata-rata Troughput

Dalam komparasi nilai rata-rata *troughput* yang dilakukan, nilai *troughput* terbesar yang merupakan nilai yang terbaik. Semakin besar nilai *troughput* maka semakin baik kondisi kualitas sebuah jaringan. Berikut gambar grafik yang menunjukan hasil komparasi nilai rata-rata *troughput* dari ketiga skenario yang ditunjukan pada gambar 5.18.



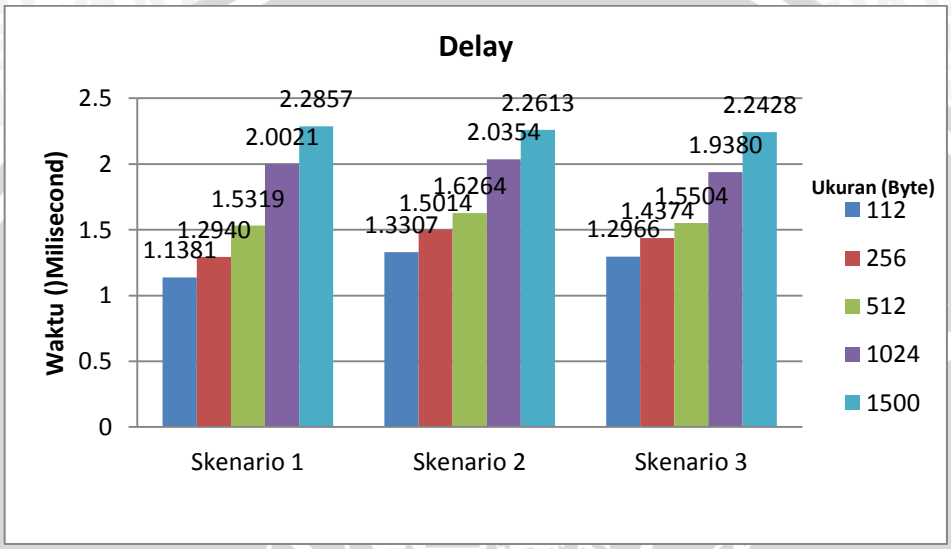
Gambar 5.18 Grafik Nilai Rata-rata Troughput

Dari gambar 5.18 nilai rata-rata *troughput* terbesar pada pengiriman paket data 128 *byte* adalah pada skenario satu yaitu 115265,29 *Bps* pada pengiriman paket data 256 *byte* nilai rata-rata *troughput* terbesar adalah pada skenario satu yaitu 203042,59 *Bps*, pada pengiriman paket data 512 *byte* nilai rata-rata *troughput* terbesar adalah pada skenario satu yaitu 342743,17 *Bps*, pada pengiriman paket data 1024 *byte* nilai rata-rata *troughput* terbesar adalah pada skenario tiga yaitu 530213,48 *Bps* dan pada pengiriman paket data 1500 *byte* nilai rata-rata *troughput* terbesar adalah pada skenario tiga yaitu 670149,05 *Bps*.

Berdasarkan hasil perbandingan nilai rata-rata terbesar antara skenario satu, skenario dua dan skenario tiga. Skenario satu memiliki nilai rata-rata *troughput* terbesar lebih banyak yaitu di pengiriman paket data 128 *byte*, 256 *byte* dan 512 *byte*. Skenario tiga memiliki dua nilai rata-rata *troughput* terbesar di pengiriman paket data 1024 *byte* dan 1500 *byte*. Nilai rata-rata *troughput* bila dihitung nilai totalnya dari seluruh besaran paket data yang didapatkan adalah pada skenario satu dengan pengujian kondisi jaringan antara gedung A dan gedung B memiliki nilai total 1839189,51 *Bps*, pada skenario dua dengan pengujian kondisi jaringan antara gedung B dan gedung C memiliki nilai total 1764302.59 *Bps* dan pada skenario tiga dengan pengujian kondisi jaringan antara gedung C dan gedung A memiliki nilai total 1812508,34 *Bps*.

5.2.2.3 Komparasi Nilai Rata-rata Delay

Dalam komparasi nilai rata-rata *delay* yang dilakukan, nilai *delay* terkecilah yang merupakan nilai yang terbaik. Semakin kecil nilai delay maka semakin baik kondisi kualitas sebuah jaringan. Berikut gambar grafik yang menunjukan hasil komparasi nilai rata-rata *delay* dari ketiga skenario yang ditunjukan pada gambar 5.19.



Gambar 5.19 Grafik Nilai Rata-rata Delay

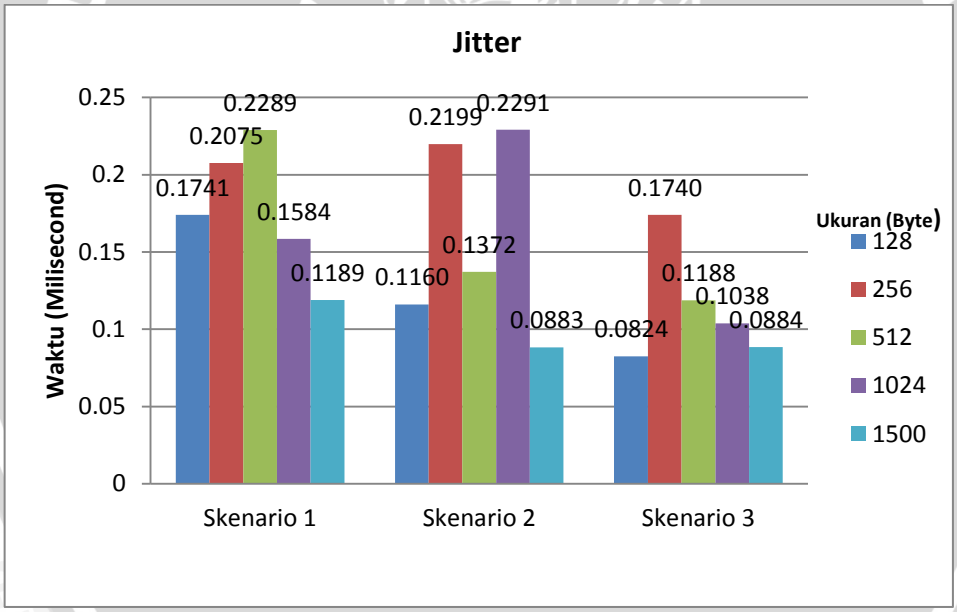
Dari gambar 5.19 nilai rata-rata *delay* terkecil pada pengiriman paket data 128 *byte* adalah pada skenario satu yaitu 1,1381 *ms*, pada pengiriman paket data 256 *byte* nilai rata-rata *delay* terkecil adalah pada skenario satu yaitu 1,5319 *ms*, pada pengiriman paket data 512 *byte* nilai rata-rata *delay* terkecil adalah pada skenario satu yaitu 1,5319 *ms*, pada pengiriman paket data 1024 *byte* nilai rata-rata *delay* terkecil adalah pada skenario tiga yaitu 1,9380 *ms* dan pada pengiriman paket data 1500 *byte* nilai rata-rata *delay* terkecil adalah pada skenario tiga yaitu 2,2428 *ms*.

Berdasarkan hasil perbandingan nilai rata-rata delay terkecil antara skenario satu, skenario dua dan skenario tiga. Skenario satu memiliki nilai rata-rata *delay* terkecil lebih banyak yaitu di pengiriman paket data 128 *byte*, 256 *byte* dan 512 *byte*. Skenario tiga memiliki dua nilai rata-rata *delay* terkecil di pengiriman paket data 1024 *byte* dan 1500 *byte*. Nilai rata-rata *delay*, bila dihitung nilai totalnya dari seluruh besaran paket data didapatkan adalah skenario satu

dengan pengujian kondisi jaringan antara gedung A dan gedung B memiliki nilai total 8,2518 *ms*, skenario dua dengan pengujian kondisi jaringan antara gedung B dan gedung C memiliki nilai total 8,7552 *ms* dan skenario tiga dengan pengujian kondisi jaringan antara gedung C dan gedung A memiliki nilai total 8,4652 *ms*.

5.2.2.4 Komparasi Nilai Rata-rata *Jitter*

Nilai *jitter* yang didapat pada pengujian ini, selalu berkaitan dengan nilai *delay* yang didapat pada setiap pengujian. Dikarenakan sesuai dasar teori tentang *jitter* yang terdapat pada bab II, *jitter* merupakan variasi *delay* yang terjadi pada jaringan. Semakin stabil atau sama nilai *delay* yang terjadi pada setiap percobaan, maka semakin stabil dan sama juga nilai *jitter*-nya. Berikut gambar grafik yang menunjukan hasil komparasi nilai rata-rata *packet loss* dari ketiga skenario yang ditunjukan pada gambar 5.20.



Gambar 5.20 Grafik Nilai Rata-rata *Jitter*

Dari gambar 5.20 nilai rata-rata *jitter* terkecil pada pengiriman paket data 128 *byte* adalah pada skenario tiga yaitu 0,0824 *ms*, pada pengiriman paket data 256 *byte* nilai rata-rata *delay* terkecil adalah pada skenario stiga yaitu 0,1740 *ms*, pada pengiriman paket data 512 *byte* nilai rata-rata *jitter* terkecil adalah pada skenario tiga yaitu tiga yaitu 0,1038 *ms*, pada pengiriman paket data 1024 *byte* nilai rata-rata *jitter* terkecil adalah pada skenario tiga yaitu 0,1038 *ms* dan pada

pengiriman paket data 1500 *byte* nilai rata-rata *jitter* terkecil adalah pada skenario dua yaitu 0,0883 *ms*.

Hasil dari komparasi nilai rata-rata *jitter* yang telah dilakukan untuk besaran paket data 128 *byte*, 256 *byte*, 512 *byte*, 1024 *byte* dan 1500 *byte* pada skenario satu, skenario dua dan skenario tiga, menunjukan pada skenario tiga didapatkan nilai rata-rata *jitter* terkecil, hanya pada nilai *jitter* di pengiriman paket data 1500 *byte* saja nilai *jitter* terkecil didapatkan pada skenario dua. Meskipun dari hasil komparasi nilai rata-rata *jitter* yang didapat pada skenario tiga merupakan nilai rata-rata *jitter* yang terkecil namun hasil komparasi ini bukan berarti menunjukan kondisi layanan jaringan terbaik dibandingkan skenario satu dan skenario dua. Dikarenakan nilai *jitter* hanya merupakan nilai variasi dari *delay* pada jaringan dan juga nilai *jitter* yang terdapat pada seluruh skenario pengujian memiliki nilai antara 0 *ms* – 0,2289 *ms.*

Hasil yang dapat disimpulkan dari komparasi tiga skenario pengujian yang telah dilakukan, dengan besaran paket data yang dikirimkan adalah sebesar 128 *byte*, 256 *byte*, 512 *byte*, 1024 *byte*, dan 1500 *byte*. Berdasarkan jumlah nilai rata-rata terbanyak parameter *packet loss, troughput, delay* dan *jitter* pada setiap jenis pengiriman paket data dan dari nilai total terbesar dan terkecil dari nilai rata-rata parameter tersebut, dapat disimpulkan bahawa kondisi kualitas jaringan pada skenario satu yaitu dengan pengujian kondisi jaringan antara gedung A dan gedung B memiliki kondisi jaringan yang terbaik bila dibandingkan dengan hasil pengujian pada skenario dua dengan pengujian kondisi jaringan antara gedung B dan gedung C dan skenario pengujian tiga dengan pengujian kondisi jaringan antara gedung C dan gedung A dengan perbandingan total nilai rata-rata yaitu nilai *troughput* 1839189,51 *Bps*, total nilai rata-rata *packet loss* 0 % dan total rata-rata nilai *delay* 8,2518 *ms*.

BAB VI
PENUTUP

Bab VI Penutup berisi kesimpulan dari hasil implementasi dan analisis sistem serta saran untuk pengembangan sistem.

6.1 Kesimpulan

Dari hasil implementasi, pengujian, dan analisis dari penelitian *embedded system network analyzer* pada jaringan *LAN* disimpulkan bahwa:

- Dalam implementasi dan pengujian pada penelitian *embedded system network analyzer* pada jaringan *LAN*, perangkat *beagleboard –XM* mampu menjalankan program *network analyzer* yang ditanam (*embed*) pada perangkat tersebut, mampu menjalankan fungsi input dari *keyboard* dan menampilkan hasil dari perhitungan kualitas jaringan berupa parameter *packet loss, troughput, delay* dan *jitter* di *LCD* yang terpasang pada perangkat *beagleboard –XM*.
- Hasil perhitungan *packet loss, troughput, delay* dan *jitter* pada penelitian *embedded system network analyzer* pada jaringan *LAN* mampu didapatkan dengan waktu yang cukup singkat, yaitu antara 5-10 detik untuk setiap skenario percobaan, dengan sepuluh kali percobaan pada setiap skenario yang berupa pengujian kondisi jaringan antara gedung A dan gedung B, antara gedung B dan gedung C dan antara gedung C dan gedung A, dengan batasan waktu satu detik untuk setiap pengiriman paket data.

6.2 Saran

Saran yang dapat disampaikan penulis untuk pengembangan pada penelitian *embedded system network analyzer* pada jaringan *LAN* adalah :

- Perlu pengembangan dan penelitian lebih lanjut untuk kemampuan cakupan area jaringan, sehingga proses perhitungan *parameter packet loss, troughput, delay* dan *jitter* dapat dilakukan pada segmen jaringan yang berbeda dan jaringan yang lebih luas dan besar.

- Perlunya menyederhanakan proses pengiriman paket data dari *client* menuju *server*, yang mana untuk melakukan proses pengiriman paket data hingga melakukan perhitungan parameter kualitas jaringan yang ditentukan membutuhkan proses konfigurasi *IP Address* dan *port* tujuan (*server*) secara manual.

DAFTAR PUSTAKA

[ANS - 08] Ansyori Rizal, 2008, "Desain, Implementasi Dan Analisis Interkoneksi Antara Protokol H.323 Dan SIP Pada Jaringan VoIP", Institut Pertanian Bogor.

[ARS - 09] Ramli, Kalamullah, 2009, "Arah Sistem Tertanam (Embedded System) dan Strategi Penguatan Industri Teknologi Informasi dan Telekomunikasi", Universitas Indonesia, hal.17-23.

[BAR-11] Barreiros, Miguel., Lundqvist, Peter, 2011, "QOS Enabled Networks : Tools And Foundations", John Willey & Sons Ltd, hal. 33-39.

[BEN-02] Bentham, Jeremy, 2002, "Web Server Embedded System - Second Edition", CMP Media LLC, hal.12-15.

[CHE-12] Chebrolu, Kameswari, 2012, "Socket Programming" Dept. of ElectricalEngineering, IIT Kanpur.

[DON-01] Donahoo, Michaelm, J. , Calvert, Kenneth, L., 2001, "TCP/IP Sockets in C – Practical Guide For Programmers", Morgan Kaufmann Publishers, hal.111-122.

[ETS - 02] ETSI TYPHON, 2002. "End-to-end Quality of Service in TIPHON Systems. Part 2: Definition of Quality of Service (QoS) Classes. *ETS*. 101 392-2"

[KUR-11] Kurniawan, Agus, 2011, "Dasar-Pemrograman-Socket-Dengan-Java", akses dari http://blog.aguskurniawan.net/post/Dasar-Pemrograman-Socket-Dengan-Java.aspx. Tanggal akses 31-1-2013.

[LEG-08] Legat Uros , 2008, "Embedded System Web Server", 9th International PhD Workshop on Systems and Control : Young Generation Viewpoint.

[MEH-12] Mehta, Vikram., Gupta, Neena, Dr., 2012, "Performance Analysis of QOS Parameter for Wimax Networks", International Journal of Engineering and Innovative Technology (IJEIT) Volume 1, Issue 5, May 2012.

[NOE-05] Nooergard, Tomy, 2005 , ”Embedded System Architecture (A Comprehensive Guide for Engineers and Programmers)”, Elsevier Inc , hal. 6-7.

[NIM-13] Anonim, 2013, ”Embedded System”, Akses dari http://en.wikipedia.org/wiki/Embedded_system

[TAN-03] Tanembaum, Andrew, S., 2003, “Computer Network – Fourth Edition”, Pearson Education, Inc. hal.301-311.

[PET-07] Peterson, L., Larry, & Davie, S., Bruce, 2007, “Computer Networks A System Approach”, Elsevier Inc, hal.40-43.

[RAH-09] Rahman Mostafijur, Ishwar Abdul Khalib Zahereel, Ahmad R. B, Mohd Salina Asi.2009.”Web-based Portable Network Traffic Monitoring System based on Embedded Linux and SBC” Universiti Malaysia Perlis (UniMAP).

[UZI-12] Uzzin, Isbat, 2012, ”Pemrogramman Socket” Politeknik Elektronika Negeri Surabaya – ITS, Surabaya.

[VAH-99] Vahid , Frank, & Givargis ,Tony, 1999, ”Embedded System Design: A Unified Hardware/Software Approach”, Department of Computer Science & Engineering, University of California, hal. 4-8,

[WAR-11] Warren, Tyler, 2011, ”Portable Network Analyzer”, Computer Engineering, University of Arkansas.

LAMPIRAN

Lampiran 1 : Kode Program server.cpp

```
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>

#define MAX_CONNECTION 5
#define BUFFER_SIZE 1500

using namespace std;

void ConnectionHandler(int Client_Socket) // conncetion handling
{
    int Recv_Msg_Length;
      char Buffer_Msgs[BUFFER_SIZE ];

    if ((Recv_Msg_Length = recv(Client_Socket, Buffer_Msgs, BUFFER_SIZE
, 0)) < 0) // socket command for reciving messages
        perror("recv() command failed");

    while (Recv_Msg_Length > 0)  //repeat until messages complete
    {

        if (send(Client_Socket, Buffer_Msgs, Recv_Msg_Length, 0) !=
Recv_Msg_Length) // send recieved message to current client connection
            perror("send() command failed");

        if ((Recv_Msg_Length = recv(Client_Socket, Buffer_Msgs,
BUFFER_SIZE , 0)) < 0)// recived command socket
            perror("recv() command failed");
            //printf("\t%s", Buffer_Msgs);
    }
    close(Client_Socket);
}

int main(int argc, char *argv[])
{
    int Server_Sockets;
    int Client_Sockets;
    struct sockaddr_in Server_IP;
    struct sockaddr_in Client_IP;
    unsigned short Server_Port;
    unsigned int Addres_Length;

    if (argc != 2)     // validate argument
    {
        fprintf(stderr, "Usage:  %s <Server Port>\n", argv[0]);
        exit(1);
    }

    Server_Port = atoi(argv[1]);   //gettting port from argument

     if ((Server_Sockets = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) <
0) //start socket connection with TCP protocol
```

```
        perror("socket() command failed");

      // init socket data for binding
    memset(&Server_IP, 0, sizeof(Server_IP));
    Server_IP.sin_family = AF_INET;
    Server_IP.sin_addr.s_addr = htonl(INADDR_ANY) ;
    Server_IP.sin_port = htons(Server_Port);

    if (bind(Server_Sockets, (struct sockaddr *) &Server_IP,
sizeof(Server_IP)) < 0) // binding data for socket listening
        perror("bind() command failed");

    if (listen(Server_Sockets, MAX_CONNECTION) < 0) // starting sokcet
listening
        perror("listen() command failed");
    printf("\nStart listening...\n");
    while (true)
    {

        Addres_Length = sizeof(Client_IP);
        if ((Client_Sockets = accept(Server_Sockets, (struct sockaddr
*) &Client_IP,  &Addres_Length)) < 0) // acceped any connection to
socket
            perror("accept() command failed");

        printf("\nClient connected %s\n",
inet_ntoa(Client_IP.sin_addr));
        ConnectionHandler(Client_Sockets); // handling  current
connection
    }
}
```

Lampiran 2 : Kode Program main.cpp

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>

#include "lcds.h"
#include "redirect.h"
#include "entry.h"
#include "buf.h"

#define BUFFER_SIZE_128 128
#define BUFFER_SIZE_256 256
#define BUFFER_SIZE_512 512
#define BUFFER_SIZE_1024 1024
#define BUFFER_SIZE_1500 1500

using namespace std;
```

```
int main() //int argc, char *argv[])
{
    int sock;
    unsigned short mode;
    struct sockaddr_in Server_IP;
    unsigned short Server_PORT = 0;
    char servIP[16]; //192.168.137.111

    const char *Msgs128
="10101010101010101010101010101010101010101010101010101010101010
11010101010101010101010101010101010101010101010101010101010101
01";
    const char *Msgs256
="10101010101010101010101010101010101010101010101010101010101010
11010101010101010101010101010101010101010101010101010101010101
10101010101010101010101010101010101010101010101010101010101011
01010101010101010101010101010101010101010101010101010101010101
01";
    const char *Msgs512
="10101010101010101010101010101010101010101010101010101010101010
11010101010101010101010101010101010101010101010101010101010101
10101010101010101010101010101010101010101010101010101010101011
01010101010101010101010101010101010101010101010101010101010101
01110101010101010101010101010101010101010101010101010101010101
11010101010101010101010101010101010101010101010101010101010101
10101010101010101010101010101010101010101010101010101010101011
01010101010101010101010101010101010101010101010101010101010101
01";
    const char *Msgs1024 =
"10101010101010101010101010101010101010101010101010101010101010
10101010101010101010101010101010101010101010101010101010101011
01010101010101010101010101010101010101010101010101010101010110
10101010101010101010101010101010101010101010101010101010101101
01010101010101010101010101010101010101010101010101010101011010
10101010101010101010101010101010101010101010101010101010110101
01010101010101010101010101010101010101010101010101010101101010
10101010101010101010101010101010101010101010101010101011010101
01010101010101010101010101010101010101010101010101010110101010
10101010101010101010101010101010101010101010101010101101010101
01010101010101010101010101010101010101010101010101011010101010
10101010101010101010101010101010101010101010101010110101010101
01010101010101010101010101010101010101010101010101101010101010
10101010101010101010101010101010101010101010101011010101010101
01010101010101010101010101010101010101010101010110101010101010
1";
    const char *Msgs1500 =
"10101010101010101010101010101010101010101010101010101010101010
10101010101010101010101010101010101010101010101010101010101011
01010101010101010101010101010101010101010101010101010101010110
10101010101010101010101010101010101010101010101010101010101101
01010101010101010101010101010101010101010101010101010101011010
10101010101010101010101010101010101010101010101010101010110101
01010101010101010101010101010101010101010101010101010101101010
10101010101010101010101010101010101010101010101010101011010101
01010101010101010101010101010101010101010101010101010110101010
```

```
10101010101010101010101010101010101010101010101010101011010101010
01010101010101010101010101010101010101010101010101010110101010101
10101010101010101010101010101010101010101010101010101101010101010
01010101010101010101010101010101010101010101010101011010101010101
10101010101010101010101010101010101010101010101010101101010101010
01010101010101010101010101010101010101010101010101011010101010101
10101010101010101010101010101010101010101010101010101101010101010
01010101010101010101010101010101010101010101010101011010101010101
10101010101010101010101010101010101010101010101010101101010101010
01010101010101010101010101010101010101010101010101011010101010101
10101010101010101010101010101010101010101010101010101101010101010
11010101010101010101010101010101010101010101010101010101010101010
10101010101010101011010101010101010101010101010101010101010101010
01010101010101010110101010101010101010101010101010101101010101010
01011010101010101010101010101";

    char * Buffer_Msgs = new char[BUFFER_SIZE_1500]();
    int Msgs_Length;
    int Bytes_Rcvd, Total_Bytes_Rcvd;


    if(lcds_init(20)==-1)  //init LCD 20 column
        exit(1);

    if(entry_init()==-1)     //init keyboard and enable lcd
        exit(1);

//ulangi terus
while(1){
    entry_ip_port_option(servIP, &Server_PORT, &mode); //entry
ip address, port and option
     //redirect printf to buffer
    if(!redirect_init())
        continue;
        //exit(1);

     printf("\n\nScanning Port : Memeriksa Port Pilihan dan
Tujuan 21,22,23,80 %u",Server_PORT);

     char cmd[100];

    sprintf(cmd,"nmap %s -
np21,22,23,80,%d>log.txt",servIP,Server_PORT);
    if(system(cmd))
        printf("");

     ostringstream oStr;
     oStr << Server_PORT;
    string portString = oStr.str() + "/tcp open";

     ifstream ifs("log.txt");

     bool isPortOpen = false;
    if (ifs.is_open())
    {
         string line;
```

```
        while (!ifs.eof()) {
           while ( getline(ifs, line)) {
          cout << line <<endl;
                    unsigned found = line.find(portString);
                    if (found!=std::string::npos) {
                        isPortOpen =true;
                    }
                }
           }

        ifs.clear();
    }
     if (!isPortOpen) {
          printf("Hasil : Port Tujuan Diblokir...\n\n");
     }else {
          printf("Hasil : Port Pilihan dan Tujuan
Terbuka...\n");;
     }
    if ((sock = socket( PF_INET,  SOCK_STREAM, IPPROTO_TCP)) <
0)

        printf("socket() command failed");

    memset(&Server_IP, 0, sizeof(Server_IP));

    Server_IP.sin_family      = AF_INET;
    Server_IP.sin_addr.s_addr = inet_addr(servIP);
    Server_IP.sin_port        = htons(Server_PORT);

    if (connect(sock, (struct sockaddr *) &Server_IP,
sizeof(Server_IP)) < 0)
        //perror
        printf("connect() command failed");

    printf("\nMemulai Pengiriman Paket Data...\n\n") ;

     int steps = 0;
     int tryCount = 10;
     int timeWait = 1 ;

     int packetSize =0;
     int packetDelivered = 0;
     double totPloss = 0;
     double totPut = 0;
     double timeDuration = 0;
     double sumJitter =0;

     double * arrayDelay = new double[tryCount]();

     while (steps <  tryCount)
     {
          timeval t1, t2;
          double elapsedTime;

          gettimeofday(&t1, NULL);
```

```
            if (mode==1) {
                Msgs_Length = strlen(Msgs128);
                printf("Percobaan-%d (128Byte) = ", steps);
                if (send(sock, Msgs128, Msgs_Length, 0) !=
Msgs_Length)
                    printf("send() command error");

            }else if (mode==2) {
                Msgs_Length = strlen(Msgs256);
                printf("Percobaan-%d (256Byte) = ", steps);
                if (send(sock, Msgs256, Msgs_Length, 0) !=
Msgs_Length)
                    printf("send() command error");

            }else if (mode==3) {
                Msgs_Length = strlen(Msgs512);
                printf("Percobaan-%d (512Byte) = ", steps);
                if (send(sock, Msgs512, Msgs_Length, 0) !=
Msgs_Length)
                    printf("send() command error");

            }else if (mode==4) {
                Msgs_Length = strlen(Msgs1024);
                printf("Percobaan-%d (1024Byte) = ", steps);
                if (send(sock, Msgs1024, Msgs_Length, 0) !=
Msgs_Length)
                    printf("send() command error");

            }else  {
                Msgs_Length = strlen(Msgs1500);
                printf("Percobaan-%d (1500Byte) = ", steps);
                if (send(sock, Msgs1500, Msgs_Length, 0) !=
Msgs_Length)
                    printf("send() command error");

            }
            packetSize += Msgs_Length;
            Total_Bytes_Rcvd = 0;

            int waitTime = 0;
            bool isFinish = false;
            Buffer_Msgs = new char[Msgs_Length]();
            while (!isFinish)
            {
                waitTime ++;

                if ((Bytes_Rcvd = recv(sock, Buffer_Msgs,
Msgs_Length - 1, 0)) <= 0)
                    printf("recv() command failed");

                Total_Bytes_Rcvd += Bytes_Rcvd;
                Buffer_Msgs[Bytes_Rcvd] = '\0';

                if (waitTime > timeWait || Total_Bytes_Rcvd ==
Msgs_Length )
                {
```

```
                    isFinish = true;
                }
            }
            packetDelivered +=  Total_Bytes_Rcvd;

            gettimeofday(&t2, NULL);

            elapsedTime = (t2.tv_sec - t1.tv_sec) * 1000.0;
            elapsedTime += (t2.tv_usec - t1.tv_usec) / 1000.0;

            double ploss = (((Msgs_Length -Total_Bytes_Rcvd) /
Msgs_Length) / elapsedTime) * 0.001;
            double troughPut = ((Total_Bytes_Rcvd / elapsedTime)
*1000.0);
            double Delay = elapsedTime ;
            double diffDelay =0;

            printf("\nPLoss:%.2lf%%; Tput:%.2lf Bps; delay:%.4lf
ms",ploss,troughPut,Delay);

            totPut += troughPut;
            totPloss += ploss;

            timeDuration += elapsedTime;
            arrayDelay[steps] = elapsedTime;
            if (steps != 0) {
                 diffDelay = ( arrayDelay[steps] -
arrayDelay[steps-1]);
                 if(diffDelay<0){
                diffDelay = -diffDelay;
                 }

                 sumJitter += diffDelay;
            }
            printf("; J:%.4lf ms \n", diffDelay);
            steps++;

      }
      double avgpacketloss = totPloss / tryCount;
      double avgtroughput = totPut / tryCount;
      double delay = timeDuration ;
      double avgDelay = delay / tryCount;
      double avgJitter = (sumJitter/tryCount);
    if(avgJitter<0){
       avgJitter = -avgJitter;
    }

      printf("\nHasil Perhitungan Parameter(Rata-rata) = \n\n");
      printf(" -Packet Loss      : %.2lf %% \n", avgpacketloss);
      printf(" -Throughput       : %.2lf Bps\n", avgtroughput);
      printf(" -Delay            : %.4lf ms\n", avgDelay);
      printf(" -Jitter           : %.4lf
ms\n\n",(sumJitter/tryCount));

    close(sock);
```

```
    redirect_flush(); //update buffer then move to buff
    redirect_deinit();

    buf_fill_from_buffer();  //move content buffer to buf
                        //buffer is in entry.h, while buf is
in buf.h

    buf_to_lcd(0,0);    //tampilkan view pertama
    //read keyboard and display result on text LCD
    //use LEFT, RIGHT, UP,DOWN to scroll one display on Text LCD
    //use ENTER to repeat testing

    while(kbd_get(KBD_UP)==KEY_ENTER);

    int key = KEY_NONE;
    int r = 0;
    int c = 0;
    while(key!=KEY_ENTER){
        key = kbd_get(KBD_DOWN);

        switch(key){
            case KEY_UP:    //kurangi baris
                lcds_puts("UP\n");
                if(r>0)
                    r--;
                buf_to_lcd(r,c);
                break;
            case KEY_DOWN:  //tambah baris
                lcds_puts("DOWN\n");
                if(r<VIEW_ROW_NUM-1)
                    r++;
                buf_to_lcd(r,c);
                break;
            case KEY_LEFT:  //kurangi kolom
                lcds_puts("LEFT\n");
                if(c>0)
                    c--;
                buf_to_lcd(r,c);
                break;
            case KEY_RIGHT: //tambah kolom
                lcds_puts("RIGHT\n");
                if(c<VIEW_COL_NUM-1)
                    c++;
                buf_to_lcd(r,c);
                break;
            default:
                break;
        }

        //tunggu tombol dilepas
        while(key!=KEY_NONE && key==kbd_get(KBD_UP));
        usleep(200);
        if(key==KEY_ENTER){
            break;
        }
    }
```

```
}
    kbd_deinit();

    serial_port_close();

    exit(0);
}
```

Lampiran 3 : kode Program kbd.h

```
#ifndef KBD_H_INCLUDED
#define KBD_H_INCLUDED

#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include <stdlib.h>
#include <termios.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <linux/input.h>

const char *ptr = "/dev/input/event2";

int kbd_handle;

int kbd_init(){

    kbd_handle = open((const char *)ptr, O_RDONLY);

    if (kbd_handle < 0)
    {
        fprintf(stderr, "gagal membuka device input%s: %s\n", (const cha
*)ptr, strerror(errno));
        return -1;
    }
    return 0;
}

#define KEY_NONE                -1
#define ERR_KBD_READ            -1
#define ERR_KBD_READ_PENDING    -2

#define KBD_UP   0
#define KBD_DOWN 1
//mendeteksi tombol yang ditekan
int kbd_get(int up_down){
    struct input_event ev;
    int code;
    if (read(kbd_handle, &ev, sizeof(struct input_event)) < 0){
        fprintf(stderr, "gagal membaca event input dari device input %s:
%s\n", (const char *)ptr, strerror(errno));
        if (errno == EINTR)
            return ERR_KBD_READ_PENDING;
        return ERR_KBD_READ;
    }

    code = KEY_NONE;
        if (ev.type == EV_KEY && ev.value==up_down)
    {
```

```
        switch (ev.code)
        {
        case KEY_0:
        case KEY_1:
        case KEY_2:
        case KEY_3:
        case KEY_4:
        case KEY_5:
        case KEY_6:
        case KEY_7:
        case KEY_8:
        case KEY_9:
        case KEY_DOT:
        case KEY_SPACE:
        case KEY_ENTER:
        case KEY_BACKSPACE:
        case KEY_UP: //eUp:
        case KEY_RIGHT: //eRight:
        case KEY_LEFT: //eLeft:
        case KEY_DOWN: //eDown:
            code = ev.code;
            break;
        default:
            break;
        }
    }
    return code;
}
void kbd_deinit(){
    close(kbd_handle);
}
#endif // KBD_H_INCLUDED
```

Lampiran 4 : kode Program entry.h

```
#ifndef ENTRY_H_INCLUDED
#define ENTRY_H_INCLUDED

#include "kbd.h"
#include "lcds.h"
int entry_init(){
    lcds_enable();  //enable LCD to receive command as well as data
    return kbd_init();
}

#define ENTRY_BUFFER_LEN    21
char entry_buffer[ENTRY_BUFFER_LEN];

//convert key to char
char entry_key_to_char(int key){

    char chr = 0;
    switch (key)
    {
        case KEY_0:
            chr = '0';
            break;
        case KEY_1:
            chr = '1';
            break;
        case KEY_2:
```

```
            chr = '2';
            break;
        case KEY_3:
            chr = '3';
            break;
        case KEY_4:
            chr = '4';
            break;
        case KEY_5:
            chr = '5';
            break;
        case KEY_6:
            chr = '6';
            break;
        case KEY_7:
            chr = '7';
            break;
        case KEY_8:
            chr = '8';
            break;
        case KEY_9:
            chr = '9';
            break;
        case KEY_DOT:
            chr = '.';
            break;
        case KEY_SPACE:
            chr = ' ';
            break;
        default:
            break;
    }
    return chr;
}

#define ALWYAS_NEW_ENTRY 1

#define ENTRY_ROW        3
void entry_ip_port_option(char *servIP, unsigned short *Server_PORT,
                          unsigned short *mode){
    int i = 0;
    int key = KEY_NONE;
    char ch;
    int argc = 0;
    lcds_cursor_off();
    lcds_enable();

    lcds_clrscr();  //hapus LCD
    lcds_puts("<IP> <Port> <Option>"); //tampilkan pesan
    lcds_goto(1,0);
    lcds_puts("contoh:");
    lcds_goto(2,0);
    lcds_puts("192.168.137.4 1122 1"); //tampilkan pesan
    lcds_goto(ENTRY_ROW,0);     //tunjuk baris kedua kolom pertama
    lcds_cursor_blink();


    #if !ALWYAS_NEW_ENTRY
    lcds_puts(entry_buffer); //tampilkan entry terakhir
    #endif
```

```
while(1){
        key = kbd_get(KBD_DOWN); //baca tombol keyboard pada saat
ditekan

        switch(key){
            case KEY_0:
            case KEY_1:
            case KEY_2:
            case KEY_3:
            case KEY_4:
            case KEY_5:
            case KEY_6:
            case KEY_7:
            case KEY_8:
            case KEY_9:
            case KEY_DOT:
            case KEY_SPACE:
                //cegah overwrite outside buffer
                if(i<ENTRY_BUFFER_LEN){
                    ch = entry_key_to_char(key);
                    entry_buffer[i++] = ch;
                    lcds_putch(ch);
                }
                break;
            case KEY_BACKSPACE:
                //hapus karakter sekarang dan kembali ke posisi
sebelumnya
                if(i>=0){
                    ch = ' ';
                    entry_buffer[i] = ch;
                    if(i>0) i--;
                    lcds_goto(ENTRY_ROW,i);
                    lcds_putch(ch);
                    lcds_goto(ENTRY_ROW,i);
                }
                break;
            case KEY_ENTER:
                entry_buffer[i++]=' '; //to enable scanning data
through loop
                entry_buffer[i] = 0; //end of string/entry
                break;
        }
                if(key==KEY_ENTER){
            argc = 0;
            char *p = entry_buffer;
            char str[16];
            char *d = servIP;
            char ever_data = 0;

            while(*p){  //end of entry indicated by 0

                                if(*p==' ' || *p==0){

                    if(ever_data){
                        argc++;
                        ever_data = 0;
                        *d = 0;          /
                        if(argc==2){
                            *Server_PORT=atoi(str);

                        }
```

```
                    else if(argc==3){

                        *mode=atoi(str);
                        if(*mode>5)
                            *mode = 1;
                                                }

                        d = str;
                    p++;

                    while(*p==' ')
                        p++;
                }
                else{
                    ever_data = 1;
                    *d = *p;
                    d++;
                    p++;
                }
            }

            lcds_cursor_off();
            //jika format entry salah maka ulangi lagi entry
            if(argc!=3){

                lcds_goto(ENTRY_ROW,0);
                lcds_puts("format salah!");
                sleep(1);
                //while(kbd_get(KBD_DOWN)!=KEY_ENTER);

            #if ALWYAS_NEW_ENTRY
                //kosongkan atau
                lcds_goto(ENTRY_ROW,0);
                lcds_puts("                ");
                lcds_goto(ENTRY_ROW,0);
                lcds_cursor_blink();
                i=0;
                entry_buffer[i]=0;
            #else
                //atau tampilkan kembali???
                lcds_puts(entry_buffer);
            #endif
                continue;  //ulangi entry lagi
            }
            else{
                lcds_cursor_off();
                break;  //keluar dari entry
            }
        }

        while(key!=KEY_NONE && key==kbd_get(KBD_UP));
        usleep(100);
    }//while
}
#endif // ENTRY_H_INCLUDED
```

Lampiran 5 : kode Program lcds.h

```
#ifndef LCDS_H_INCLUDED
#define LCDS_H_INCLUDED

#if defined(__AVR_LIBC_VERSION_STRING__)
    #include "fcpu.h"
    #include <util/delay.h>
    #include "uart.h"


#elif defined(__CODEVISION_AVR__)
    #include <delay.h>
    #define _delay_ms(d)        delay_ms(d)
    #define uart_9600_init()
    #define uint8_t             unsigned char
    #define uart_putc(c)        putchar(c)
#else
    #include "serial.h"
    #include <unistd.h> //usleep dan sleep
    #define _delay_ms(d)         {
                                 for(int i=0; i<d; i++)\
                                 usleep(1000);   }

    #define uart_9600_init()     serial_port_open()
    #define uint8_t              unsigned char
    #define uart_putc(c)         serial_port_write_byte(c)

#endif
#defineLCDS_CMD_CLEAR_SCREEN                0x01
#define LCDS_CMD_DISPLAY_LEFT               0x18
#define LCDS_CMD_DISPLAY_RIGHT              0x1c
#define LCDS_CMD_HOME                       0x00
#define LCDS_CMD_CURSOR_LEFT                0x10
#define LCDS_CMD_CURSOR_RIGHT               0x14
#define LCDS_CMD_UNDERLINE_CURSOR_ON        0x0e
#define LCDS_CMD_CURSOR_BLINK               0x0d
#define LCDS_CMD_CURSOR_OFF                 0x0c  /
#define LCDS_CMD_BLANK_DISPLAY              0x08
#define LCDS_CMD_RESTORE_DISPLAY            0x0c  R
#define LCDS_CMD_SET_ADDRESS_DDRAM          0x80
#define LCDS_CMD_SET_ADDRESS_CGRAM          0x40  /

#define LCDS_CMD_ENABLE                      253
#define LCDS_CMD_DISABLE                     252

#define COL_NUM_DEFAULT 16
unsigned char col_num = COL_NUM_DEFAULT;

int lcds_init(unsigned char col){
    int status = -1;
      status = uart_9600_init();
      col_num = col;
      _delay_ms(500); //wait after power up
      return status;
}

void lcds_command(uint8_t cmd){
      uart_putc(254);
      uart_putc(cmd);
      if(cmd==LCDS_CMD_CLEAR_SCREEN || cmd==LCDS_CMD_HOME){
       _delay_ms(20);
```

```
        }
}
void lcds_set_address_ddram(uint8_t addr){
        lcds_command(LCDS_CMD_SET_ADDRESS_DDRAM + addr);
}

void lcds_set_address_cgram(uint8_t addr){
        lcds_command(LCDS_CMD_SET_ADDRESS_CGRAM + addr);
}
/**
put a char to serial LCD
*/
void lcds_putch(uint8_t ch){
        uart_putc(ch);
        usleep(50);
}
void lcds_puts(const char* str){
    serial_port_write((char*)str);
    usleep(50);
    //usleep(9500);
        }
void lcds_disable(void){
    lcds_command(LCDS_CMD_DISABLE);
}

void lcds_enable(void){
    lcds_command(LCDS_CMD_ENABLE);
}

void lcds_goto(uint8_t row, uint8_t col){
    lcds_enable();
      lcds_set_address_ddram((row%2? (row>>1)*col_num+0x40 :
                        (row>>1)*col_num )+ col);
}
void lcds_clrscr(void){
    lcds_enable();
      lcds_command(LCDS_CMD_CLEAR_SCREEN);
}
void lcds_display_left(void){
        lcds_command(LCDS_CMD_DISPLAY_LEFT);
}
void lcds_display_right(void){
        lcds_command(LCDS_CMD_DISPLAY_RIGHT);
}
void lcds_home(void){
        lcds_command(LCDS_CMD_HOME);
}
void lcds_cursor_left(void){
        lcds_command(LCDS_CMD_CURSOR_LEFT);
}

void lcds_cursor_right(void){
        lcds_command(LCDS_CMD_CURSOR_RIGHT);
}
void lcds_underline_cursor(void){
        lcds_command(LCDS_CMD_UNDERLINE_CURSOR_ON);
}

vid lcds_cursor_blink(void){
        lcds_command(LCDS_CMD_CURSOR_BLINK);
}
```

```
void lcds_cursor_off(void){
      lcds_command(LCDS_CMD_CURSOR_OFF);
}
void lcds_blank_display(void){
      lcds_command(LCDS_CMD_BLANK_DISPLAY);
}
void lcds_restore(void){
      lcds_command(LCDS_CMD_RESTORE_DISPLAY);
}
void lcds_newchar(uint8_t char_id, uint8_t* pattern){
    uint8_t i;
      lcds_set_address_cgram(char_id*8);
      for(i=0; i<8; i++){
            uart_putc(*pattern++);
      }
}
#endif // LCDS_H_INCLUDED
```

Lampiran 6 : kode Program buf.h

```
#ifndef BUF_H_INCLUDED
#define BUF_H_INCLUDED

#include "lcds.h"
#include "redirect.h"
#define LCD_ROW_NUM     4
#define LCD_COL_NUM     20

#define VIEW_ROW_NUM    11 //must >= MAX_ROW/4, MAX_ROW ada di
redirect.h
#define VIEW_COL_NUM    4


#define BUF_ROW_NUM     (VIEW_ROW_NUM*LCD_ROW_NUM)
#define BUF_COL_NUM     (VIEW_COL_NUM*LCD_COL_NUM)

char buf[BUF_ROW_NUM][BUF_COL_NUM];
/
void buf_clear(){
    unsigned int r, c;
    for(r=0; r<BUF_ROW_NUM; r++){
        for(c=0; c<BUF_COL_NUM; c++)
            buf[r][c] = ' ';
    }
}
char lcd_buf[LCD_ROW_NUM][LCD_COL_NUM+1]; //+1 is for null

void buf_view(char rowv, char colv){
    unsigned int r, c, rr, cc;
    char chr;
    for(r=0; rr = r + (rowv*LCD_ROW_NUM), r<LCD_ROW_NUM; r++){
                for(c=0; cc = c+(colv*LCD_COL_NUM), c<LCD_COL_NUM;
c++){
            if(rr>=BUF_ROW_NUM || cc>=BUF_COL_NUM){
                chr = 'x';
            }
            else
                chr = buf[rr][cc];
            lcd_buf[r][c] = chr==0?' ':chr;
            //printf("%c",chr);
        }
        lcd_buf[r][c] = 0;
```

```
        //printf("\n");
    }
}
void buf_to_lcd(char rowv, char colv){

    buf_view(rowv,colv);
//
  //  lcds_blank_display();
    for(int i=0; i<LCD_ROW_NUM; i++){
        lcds_goto(i,0);
        //printf("%s",&lcd_buf[i][0]);
        lcds_puts(&lcd_buf[i][0]);
    }
  }

void buf_fill_from_buffer(){

    char *s = (char*)buffer;

    int ln = 0;
    int col=0;
    char str[10];

    buf_clear();

        while(*s=='\n')
        s++;

    while(*s){
        if((*s=='\n') || (col>=BUF_COL_NUM)){
            col=0;
            ln++;
            s++;
            continue;
        }
        buf[ln][col] = *s;
        col++;
        s++;
    }

    sprintf(str,"number of line=%d\n",ln);
    lcds_puts(str);

}
#endif // BUF_H_INCLUDED
```

Lampiran 7: Kode Program redirect.h

```
#ifndef REDIRECT_H_INCLUDED
#define REDIRECT_H_INCLUDED
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define MAX_ROW     44
#define MAX_COL     80
#define MAX_LEN     (MAX_ROW*MAX_COL)

char buffer[MAX_LEN+1] = {0};
int out_pipe[2];
int saved_stdout;
```

```
//menginisialisasi redirect supaya printf
int redirect_init(){
    //lcds_puts("redirect_init begin\n");
    saved_stdout = dup(STDOUT_FILENO);
        //exit(1);
    }
    dup2(out_pipe[1], STDOUT_FILENO);
    close(out_pipe[1]);

    //lcds_puts("redirect_init end\n");
    return 1;
}

int redirect_flush(){
    fflush(stdout);
    return read(out_pipe[0], buffer, MAX_LEN);

void redirect_deinit(){
    dup2(saved_stdout, STDOUT_FILENO);
    close(out_pipe[0]);
}
#endif // REDIRECT_H_INCLUDED
```

Lampiran 8 : Kode Program serial.h

```
#ifndef SERIAL_H_INCLUDED
#define SERIAL_H_INCLUDED

#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include <stdlib.h>
#include <termios.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

#define BAUDRATE    B38400

#define PORT_NAME "/dev/ttyS2"
int serial_port;
struct termios options_original;

// Resets the terminal and closes the serial port.

void serial_port_close()
{
      printf("serial port close...\n");
      //signal(SIGINT,(void*)0); //detach handler
      tcsetattr(serial_port,TCSANOW,&options_original);
      close(serial_port);
}

void  sigint_handler(int sig);
int serial_port_open(void)
{
  struct termios options;

  serial_port = open(PORT_NAME, O_RDWR | O_NONBLOCK) ;
```

```
  if (serial_port != -1)
  {
        printf("Serial Port open...\n");

        signal(SIGINT,(sighandler_t)sigint_handler); //attach handler

        tcgetattr(serial_port,&options_original);
        tcgetattr(serial_port, &options);

      printf("%d\n",options_original.c_cflag);
        //set baudrate

        cfsetispeed(&options, BAUDRATE); //B115200
        cfsetospeed(&options, BAUDRATE);

        options.c_cflag |= (CLOCAL | CREAD); //CS8
        //set 8N1
        options.c_cflag &= ~PARENB;
        options.c_cflag &= ~CSTOPB;
        options.c_cflag &= ~CSIZE;
        options.c_cflag |= CS8;
        //no flow control
        options.c_cflag &= ~CRTSCTS;

        //options.c_lflag |= ICANON | ISIG;

        if (tcsetattr(serial_port, TCSANOW, &options)!=0)
        {
              printf("error %d from tcsetattr\n", errno);
              return (-1);
        }
        else
          printf("%d\n",options.c_cflag);
  }
  else
  {
        printf("Unable to open %s",PORT_NAME);
        printf("Error %d opening %s: %s",errno, PORT_NAME,
strerror(errno));
  }
  return (serial_port);
}

int serial_port_read(char *read_buffer, size_t max_chars_to_read)
{
      int chars_read = read(serial_port, read_buffer,
max_chars_to_read);

      return chars_read;
}

void serial_port_write(char *write_buffer)
{
      size_t bytes_written;
      size_t len = 0;

      len = strlen(write_buffer);
      bytes_written = write(serial_port, write_buffer, len);
      if (bytes_written < len)
      {
            printf("Write failed \n");
```

```
        }
        }

void serial_port_write_byte(char c){
    int bytes_written = write(serial_port, &c, 1);
    if (bytes_written != 1)
        {
            printf("Write failed \n");
        }
}

void  sigint_handler(int sig)
{
        serial_port_close();
        exit (sig);
}
#endif // SERIAL_H_INCLUDED
```