Hasil Penelitian

Yang Tidak Dipublikasikan

SimPyNetLib v1.0: Simulator Pemodelan Lapisan Fisik dan Datalink Jaringan Komputer berbasis Pustaka Python pada Sistem Operasi Linux

Berkah I. Santoso, ST, MTI

Guson P. Kuntarto, ST, M.Sc

Irwan Prasetya Gunawan, Ph.D

Yusuf Lestanto, ST, M.Sc, MBA

Dimas Aryo Anggoro, S.Kom, M.Sc

Dewi Fatmawati Surianto, S.Kom, M.Kom

Fakultas Teknik dan Ilmu Komputer

Universitas Bakrie

Jakarta

2025/2026

LEMBAR PENGESAHAN HASIL PENELITIAN YANG TIDAK DIPUBLIKASIKAN

1.	Judul Penelitian	: SimPyNetLib v1.0: Simulator Pemodelan Lapisan Fisik dan Datalink Jaringan Komputer berbasis Pustaka Python pada Sistem Operasi Linux
2.	Peneliti Utama	
	a. Nama Lengkap	: Berkah I. Santoso, ST, MTI
	b. Jenis Kelamin	: Laki Laki
	c. Pangkat/Golongan/NIDN	: Asisten Ahli / III b / 0309068003
	d. Bidang Keahlian	: Cloud Computing, Networking, Simulation
	e. Program Studi	: Informatika
3.	Tim Peneliti	: Guson P. Kuntarto, ST, M.Sc, Irwan Prasetya Gunawan, Ph.D, Yusuf Lestanto, ST, M.Sc, MBA Dimas Aryo Anggoro, S.Kom, M.Sc Dewi Fatmawati Surianto, S.Kom, M.Kom
4.	Jangka waktu penelitian	: 5 Januari 2026 – 28 Januari 2026

Jakarta, 29 Januari 2026

Menyetujui,

Ketua Lembaga Penelitian dan Pengembangan	**Peneliti Utama**
(**Deffi Ayu Puspito Sari, Ph.D.**)	(**Berkah I. Santoso, ST, MTI**.)
NIDN: 0308078203	NIDN: 0309068003

Abstrak

Keberhasilan suatu aplikasi simulasi untuk dapat mendeskripsikan cara kerja suatu lapisan jaringan komputer sangat tergantung pada algoritma untuk menggambarkan perilaku *traffic* jaringan komputer secara lengkap dan akurat. Pengujian yang dilakukan secara langsung terhadap algoritma dan model yang diusulkan umumnya membutuhkan biaya tinggi serta waktu yang relatif panjang. Portabilitas aplikasi simulasi pada banyak sistem operasi menentukan besar atau kecilnya penerimaan penggunaan simulator. Hal tersebut berpengaruh terhadap efektivitas penerapan model berikut skema yang digunakan pada perangkat simulasi jaringan. Pengembangan simulator SimPyNetLib v1.0 ditujukan untuk mendeskripsikan lapisan jaringan komputer kepada pengguna yang dapat berjalan pada banyak sistem operasi. Pada versi pertama ini pengembangan simulator difokuskan pada *physical layer* dan *datalink layer* untuk mendapatkan karakteristik pemodelan pada masing-masing *layer* dengan menggunakan bahasa pemrograman Python berikut pustaka yang dibutuhkan seperti PyGObject, Numpy, Pandas, Matplotlib untuk membentuk simulator lapisan jaringan komputer sehingga dapat dijalankan pada beberapa sistem operasi. Hasil pengujian fungsional simulator SimPyNetLib v1.0 yang dilakukan pada *environment* Ubuntu Linux versi terakhir didapatkan bahwa simulator dapat berjalan dengan normal untuk masing-masing *physical layer and datalink layer* dengan menghasilkan representasi grafis simulator beserta fitur hasil simulasi dalam *export comma separated value* (CSV). Hasil simulasi berformat CSV dapat digunakan untuk menghasilkan representasi grafis pada aplikasi visualisasi *plotting* seperti Matlab®, atau GNU Octave.

Daftar Isi

Bab 1

Pendahuluan

1.1 Latar Belakang

Jaringan komputer hingga jaringan *Internet of Things* (IoT) terdiri dari perangkat yang saling berkomunikasi melalui media perantara atau kanal, baik perangkat yang membutuhkan daya listrik tinggi seperti *server* atau *storage* maupun perangkat IoT yang membutuhkan daya listrik kecil. Peningkatan *traffic* jaringan komputer yang bersifat masif namun kecil hingga kontinyu dan sensitif terhadap *latency* beserta berbagai macam paket yang dilewatkan mulai dari paket berukuran besar hingga paket berukuran kecil menuntut kendali protokol yang andal dan memiliki daya listrik yang tepat guna [1].
Pada sisi lain, perangkat simulasi jaringan komputer yang mendeskripsikan perilaku *traffic* pada masing-masing lapisan beserta dengan protokol yang digunakan sangat diperlukan untuk pengukuran akurasi dan efektifitas algoritma pada protokol yang akan diobservasi. Permasalahan beranjak dari lapisan fisik yang dapat diobservasi, yaitu kualitas transmisi dari media yang digunakan, dimana sinyal yang dikirimkan dapat terpengaruh oleh degradasi fisik media [2]. Pada lapisan fisik dengan media *wireless*, kinerja jaringan yang dinyatakan dengan perbandingan pengiriman paket dan *throughput* dapat terpengaruh oleh keberagaman *link* antara *wireless client* dengan *wireless access point* [3].
Kebutuhan simulator jaringan komputer yang bersifat modular, deskriptif, *multi-platform* berbasis pustaka grafis untuk dapat menjelaskan perilaku *traffic* pada masing-masing lapisan jaringan merupakan keniscayaan pada usulan algoritma dan protokol yang digunakan dalam rangka efisiensi waktu desain topologi. Keberagaman *link* antara *transmitter* dengan *receiver* menentukan pemenuhan persyaratan kualitas usulan desain jaringan mulai dari penggunaan pada jaringan *Internet of Things*, komputasi awan hingga *Software Defined Networking* [4].

1.2 Rumusan Masalah

Laporan ini mencoba membahas penggunaan simulator jaringan komputer dimulai dari lapisan fisik hingga lapisan *datalink* berbasis pustaka grafis menggunakan bahasa pemrograman Python yang bersifat *multi platform*, memiliki ragam obyek pada setiap lapisan dalam rangka mempelajari karakteristik dan perilaku *traffic* jaringan komputer pada kedua lapisan untuk memenuhi kebutuhan penyediaan simulator untuk meningkatkan efisiensi waktu desain jaringan komputer.

1.3 Batasan Masalah

Ruang lingkup pada pembahasan ini meliputi beberapa hal sebagai berikut, diantaranya adalah :

1. Pengembangan simulator SimPyNetLib v1.0 difokuskan pada *physical layer* dan *datalink layer* untuk mendapatkan karakteristik pemodelan pada masing-masing *layer* jaringan komputer.
2. Fokus bahasa pemrograman pengembangan simulator yang digunakan adalah Python berikut pustaka yang dibutuhkan seperti PyGObject, Numpy, Pandas, Matplotlib sehingga dapat dijalankan pada beberapa sistem operasi.

1.4 Tujuan Penelitian

Tujuan pengembangan simulator SimPyNetLib v1.0 adalah dalam rangka mendeskripsikan lapisan jaringan komputer kepada pengguna yang dapat berjalan pada banyak sistem operasi. Pada versi pertama ini pengembangan simulator difokuskan pada *physical layer* dan *datalink layer* untuk mendapatkan karakteristik pemodelan pada masing-masing *layer* dengan menggunakan bahasa pemrograman Python berikut pustaka yang dibutuhkan seperti PyGObject, Numpy, Pandas, Matplotlib untuk membentuk simulator lapisan jaringan komputer sehingga dapat dijalankan pada beberapa sistem operasi.

1.5 Sistematika Penulisan

Adapun sistematika penulisan pada laporan penelitian ini meliputi :

1. Bab 1 membahas latar belakang permasalahan dari kebutuhan simulator jaringan komputer yang bersifat deskriptif dan *multi-platform*. Selanjutnya merupakan perumusan masalah yang dibingkai berikut batasan masalah pembahasan agar dapat memenuhi tujuan penelitian yang diharapkan.
2. Bab 2 membahas konsep yang mendasari penelitian terkait gambaran arsitektur SimPyNetLib terdiri dari beberapa komponen SimPyNetLib, *Main Launcher, Physical Layer, Data link Layer, Benefit* SimPyNetLib dan *Drawbacks* penggunaan SimPyNetLib.
3. Bab 3 membahas tinjauan SimPyNetLib berupa fungsi-fungsi pada baris kode sumber yang digunakan sehingga terbentuk simulator tersebut.
4. Bab 4 membahas implementasi SimPyNetLib berupa hasil eksekusi simulator dalam tampilan grafis dan fasilitas *export* hasil simulasi dalam format *comma separated value* yang dapat diolah secara lebih lanjut.
5. Bab 5 memberikan kesimpulan dari hasil penelitian terkait simulator SimPyNetLib.

Bab 2

Gambaran Arsitektur SimPyNetlib

2.1 Komponen SimPyNetLib

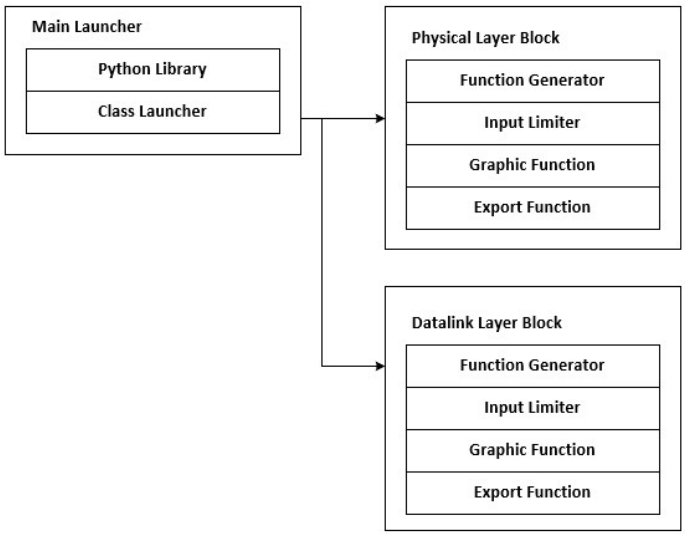
Pada pengembangan arsitektur simulator SimPyNetLib v1.0 terdiri dari beberapa komponen pembentuknya, seperti berikut ini:

1. Main Launcher
   (a) Python Library
   (b) Class Launcher
2. Physical Layer Block
   (a) Function Generator
   (b) Input Limiter
   (c) Graphic Function
   (d) Export Function
3. Datalink Layer Block
   (a) Function Generator
   (b) Input Limiter
   (c) Graphic Function
   (d) Export Function

Main Launcher merupakan aplikasi utama berbasis grafis yang akan memanggil aplikasi simulator yaitu Physical Layer Block dan Datalink Layer Block. Penggambaran arsitektur SimPyNetLib v1.0 dapat pembaca lihat pada gambar berikut ini:

2.2 Komponen *Main Launcher*

Aplikasi Main Launcher yang menjadi bagian dari SimPyNetLib v1.0 ini merupakan aplikasi untuk memanggil kedua aplikasi lain sesuai dengan fungsinya, yaitu: *Physical Layer Block* dan *Datalink Layer Block*.



Gambar 2.1: Arsitektur SimPyNetLib v1.0

2.2.1 Python Library

Pada aplikasi Main Launcher, pada file simpynetlib.py menggunakan beberapa Python Library, yaitu:

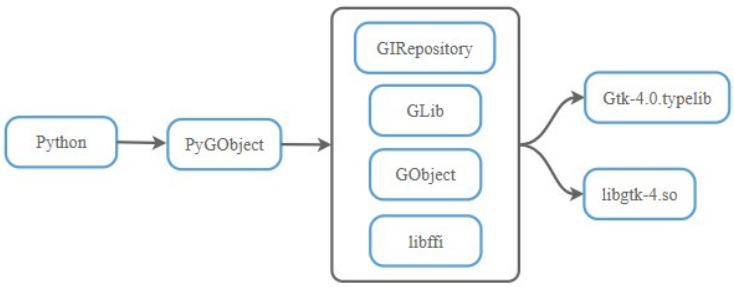
- PyGObject (gi)
- subprocess
- sys
- os

2.2.1.1 PyGObject

Pustaka **PyGObject** (https://pygobject.gnome.org) merupakan salah satu pustaka grafis pada bahasa pemrograman Python dalam bentuk paket untuk dapat bekerja pada GNOME *desktop environemnt* dengan pustaka berbasis GObject (https://docs.gtk.org/gobject/) seperti pustaka GTK (https://www.gtk.org), GStreamer (https://gstreamer.freedesktop.org/), WebKitGTK (https://webkitgtk.org/), GLib (https://docs.gtk.org/glib/) dan GIO (https://docs.gtk.org/gio/). Adapun cara kerja PyGObject adalah dengan menggunakan pustaka GLib, GObject, GIRepository, libffi beserta pustaka lainnya untuk mengakses pustaka bahasa pemrograman C yaitu pada file libgtk-4.so sebagai kombinasi dengan tambahan metadata dari typelib file (Gtk-4.0.typelib), sehingga dapat menyediakan antar muka Python secara dinamis berdasarkan informasi yang didapatkan [5]. Pembaca dapat melihat skema kerja PyGObject seperti pada gambar berikut ini: Beberapa contoh penggunaan PyGObject yaitu pada pengembangan *prototype* robot interaktif berbasis Zoomorphic dengan fitur pengenalan emosi dan interaksi suara untuk orang yang berusia lanjut [6]. Selain itu PyGObject telah menjadi salah satu obyek penelitian dalam rangka membandingkan efektifitas algoritma pemodelan *Labeled LatentDirichlet Allocation* (LLDA) sebagai penggunaan API (*Application Programming Interface*) [7].

2.2.1.2 Subprocess

Pustaka **subprocess** merupakan salah satu pustaka pada bahasa pemrograman Python yang dapat digunakan untuk menjalankan proses baru, membangun hubungan dengan *input, output*, dan menampilkan



Gambar 2.2: Cara Kerja pustaka PyGObject [5]

pesan *error*, serta menjalankan *return code*. Pustaka subprocess ini merupakan pustaka untuk menjalankan dan mengendalikan program eksternal dalam sebuah kode sumber Python. Beberapa fasilitas kunci untuk pustaka Subprocess adalah sebagai berikut [8]:

- Menjalankan dan mengendalikan proses eksternal;
- Menangkap aliran *standard output* dan *standard error*;
- Mendukung mekanisme *piping* menuju dan keluar proses;
- Melakukan mekanisme *process return codes*.

2.2.1.3 Sys

Pustaka **Sys** pada bahasa pemrograman Python menyediakan akses ke parameter dan fungsi yang bersifat spesifik terhadap sistem. Pustaka Sys memungkinkan pengembang aplikasi dapat berinteraksi dengan lingkungan runtime Python serta sistem operasi yang mendasari aplikasi Python tersebut. Beberapa fasilitas kunci untuk pustaka Sys adalah sebagai berikut [9]:

- Mengakses argumen baris perintah *command-line arguments* yang diberikan pada kode sumber Python;
- Memungkinkan pengembang aplikasi dapat berinteraksi dengan interpreter Python;
- Menyediakan perangkat untuk dapat memeriksa dan mengendalikan lingkungan *runtime* pada bahasa pemrograman Python;
- Memberikan akses aplikasi pada penggunaan standar *input, output,* dan *error*;
- Menyediakan alat bantu untuk melakukan pemeriksaan jalannya aplikasi secara mandiri.

2.2.1.4 os

Pustaka **os** pada bahasa pemrograman Python menyediakan berbagai alat bantu agar pengembang aplikasi dapat menggunakan fungsi yang bergantung pada sistem operasi, seperti membaca atau menulis ke sistem berkas (*file system*) dan lainnya. Pustaka os memungkinkan pengembang aplikasi untuk dapat berinteraksi dengan sistem operasi secara portabel. Beberapa fasilitas kunci untuk pustaka os dapat pembaca lihat pada bagian berikut [10]:

- Memungkinkan pengembang aplikasi dapat berinteraksi dengan sistem operasi;
- Menyediakan fasilitas untuk pengembang aplikasi agar dapat memanipulasi *file path* dan direktori;

- Menyediakan akses pengembang aplikasi kepada *environment variables*;
- Memberikan fasilitas pengembang aplikasi dalam pengelolaan proses.

Berikut ini merupakan listing kode sumber untuk bagian Python Library pada file simpynetlib.py:

Listing 2.1: Bagian Python Library pada Main Launcher

```
#!/usr/bin/python3
import gi
gi.require_version("Gtk", "3.0")
from gi.repository import Gtk
import subprocess
import sys
import os
```

Penjelasan mengenai baris kode sumber aplikasi pada bagian Python Library adalah sebagai berikut:

1. Baris pertama merupakan kode sumber untuk memicu interpreter Python versi 3 agar file Main Launcher simpynetlib.py dapat bersifat *executable*.
2. Baris kedua merupakan kode sumber untuk melakukan import pustaka PyGObject (gi).
3. Baris ketiga merupakan kode sumber yang mendefinisikan minimal versi Gtk 3.0 untuk dapat digunakan oleh pustaka PyGObject.
4. Baris keempat merupakan kode sumber untuk melakukan import fungsi Gtk dari pustaka PyGObject bagian *repository*.
5. Baris kelima hingga baris ketujuh merupakan kode sumber untuk melakukan import pustaka subprocess, sys, os.

2.2.2 Class Launcher

Kode sumber di bawah ini merupakan aplikasi *launcher* berbasis grafis dengan memanfaatkan pustaka GNOME *Tool Kit* (GTK) dalam bahasa pemrograman Python. Tujuan aplikasi ini adalah sebagai *launcher* aplikasi simulasi yang dikembangkan dengan bahasa pemrograman Python lainnya menggunakan area aplikasi dengan beberapa tombol. Aplikasi *launcher* ini menggunakan pustaka GTK melalui PyGObject untuk membangun *Graphical User Interface* (GUI) dan memanfaatkan pustaka sys, os dan subprocess untuk menjalankan aplikasi Python lainnya. Berikut ini merupakan listing kode sumber untuk bagian Class Launcher pada file simpynetlib.py:

Listing 2.2: Bagian Class Launcher pada Main Launcher

```
class Launcher(Gtk.Window):
    def __init__(self):
        super().__init__(title="SimPyNetLib ver1.0")
        self.set_default_size(300, 200)
        self.set_border_width(15)

        box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=10)
        self.add(box)

```

```
        # Button Datalink Layer Simulation
        button_a = Gtk.Button(label="Datalink Layer Simulation")
        button_a.connect("clicked", self.run_script, "final_datalink_gilbert_simulation.py
            ")
        box.pack_start(button_a, True, True, 0)

        # Button Physical Layer Simulation
        button_b = Gtk.Button(label="Physical Layer Simulation")
        button_b.connect("clicked", self.run_script, "
            final_physical_layer_snr_research_simulation.py")
        box.pack_start(button_b, True, True, 0)

        # Exit Button
        exit_button = Gtk.Button(label="Exit")
        exit_button.connect("clicked", self.on_exit_clicked)
        box.pack_start(exit_button, True, True, 0)

    def run_script(self, widget, script_name):
        python_exe = sys.executable
        script_path = os.path.join(os.path.dirname(__file__), script_name)

        if os.path.exists(script_path):
            subprocess.Popen([python_exe, script_path])
        else:
            print(f"{script_name} not found.")

    def on_exit_clicked(self, widget):
        Gtk.main_quit()

win = Launcher()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

Penjelasan mengenai baris kode sumber aplikasi pada bagian Class Main Launcher adalah sebagai berikut:

1. Baris ke-1 merupakan definisi Class dengan nama Launcher yang menurunkan sifat dari Gtk.Window, dimana Gtk.Window merupakan *container* utama untuk aplikasi GTK dan melalui penurunan sifat dari Class tersebut, memungkinkan pengembang untuk menuliskan kode sumber aplikasi dengan beberapa tambahan sifat.

2. Baris ke-2 dan ke-3, merupakan metode konstruktor untuk inisialisasi window. Pada baris ketiga berfungsi untuk melakukan panggilan pada obyek konstruktor diatasnya yaitu Gtk.Window dan nama windownya adalah "SimPyNetLib ver1.0".

3. Baris ke-4 dan ke-5 merupakan ukuran pada konfigurasi Window, yaitu berukuran 300x200 pixel. Pada baris kelima bertujuan untuk menambahkan 15 pixel *padding* antara sudut window dengan isi window.

4. Baris ke-7 dan ke-8 bertujuan untuk membuat tata letak *container* dengan Gtk.Box yaitu *widget* atau komponen grafis diatur secara vertikal (dari atas ke bawah) dan penambahan 10 pixel jeda antar *widget* serta penambahan kotak pada window.

5. Baris ke-11 sampai baris ke-13 merupakan pembuatan tombol dengan label Datalink Layer Simulation, koneksi sinyal yang digunakan pada GTK, dimana tombol diklik akan memanggil baris $self.run\_script(...)$ dan meneruskan ke file final_datalink_gilbert_simulation.py dilanjutkan dengan baris kesepuluh untuk menambahkan tombol pada layout window, dengan keterangan: $expand = True$ adalah obyek *widget* akan mengembang memenuhi ruang window, $fill = True$ adalah obyek *widget* akan mengisi area yang telah dialokasikan, $padding = 0$ adalah tidak menambahkan *padding* tambahan.

6. Baris ke-16 sampai baris ke-18 merupakan pembuatan tombol dengan label Physical Layer Simulation, dengan memanggil file final_physical_layer_snr_research_simulation.py dengan fungsi yang sama dengan barus kedelapan sampai barus kesepuluh.

7. Baris ke-21 sampai baris ke-23 merupakan pembuatan tombol dengan label Exit, ketika tombol Exit diklik maka akan memanggil baris perintah $self.on\_exit\_clicked()$ yang menutup aplikasi.

8. Baris ke-25 sampai baris ke-32 merupakan pendefinisian fungsi $run\_script()$ yang dipilih dengan tahapan mendapatkan interpreter Python, dilanjutkan dengan membangun aplikasi sesuai dengan lintasan tempat file kode sumber berada, lalu melakukan pemeriksaan apabila kode sumber tersebut tersedia dan menjalankan kode sumber serta penanganan *error*.

9. Baris ke-34 sampai baris ke-35 merupakan penanganan keluar dari aplikasim ketika tombol Exit diklik makan akan menjalankan kode sumber $Gtk.main\_quit()$ yang akan menghentikan perulangan *event* dan menutup aplikasi.

10. Baris ke-37 smpai baris ke-40 merupakan kode sumber untuk membuat obyek *class* Launcher yang menginisialisasi window Graphical User Interface (GUI), selanjutnya apabila pengguna aplikasi menutup window menggunakan tombol X pada *window manager* maka pustaka GTK akan mengirimkan sinyal *destroy* dan aplikasi akan tertutup, selanjutnya pada baris ke-39 akan menampilkan semua obyek *widget* dalam window dan baris ke-40 akan memulai perulangan pada pustaka GTK.

Pembaca dapat melihat gambar berikut ini merupakan tampilan aplikasi Main Launcher yang dijalankan pada sistem operasi GNU/Linux.

Gambar 2.3: Tampilan aplikasi Main Launcher

2.3 Komponen *Physical Layer*

Komponen *Physical Layer* merupakan aplikasi yang dikembangkan dengan bahasa pemrograman Python untuk mendeskripsikan simulasi pada *Physical Layer* dengan antar muka grafis dengan memanfaatkan pustaka PyGObject, pustaka grafis pengembangan lebih lanjut dari pustaka PyGTK. Berikut ini merupakan listing kode sumber pada file final_physical_layer_snr_research_simulation.py:

Listing 2.3: Physical Layer Simulation

```
import gi
gi.require_version("Gtk", "3.0")
from gi.repository import Gtk, GLib

import numpy as np
import pandas as pd
from scipy.special import erfc
from scipy.stats import bernoulli

import matplotlib
matplotlib.use("GTK3Agg")
from matplotlib.figure import Figure
from matplotlib.backends.backend_gtk3agg import FigureCanvasGTK3Agg as FigureCanvas

# ================================================
# SNR     BER ANALYTICAL MODEL (BPSK approx)
# ================================================
def snr_to_ber(snr_db):
    snr_lin = 10 ** (snr_db / 10)
    return 0.5 * erfc(np.sqrt(snr_lin))

# ================================================
# PHYSICAL LAYER RESEARCH MODEL
# ================================================
class PhysicalLayerModel:
    MEDIA = {
        "Fiber Optical": (10e9, 25),
        "Gigabit Ethernet": (1e9, 18),
        "Ethernet": (100e6, 12),
        "Wireless P2P": (50e6, 6),
    }

    def __init__(self, medium, rtt_ms):
        self.medium = medium
        self.rtt = rtt_ms / 1000
        self.time = 0
        self.history = []

    def step(self):
```

```
        results = []

        for name, (rate, snr_db) in self.MEDIA.items():
            if self.medium not in [name, "ALL"]:
                continue

            ber = snr_to_ber(snr_db)
           bits_sent = int(rate * self.rtt)
            error = bernoulli.rvs(ber)

            bits_rx = int(bits_sent * (0.75 if error else 1.0))
            throughput = bits_rx / self.rtt / 1e6

            record = {
                "RTT": self.time,
                "Medium": name,
                "Bits Sent": bits_sent,
                "Bits Rx": bits_rx,
                "Throughput (Mbps)": throughput,
                "SNR(dB)": snr_db,
                "BER": ber,
                "Error": bool(error)
            }

            results.append(record)
            self.history.append(record)

        self.time += 1
        return results

    def dataframe(self):
        return pd.DataFrame(self.history)
# ======================================================
# GTK RESEARCH SIMULATOR
# ======================================================
class Simulator(Gtk.Window):
    def __init__(self):
        super().__init__(title="Physical Layer Research Simulator ( S N R BER  + CRC)")
        self.set_default_size(1800, 1050)

        self.model = None
        self.running = False

        main = Gtk.Box(orientation=Gtk.Orientation.HORIZONTAL, spacing=10)
        self.add(main)

```

```
        # ---------------- CONTROLS ----------------
        control = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
        main.pack_start(control, False, False, 10)

        self.medium_combo = Gtk.ComboBoxText()
        for m in list(PhysicalLayerModel.MEDIA.keys()) + ["ALL"]:
            self.medium_combo.append_text(m)
        self.medium_combo.set_active(len(PhysicalLayerModel.MEDIA))

        self.rtt_entry = Gtk.Entry(text="10")

        buttons = {
            "Start": self.start,
            "Stop": self.stop,
            "Clear": self.clear,
            "Export CSV": self.export_csv,
            "Exit": lambda *_: Gtk.main_quit()
        }

        control.pack_start(Gtk.Label(label="Physical Medium"), False, False, 0)
        control.pack_start(self.medium_combo, False, False, 0)
        control.pack_start(Gtk.Label(label="RTT (ms)"), False, False, 0)
        control.pack_start(self.rtt_entry, False, False, 0)

        for name, cb in buttons.items():
            b = Gtk.Button(label=name)
            b.connect("clicked", cb)
            control.pack_start(b, False, False, 0)

        # ---------------- PLOTS ----------------
        plotbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL)
        main.pack_start(plotbox, True, True, 0)

        self.fig = Figure(figsize=(7, 10))
        self.ax1 = self.fig.add_subplot(311)
        self.ax2 = self.fig.add_subplot(312)
        self.ax3 = self.fig.add_subplot(313)
        self.canvas = FigureCanvas(self.fig)
        plotbox.pack_start(self.canvas, True, True, 0)

        # ---------------- TABLE ----------------
        self.store = Gtk.ListStore(int, str, int, int, float, float, float, bool)
        self.tree = Gtk.TreeView(self.store)

        headers = [
            "RTT", "Medium", "Bits Sent", "Bits Rx",
```

```
            "Throughput", "SNR(dB)", "BER", "Error"
        ]

        for i, h in enumerate(headers):
            self.tree.append_column(
                Gtk.TreeViewColumn(h, Gtk.CellRendererText(), text=i)
            )

        scroll = Gtk.ScrolledWindow()
        scroll.set_size_request(-1, 240)
        scroll.add(self.tree)
        plotbox.pack_start(scroll, False, False, 5)

    # ------------------------------------------------
    def start(self, *_):
        medium = self.medium_combo.get_active_text()
        rtt = float(self.rtt_entry.get_text())

        self.model = PhysicalLayerModel(medium, rtt)
        self.running = True
        GLib.timeout_add(700, self.run_step)

    def stop(self, *_):
        self.running = False

    def clear(self, *_):
        self.running = False
        self.store.clear()
        self.ax1.clear()
        self.ax2.clear()
        self.ax3.clear()
        self.canvas.draw()

    def run_step(self):
        if not self.running:
            return False

        results = self.model.step()
        df = self.model.dataframe()

        self.ax1.clear()
        self.ax2.clear()
        self.ax3.clear()

        for m in df["Medium"].unique():
            sub = df[df["Medium"] == m]
```

```
            self.ax1.plot(sub["RTT"], sub["Bits Sent"], marker="o", label=m)
            self.ax2.plot(sub["BER"], sub["Throughput (Mbps)"], marker="o", label=m)

        last = df.iloc[-1]
        prob = 0.95 if not last["Error"] else 0.55
        bits = np.random.choice([0, 1], 100, p=[1-prob, prob])
        self.ax3.bar(range(len(bits)), bits)

        self.ax1.set_title("Physical Traffic Load Evolution")
        self.ax2.set_title("Throughput vs BER (Analytical)")
        self.ax3.set_title("Bit-Level CRC Visualization")

        self.ax1.set_ylabel("Bits Sent")
        self.ax2.set_ylabel("Throughput (Mbps)")
        self.ax3.set_ylabel("Bit Value")

        self.ax1.legend()
        self.ax2.legend()

        self.canvas.draw()
        for r in results:
            self.store.append([
                r["RTT"], r["Medium"], r["Bits Sent"],
                r["Bits Rx"], round(r["Throughput (Mbps)"], 2),
                r["SNR(dB)"], r["BER"], r["Error"]
            ])

        return True

    # ------------------------------------------------
    def export_csv(self, *_):
        if not self.model:
            return

        dialog = Gtk.FileChooserDialog(
            title="Save CSV",
            parent=self,
            action=Gtk.FileChooserAction.SAVE,
            buttons=(
                Gtk.STOCK_CANCEL, Gtk.ResponseType.CANCEL,
                Gtk.STOCK_SAVE, Gtk.ResponseType.OK
            )
        )
        dialog.set_current_name("physical_layer_results.csv")

        if dialog.run() == Gtk.ResponseType.OK:
```

```
224             self.model.dataframe().to_csv(dialog.get_filename(), index=False)
225
226         dialog.destroy()
227
228
229 # ================================================
230 # RUN
231 # ================================================
232 if __name__ == "__main__":
233     win = Simulator()
234     win.connect("destroy", Gtk.main_quit)
235     win.show_all()
236     Gtk.main()
```

Berikut ini merupakan penjelasan mengenai baris kode sumber aplikasi pada bagian Physical Layer Simulation dengan menggunakan antarmuka grafis menggunakan pustaka PyGObject sebagai turunan dari PyGTK, dengan pemodelan analitis terhadap *Signal to Noise Ratio-Bit Error Rate* (SNR-BER) dengan visualisasi secara *realtime* menggunakan pustaka Matplotlib, adalah sebagai berikut:

1. Baris ke-1 hingga baris ke-13 merupakan instruksi import terhadap fungsi-fungsi pada pustaka PyGObject, numpy, pandas, scipy, matplotlib yang diperlukan aplikasi serta melakukan *load* GTK3 melalui antarmuka PyGObject.

2. Baris ke-18 hingga baris ke-20 merupakan fungsi untuk menghitung *Bit Error Rate* (BER) dari *Signal-to-Noise Ratio* (SNR), dengan tahapan melakukan konversi SNR dari dB ke skala linier, melakukan perhitungan BER dari modulasi *Bi-Phase Shift Keying* (BPSK).

3. Baris ke-25 hingga baris ke-31 merupakan pendefinisian class PhysicalLayerModel yang melakukan simulasi pengiriman data pada *physical layer*. Media yang digunakan adalah *fiber optic, gigabit ethernet, ethernet, wireless peer to peer* (P2P) berikut *data rate* dan SNR masing-masing, dimana nilai SNR lebih tinggi maka akan menurunkan nilai BER sehingga meningkatkan kehandalan media.

4. Baris ke-33 hingga baris ke-37 merupakan pendefinisian konstruktor dengan menggunakan beberapa parameter yaitu *medium, round trip time* dalam milidetik. Selanjutnya inisialisasi variabel dan perhitungan medium, round trip time, indeks waktu simulasi, penyimpanan semua hasil.

5. Baris ke-39 hingga baris ke-42 merupakan pendefinisian tahapan simulasi untuk satu siklus, dimulai dari iterasi melalui media, dilanjutkan dengan mekanisme *loop* melalui masing-masing medium yang telah didefinisikan.

6. Baris ke-43 hingga baris ke-44 merupakan filtering medium apabila pengguna memilih salah satu medium, maka pilihan medium lainnya dapat diabaikan.

7. Baris ke-46 sampai dengan baris ke-47 merupakan perhitungan *Bit Error Rate* (BER) dan bit-bit yang dikirimkan menggunakan rumus analitis.

8. Baris ke-48 hingga baris ke-51 merupakan simulasi bit error event dan membuat simulasi kesalahan pengiriman acak, serta perhitungan throughput. Apabila terjadi kesalahan maka terdapat data loss sebesar 25 persen, namun bila tidak terdapat kesalahan maka semua bit dapat diterima.

9. Baris ke-53 hingga baris ke-62 merupakan penyimpanan rekaman dalam variabel, dengan kelengkapan RTT adalah indeks waktu simulasi, Medium adalah media pengiriman, bit yang dikirimkan, bit yang diterima (Rx), *throughput* (dalam Mbps), SNR adalah kualitas sinyal, BER adalah probabilitas *error* serta kesalahan yang terjadi.

10. Baris ke-65 sampai baris ke-71 merupakan penyimpanan untuk plotting dan ekspor comma-separated-value (CSV) serta tampilan tabel pada aplikasi. Selanjutnya perhitungan waktu simulasi lanjutan yang dilakukan setiap siklus *round trip time* dan memasukkan proses sebelumnya kedalam DataFrame serta pustaka panda untuk analisis.

11. Baris ke-75 hingga baris ke-113 merupakan aplikasi simulator dengan antar muka grafis berbasis pustaka GTK berikut struktur layout aplikasi yaitu *Control Panel* dan *Plots* serta Tabel Data. Visualisasi yang ditampilkan adalah Bit yang dikirimkan vs waktu, *throughput* vs *Bit Error Rate* (BER), *Bit-level Cyclic Redundancy Check* (CRC).

12. Baris ke-116 hingga baris ke-124 merupakan pendefinisian plot untuk visualisasi grafik pada saat aplikasi simulator dijalankan.

13. Baris ke-176 hingga baris ke-179 merupakan bagian perulangan untuk plotting evolusi *traffic* untuk menunjukkan beban jaringan seiring berlalunya waktu dan plotting untuk memperlihatkan hubungan antara *throughput* vs *Bit Error Rate* (BER). Hubungan yang dapat terlihat adalah nilai BER lebih tinggi akan memperkecil nilai *Throughput*.

14. Baris ke-183 hingga baris ke-205 merupakan visualisasi Bit-level Cyclic Redundancy Check (CRC) untuk melakukan simulasi keandalan bentuk bit, apabila terjadi *error* maka akan menurunkan keandalan bentuk bit.

15. Baris ke-208 hingga baris ke-224 merupakan fungsi untuk melakukan ekspor hasil simulasi ke format Comma Separated Value (CSV) untuk kebutuhan analisis penelitian, statistik, plot menggunakan aplikasi lain.

16. Baris ke-232 hingga baris ke-236 merupakan *entry point application* dengan tahapan eksekusi: Membuat window (baris ke-233), menghubungkan window dengan event penutup (baris ke-234), menampilkan antar muka grafis (baris ke-235), memulai sirkular event pustaka GTK (baris ke-236).

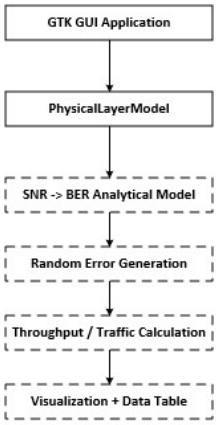
Sehingga arsitektur akhir dari aplikasi simulator untuk Physical Layer terlihat pada Gambar 2.4 sebagai berikut: Setelah aplikasi Physical Layer Simulator dijalankan, maka akan terlihat seperti terlihat pada Gambar 2.5 berikut ini: Untuk semua medium yang divisualisasikan, maka akan terlihat seperti pada Gambar 2.6 berikut ini:

2.4 Komponen *Datalink Layer*

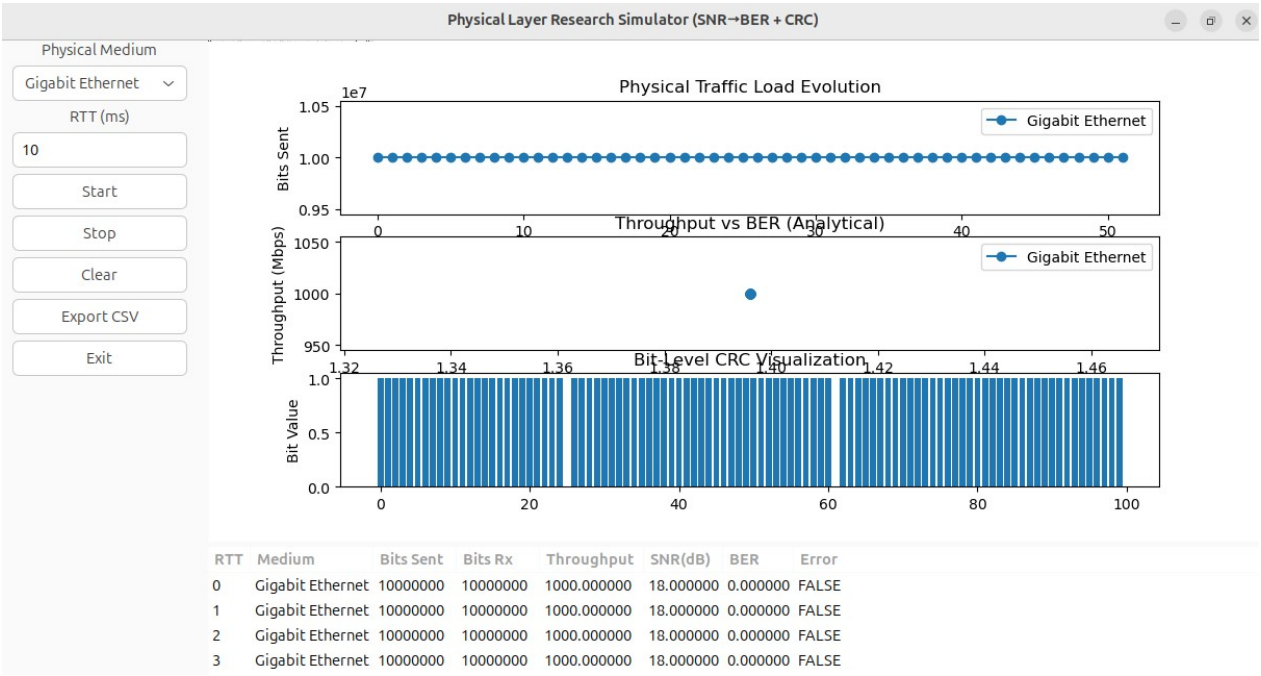
Komponen *Datalink Layer* merupakan aplikasi yang dikembangkan dengan bahasa pemrograman Python untuk mendeskripsikan simulasi pada *Datalink Layer* dengan antar muka grafis dengan memanfaatkan pustaka PyGObject, pustaka grafis pengembangan lebih lanjut dari pustaka PyGTK. Berikut ini merupakan listing kode sumber pada file final_datalink_gilbert_simulation.py:

Listing 2.4: Datalink Layer Simulation

```
1 import gi
2 gi.require_version("Gtk", "3.0")
```

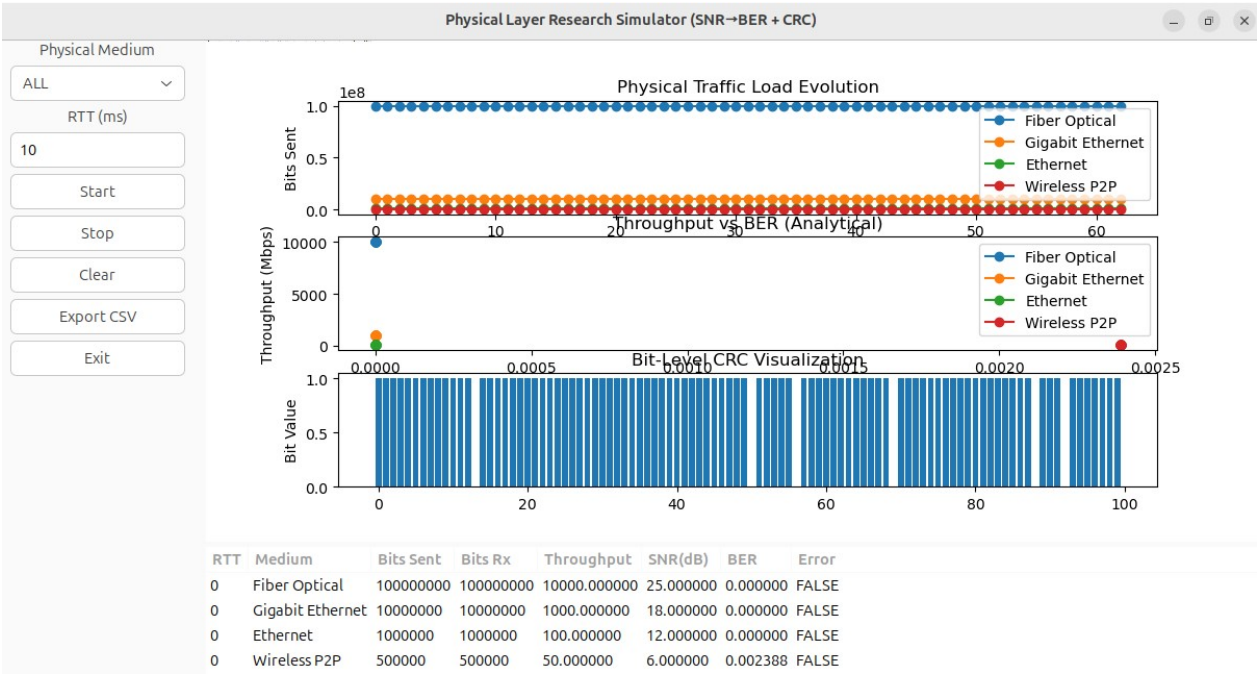


Gambar 2.4: Skema aplikasi Physical Layer Simulator



Gambar 2.5: Tampilan aplikasi Physical Layer Simulator untuk medium Gigabit Ethernet

```
from gi.repository import Gtk, GLib

import numpy as np
import pandas as pd
from scipy.stats import bernoulli

import matplotlib
matplotlib.use("GTK3Agg")
from matplotlib.figure import Figure
from matplotlib.backends.backend_gtk3agg import FigureCanvasGTK3Agg as FigureCanvas

# ============================================================
# GilbertElliott Burst Error Channel
```



Gambar 2.6: Tampilan aplikasi Physical Layer Simulator untuk semua medium

```
# ============================================================
class GilbertElliott:
    def __init__(self, p_good=0.995, p_bad=0.9, ber_good=1e-4, ber_bad=0.3):
        self.p_good = p_good
        self.p_bad = p_bad
        self.ber_good = ber_good
        self.ber_bad = ber_bad
        self.state = "GOOD"
   def step(self):
        if self.state == "GOOD":
            if np.random.rand() > self.p_good:
                self.state = "BAD"
        else:
            if np.random.rand() > self.p_bad:
                self.state = "GOOD"

        return self.ber_good if self.state == "GOOD" else self.ber_bad


# ============================================================
# Data Link Layer Research Model
# ============================================================
class DataLinkLayerModel:
    MODES = {
        "High-Speed LAN": (1e9, 1e-3),
        "Reliable Framed Link": (5e8, 5e-4),
        "Noisy Shared Medium": (1e8, 1e-1),
```

```
    }

    FRAME_OVERHEAD = 64  # bits

    def __init__(self, mode, rtt_ms):
        self.mode = mode
        self.rtt = rtt_ms / 1000
        self.time = 0
        self.history = []
        self.burst_model = GilbertElliott()

    def step(self):
        results = []

        for name, (rate, base_ber) in self.MODES.items():
            if self.mode not in [name, "ALL"]:
                continue

            if name == "Noisy Shared Medium":
                ber = self.burst_model.step()
            else:
                ber = base_ber

            payload_bits = int(rate * self.rtt)
            frame_bits = payload_bits + self.FRAME_OVERHEAD

            error = bernoulli.rvs(ber)

            bits_rx = int(frame_bits * (0.7 if error else 1.0))
            throughput = bits_rx / self.rtt / 1e6

            record = {
                "RTT": self.time,
                "Mode": name,
                "Frame Bits": frame_bits,
                "Bits Rx": bits_rx,
                "Throughput (Mbps)": throughput,
                "BER": ber,
                "Error": bool(error)
            }

            results.append(record)
            self.history.append(record)

        self.time += 1
        return results
```

```
89
90      def dataframe(self):
91          return pd.DataFrame(self.history)
92
93
94  # ============================================================
95  # SIMULATOR
96  # ============================================================
97  class Simulator(Gtk.Window):
98      def __init__(self):
99          super().__init__(title="Data Link Layer Research Simulator (GilbertElliott BER)
                ")
100         self.set_default_size(1800, 1050)
101
102         self.model = None
103         self.running = False
104
105         main = Gtk.Box(orientation=Gtk.Orientation.HORIZONTAL, spacing=10)
106         self.add(main)
107
108         # ---------------- CONTROLS ----------------
109         control = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
110         main.pack_start(control, False, False, 10)
111
112         self.mode_combo = Gtk.ComboBoxText()
113         for m in list(DataLinkLayerModel.MODES.keys()) + ["ALL"]:
114             self.mode_combo.append_text(m)
115         self.mode_combo.set_active(len(DataLinkLayerModel.MODES))
116
117         self.rtt_entry = Gtk.Entry(text="10")
118
119         buttons = {
120             "Start": self.start,
121             "Stop": self.stop,
122             "Clear": self.clear,
123             "Export CSV": self.export_csv,
124             "Exit": lambda *_: Gtk.main_quit()
125         }
126
127         control.pack_start(Gtk.Label(label="Link Mode"), False, False, 0)
128         control.pack_start(self.mode_combo, False, False, 0)
129        control.pack_start(Gtk.Label(label="RTT (ms)"), False, False, 0)
130         control.pack_start(self.rtt_entry, False, False, 0)
131
132         for name, cb in buttons.items():
133             b = Gtk.Button(label=name)
```

```
            b.connect("clicked", cb)
            control.pack_start(b, False, False, 0)

        # ---------------- PLOTS ----------------
        plotbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL)
        main.pack_start(plotbox, True, True, 0)

        self.fig = Figure(figsize=(7, 10))
        self.ax1 = self.fig.add_subplot(311)
        self.ax2 = self.fig.add_subplot(312)
        self.ax3 = self.fig.add_subplot(313)

        self.canvas = FigureCanvas(self.fig)
        plotbox.pack_start(self.canvas, True, True, 0)

        # ---------------- TABLE ----------------
        self.store = Gtk.ListStore(int, str, int, int, float, float, bool)
        self.tree = Gtk.TreeView(self.store)

        headers = [
            "RTT", "Mode", "Frame Bits",
           "Bits Rx", "Throughput", "BER", "Error"
        ]

        for i, h in enumerate(headers):
            self.tree.append_column(
                Gtk.TreeViewColumn(h, Gtk.CellRendererText(), text=i)
            )

        scroll = Gtk.ScrolledWindow()
        scroll.set_size_request(-1, 240)
        scroll.add(self.tree)
        plotbox.pack_start(scroll, False, False, 5)

    # ------------------------------------------------
    def start(self, *_):
        mode = self.mode_combo.get_active_text()
        rtt = float(self.rtt_entry.get_text())

        self.model = DataLinkLayerModel(mode, rtt)
        self.running = True
        GLib.timeout_add(700, self.run_step)

    def stop(self, *_):
        self.running = False

```

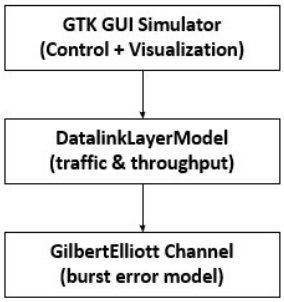
```
    def clear(self, *_):
        self.running = False
        self.store.clear()
        self.ax1.clear()
        self.ax2.clear()
        self.ax3.clear()
        self.canvas.draw()

    def run_step(self):
        if not self.running:
            return False

        results = self.model.step()
        df = self.model.dataframe()

        self.ax1.clear()
        self.ax2.clear()
        self.ax3.clear()

        for m in df["Mode"].unique():
            sub = df[df["Mode"] == m]
            self.ax1.plot(sub["RTT"], sub["Frame Bits"], marker="o", label=m)
            self.ax2.plot(sub["BER"], sub["Throughput (Mbps)"], marker="o", label=m)

        error_rate = df.groupby("RTT")["Error"].mean()
        self.ax3.plot(error_rate.index, error_rate.values, label="Frame Error Ratio")

        self.ax1.set_title("Data Link Traffic Load Evolution")
        self.ax2.set_title("Throughput vs BER")
        self.ax3.set_title("Burst Error Timeline (GilbertElliott)")

        self.ax1.set_ylabel("Frame Bits")
        self.ax2.set_ylabel("Throughput (Mbps)")
        self.ax3.set_ylabel("Error Probability")

        self.ax1.legend()
        self.ax2.legend()
        self.ax3.legend()

        self.canvas.draw()

        for r in results:
            self.store.append([
                r["RTT"], r["Mode"], r["Frame Bits"],
                r["Bits Rx"], round(r["Throughput (Mbps)"], 2),
                r["BER"], r["Error"]
```

```
            ])
        return True

    # ------------------------------------------------
    def export_csv(self, *_):
        if not self.model:
            return

        dialog = Gtk.FileChooserDialog(
            title="Save CSV",
            parent=self,
            action=Gtk.FileChooserAction.SAVE,
            buttons=(
                Gtk.STOCK_CANCEL, Gtk.ResponseType.CANCEL,
                Gtk.STOCK_SAVE, Gtk.ResponseType.OK
            )
        )
        dialog.set_current_name("datalink_gilbert_ber_results.csv")

        if dialog.run() == Gtk.ResponseType.OK:
            self.model.dataframe().to_csv(dialog.get_filename(), index=False)

        dialog.destroy()

# ============================================================
# RUN
# ============================================================
if __name__ == "__main__":
    win = Simulator()
    win.connect("destroy", Gtk.main_quit)
    win.show_all()
    Gtk.main()
```

Aplikasi ini adalah simulator Data Link Layer yang memodelkan *burst error channel* menggunakan model Gilbert–Elliott [11] (Markov channel) [12] dan menampilkan hasil simulasi dalam antar muka grafis pustaka GTK dan grafik dengan pustaka Matplotlib. Arsitektur aplikasi simulator Datalink Layer dapat kita lihat pada gambar berikut ini: Dimana GUI melakukan kontrol simulasi, DataLinkLayerModel melakukan penghitungan throughput, GilbertElliott menghasilkan *Bit Error Rate* (BER) dinamis. Berikut ini merupakan penjelasan kode sumber pada aplikasi simulator Datalink Layer:

2.4.1 Import Library

Beberapa pustaka Python yang digunakan yaitu: GTK untuk mendapatkan fungsi *Graphical User Interface* (GUI), Numpy untuk mewadahi angka acak, Pandas untuk penyimpanan data, SciPy untuk melakukan distribusi Bernoulli, Matplotlib untuk melakukan plotting grafik.



Gambar 2.7: Skema Datalink Layer Simulator

2.4.2 Gilbert-Elliott Channel

class GilbertElliott: digunakan untuk memberi simulasi *burst error* pada channel komunikasi. Pada Channel memilii 2 keadaan yaitu: GODD memiliki karakteristik *error* yang kecil, sedangkan keadaan BAD memiliki karakteristik *error* yang besar. Parameter model memiliki beberapa nilai, yaitu: p_good berarti probabilitas tetap pada kondisi GOOD, p_bad berarti probabilitas tetap pada kondisi BAD, ber_good berarti kondisi BER pada saat GOOD, ber_bad berarti kondisi BER pada saat BAD. Fungsi *step*() untuk melakukan transisi kondisi Markov, jika channel dalam kondisi GOOD, maka nilai variabel $P(G-G)adalahp\_good$, sedangkan nilai variabel $P(G-B)adalah1-p\_good$. Sedangkan apabila channel pada kondisi BAD, maka variabel $P(B-B)adalahp\_bad$, sedangkan nilai variabel $P(B-G) = 1-p\_bad$. Kemudian fungsi mengembalikan BER sesuai state.

2.4.3 Data Link Layer Model

Pada class DataLinkLayerModel menyimulasikan unjuk kerja Datalink Layer. Medium yang digunakan pada simulator Datalink Layer ini adalah *High-Speed LAN* dengan *Data Rate* 1 Gbps dan *Bit Error Rate* $1e-3$, *Reliable Framed Link* dengan *Data Rate* 500 Mbps dan *Bit Error Rate* $5e-4$ dan *Noisy Shared Medium* dengan *Data Rate* 100 Mbps dan *Bit Error Rate* $1e-1$. Mode terakhir menggunakan *burst error* Gilbert–Elliott.

2.4.4 Frame Overhead

Pada baris frame overhead terdapat satuan 64 bit untuk merepresentasikan *header frame data link layer.* Beberapa contoh isian pada kolom *header* adalah *Ethernet header, Cyclic Redundancy Check* (CRC), *control bits.*

2.4.5 Mekanisme Simulasi

Dinyatakan dengan fungsi utama: *step*(). Pada setiap langkah per satuan waktu simulasi terdapat mekanisme sebagai berikut:

- menentukan Bit Error Rate (BER),
- menghitung ukuran frame,
- menentukan apakah terjadi error,
- menghitung throughput

2.4.6 Payload

Pada baris dengan syntax: $payload\_bits = rateRTT$ dinyatakan variabel rate dengan nilai 1 Gbps dan RTT bernilai 10 ms. Sedangkan untuk menghitung *payload* dinyatakan dengan: $1x10^9x0.01 = 10Mb$

2.4.7 Frame Size dan Frame Error

Untuk melakukan penghitungan frame bit dinyatakan dengan formula: $frame\_bits = payload_bits + overhead$. Sedangkan eksekusi simulasi *error* menggunakan Bernoulli distribution, yang dinyatakan dengan baris: $error = bernoulli.rvs(ber)$, dengan 2 kemungkinan hasil yaitu:

- 1 = frame error
- 0 = frame sukses

2.4.8 Bits received

Pada *syntax* baris bit yang diterima, maka apabila terjadi *error*, maka formula yang digunakan adalah: $bits\_rx = 0.7frame\_bits$ Apabila tidak terjadi *error*, maka formula yang digunakan adalah: $bits\_rx = frame\_bits$

2.4.9 Throughput

Formula untuk melakukan penghitungan *throughput* adalah sebagai berikut: $Throughput = bits\_received/RTT$ Sehingga dijabarkan pada *syntax* baris kode sumber adalah sebagai berikut: $throughput = bits\_rx/rtt/1e6$ dengan nilai Output dalam satuan Mbps.

2.4.10 GUI Simulator

Memiliki class utama yaitu class: $Simulator(Gtk.Window)$ Layout Graphical User Interface (GUI) terdiri dari:

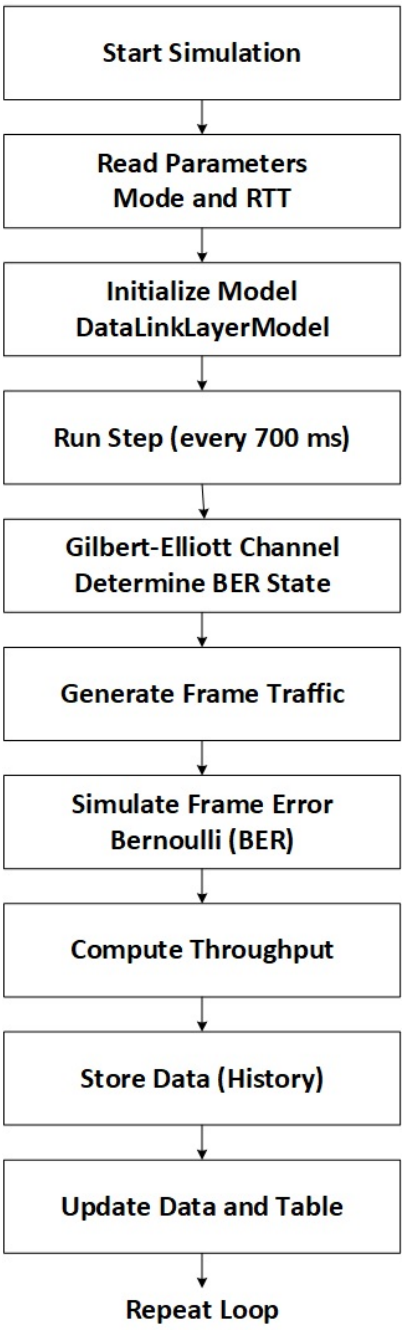
- *Mode : Traffic Load Graph*
- *RTT : Throughput vs Bit Error Rate (BER)*
- *Buttons : Burst error timeline*
- *Data table*

2.4.11 Grafik yang Ditampilkan

Pada Grafik 1 menampilkan *Data Link Traffic Load Evolution* yang direpresentasikan dengan tampilan *Frame Bits vs Time*. Pada Grafik 2 menampilkan *Throughput vs BER*, dalam rangka menunjukkan hubungan antara Error rate dengan Throughput, semakin tinggi BER maka throughput turun. Pada Grafik 3 menampilkan *Burst Error Timeline*, dalam rangka menunjukkan *Frame Error Ratio* terhadap waktu, untuk memperlihatkan *burst error pattern*.

2.4.12 Workflow Simulator

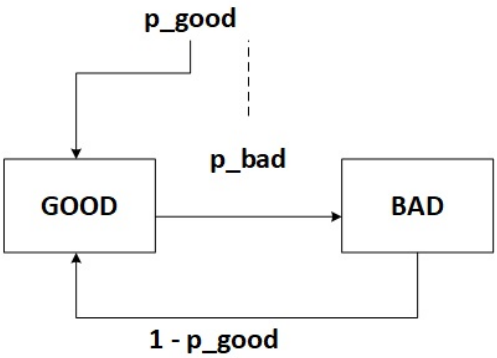
Pada gambar berikut ini merupakan alur kerja simulator:



Gambar 2.8: Alur Kerja Datalink Layer Simulator

2.4.13 State Diagram

Pada gambar berikut ini merupakan state diagram Datalink Layer simulator: Probabilitas transisi dapat dinotasikan sesuai dengan aturan berikut ini:
$P(G\text{ß}G) = p\_good$
$P(G\text{ß}B) = 1 \;\; p\_good$
$P(BtoB) = p\_bad$



Gambar 2.9: State Diagram pada Datalink Layer Simulator

$P(BtoG) = 1 \;\; p\_bad$
Bit Error Rate akan tergantung pada aturan:
Jika state GOOD maka $BER = ber\_good$
Jika state BAD maka $BER = ber\_bad$.
Rata-rata BER channel akan tergantung pada aturan:
Expected BER dinotasikan dengan :
$BER\_avg = G \times BER\_good + B \times BER\_bad$
di mana:
$G = steady - state\ probabilityGOOD$
$B = steady - state\ probabilityBAD$
Selanjutnya Probabilitas steady state dapat dinotasikan dengan keterangan berikut ini:
$G = (1 \;\; p\_bad)/(2 \;\; p\_good\ p\_bad)$
$B = (1 \;\; p\_good)/(2 \;\; p\_good\ p\_bad)$

2.4.14 Interpretasi Model

Pada pemodelan ini merepresentasikan kondisi nyata jaringan yaitu:

- Kondisi nyata: Wireless fading dengan state BAD.
- Kondisi nyata: Congestion dengan state BAD.
- Kondisi nyata: Interference dengan state BAD.
- Kondisi nyata: Normal transmission dengan state GOOD.

Sehingga pada pemodelan ini, ditemukan bahwa *error* sering terjadi dalam kondisi *burst*.

2.4.15 Output Dataset

Pada aplikasi simulator Datalink Layer dapat menghasilkan dataset berupa:

- Round Trip Time
- Mode
- Frame Bits
- Bits Rx

- Throughput
- Bit Error Rate
- Error

Bab 3

Kesimpulan

Pengembangan simulator SimPyNetLib v1.0 telah dapat mendeskripsikan simulasi lapisan jaringan komputer dalam hal ini *Physical Layer* dan *Datalink Layer* kepada pengguna dan dapat berjalan pada beberapa sistem operasi. Hasil yang didapatkan berupa karakteristik dan perilaku traffic untuk beberapa varian medium terkait pemodelan pada masing-masing *layer* dengan menggunakan bahasa pemrograman Python berikut pustaka yang dibutuhkan seperti PyGObject, Numpy, Pandas, Matplotlib.

Beberapa hal yang masih terbuka untuk dikembangkan lebih lanjut dari penelitian ini adalah pengembangan simulator untuk mendapatkan karakteristik dan perilaku traffic untuk beberapa layer selanjutnya dari OSI *Seven Layer* seperti Network Layer, Transport Layer, Session Layer, Presentation Layer dan Application Layer yang dapat berjalan pada banyak sistem operasi pengguna. Melalui pengembangan lebih lanjut dari penelitian simulator jaringan komputer ini, maka diharapkan terbuka penelitian lanjutan terkait *wireless channel modeling, satellite link simulation, network protocol.*

Bibliography

[1] S. Khisa and S. Moh, "Performance simulation tools for mac protocols in the internet of things," in *The 9th International Conference on Smart Media and Applications*, ser. SMA 2020. New York, NY, USA: Association for Computing Machinery, 2021, p. 28–33. [Online]. Available: https://doi.org/10.1145/3426020.3426026 3

[2] V. V. Garbhapu, C. Ware, and M. Lourdiane, "Physical-layer-aware network simulator for future optical functionalities," in *2023 14th International Conference on Network of the Future (NoF)*, 2023, pp. 28–36. 3

[3] M. F. Monir and T. A. Ishmam, "Exploiting link diversity in ieee 802.11 wlan using omnet++," in *2022 IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC)*, 2022, pp. 355–359. 3

[4] A. O. Nyanteh, M. Li, M. F. Abbod, and H. Al-Raweshidy, "Cloudsimhypervisor: Modeling and simulating network slicing in software-defined cloud networks," *IEEE Access*, vol. 9, pp. 72 484–72 498, 2021. 3

[5] Unknown, "Overview - pgobject," accessed December 19th, 2025. [Online]. Available: https://pygobject.gnome.org/index.html 6, 7

[6] B. Schnitzer, U. C. Vural, B. Schnitzer, M. U. Sardar, O. Fuerst, and O. Korn, "Prototyping a zoomorphic interactive robot companion with emotion recognition and affective voice interaction for elderly people," *Proc. ACM Hum.-Comput. Interact.*, vol. 8, no. EICS, Jun. 2024. [Online]. Available: https://doi.org/10.1145/3660244 6

[7] M. F. Zibran, "On the effectiveness of labeled latent dirichlet allocation in automatic bug-report categorization," in *Proceedings of the 38th International Conference on Software Engineering Companion*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 713–715. [Online]. Available: https://doi.org/10.1145/2889160.2892646 6

[8] R. P. T. Writers, "Subprocess - python standard library," accessed December 20th, 2025. [Online]. Available: https://realpython.com/ref/stdlib/subprocess/ 7

[9] ——, "Sys - python standard library," accessed December 20th, 2025. [Online]. Available: https://realpython.com/ref/stdlib/sys/ 7

[10] ——, "os - python standard library," accessed December 20th, 2025. [Online]. Available: https://realpython.com/ref/stdlib/os/ 7

[11] D. Ding, Z. Wang, Q.-L. Han, and X.-M. Zhang, "Recursive secure filtering over gilbert-elliott channels in sensor networks: The distributed case," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 7, pp. 75–86, 2021. 24

[12] Y. Fang and J. Chen, "Decoding polar codes for a generalized gilbert-elliott channel with unknown parameter," *IEEE Transactions on Communications*, vol. 69, no. 10, pp. 6455–6468, 2021. 24