Hasil Penelitian

Yang Tidak Dipublikasikan

SimPyNetLib v2.0: Simulator Pemodelan Lapisan Fisik, Datalink, Jaringan, Transpor dan Sesi untuk 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

2026

LEMBAR PENGESAHAN HASIL PENELITIAN YANG TIDAK DIPUBLIKASIKAN

1.	Judul Penelitian	: SimPyNetLib v2.0: Simulator Pemodelan Lapisan Fisik, Datalink, Jaringan, Transpor dan Sesi untuk 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 Juli 2026 – 28 Juli 2026

Jakarta, 29 Juli 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

Deskripsi cara kerja beberapa jaringan komputer dapat menentukan keberhasilan penggunaan algoritma pada suatu aplikasi simulasi untuk dapat menggambarkan perilaku *traffic* yang memiliki fitur lengkap dan memiliki akurasi tampilan data. Sebelum pengembangan simulasi, kondisi pengujian yang dilakukan secara langsung terhadap algoritma dan usulan model seringkali membutuhkan biaya tinggi serta waktu relatif lama. Masalah terkait portabilitas aplikasi simulasi pada banyak sistem operasi berakibat dalam penentuan besar atau kecilnya penerimaan pengguna atas usulan simulator dalam rangka analisis efektivitas penerapan model dan skema yang digunakan pada aplikasi simulasi jaringan. Pengembangan simulator SimPyNetLib v2.0 ditujukan untuk mendeskripsikan lapisan jaringan komputer kepada pengguna yang dapat berjalan pada banyak sistem operasi. Pada versi kedua ini pengembangan simulator difokuskan pada *network layer, transport layer* dan *session layer* dalam rangka penelusuran karakteristik pemodelan pada masing-masing *layer* 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 v2.0 yang dilakukan pada *environment* Ubuntu Linux versi stabil terakhir didapatkan bahwa simulator dapat berjalan dengan normal untuk masing-masing *network layer, transport layer and session 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 representasi *plotting* lainnya.

Daftar Isi

Bab 1

Pendahuluan

1.1 Latar Belakang

Kebutuhan infrastruktur teknologi informasi berupa jaringan komputer hingga jaringan Internet of Things (IoT) dibentuk oleh rangkaian perangkat yang saling berkomunikasi melalui media perantara atau kanal. Perangkat pembentuk infrastruktur memerlukan daya mulai dari daya listrik tinggi seperti *server* atau *storage* hingga perangkat IoT yang membutuhkan daya listrik kecil. Beberapa sifat peningkatan *traffic* jaringan komputer yang bersifat *converged* untuk dapat mengadopsi *traffic* masif namun kecil hingga *traffic* yang 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]. Sebagai alternatif solusi dari permasalahan tersebut, maka diperlukan perangkat simulasi jaringan komputer yang dapat mendeskripsikan perilaku *traffic* pada masing- masing lapisan beserta dengan protokol yang digunakan untuk mengukur 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 dan keberagaman protokol antara *wireless client* dengan *wireless access point* [3].

Pada lapisan *datalink* dengan sub lapisan *Medium Access Control* dan *Logical Link Control*, kinerja jaringan dapat dinyatakan dengan perbandingan antara kecepatan, kapasitas, latensi, *delay*, keandalan dan utilisasi, sedangkan pada lapisan *network* perbandingan antara parameter b*andwidth, throughput, latency, jitter, packet loss,* waktu proses, waktu antrian, waktu pengiriman, waktu penyebaran, waktu konvergensi lintasan, jumlah perangkat perantara dan *Maximum Transmission Unit* (MTU) dapat digunakan untuk menentukan kinerja jaringan [4].

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 *session* berbasis pustaka grafis menggunakan bahasa pemrograman Python yang bersifat *multi platform*, memiliki berbagai macam ragam obyek pada setiap lapisan dalam rangka mempelajari karakteristik dan perilaku *traffic* jaringan komputer pada kelima lapisan untuk memenuhi kebutuhan penyediaan simulator dalam rangka peningkatan efisiensi waktu desain tepat guna 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, datalink layer, network layer, transport layer* dan *session 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 pustaka PyGObject, Numpy, Pandas, Matplotlib sehingga dapat dijalankan pada beberapa sistem operasi.

1.4 Tujuan Penelitian

Tujuan pengembangan simulator SimPyNetLib v2.0 adalah dalam rangka mendeskripsikan lapisan jaringan komputer kepada pengguna untuk dapat berjalan pada banyak sistem operasi. Pada versi kedua, pengembangan simulator difokuskan pada *physical layer, datalink layer, network layer, transport layer* dan *session layer* untuk mendapatkan karakteristik pemodelan pada masing-masing *layer* dengan menggunakan bahasa pemrograman Python berikut pustaka yang dibutuhkan seperti pustaka 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* (CSV) yang dapat diolah secara lebih lanjut.
5. Bab 5 memberikan kesimpulan dari hasil penelitian terkait simulator SimPyNetLib versi 2.0.

Bab 2

Gambaran Arsitektur SimPyNetlib

2.1 Komponen SimPyNetLib

Pada pengembangan arsitektur simulator SimPyNetLib v2.0 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
4. Network Layer Block
    (a) Function Generator
    (b) Input Limiter
    (c) Graphic Function
    (d) Export Function
5. Transport Layer Block
    (a) Function Generator
    (b) Input Limiter

   (c) Graphic Function

   (d) Export Function

6. Session 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, Datalink Layer Block, Network Layer Block, Transport Layer Block* dan *Session Layer Block*. Penggambaran arsitektur SimPyNetLib v2.0 dapat pembaca lihat pada gambar 2.1 berikut ini:

2.2 Komponen *Main Launcher*

Aplikasi Main Launcher yang menjadi bagian dari SimPyNetLib v2.0 ini merupakan aplikasi untuk memanggil kelima aplikasi lain sesuai dengan fungsinya, yaitu: *Physical Layer Block, Datalink Layer Block, Network Layer Block, Transport Layer Block* dan *Session Layer Block*.

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 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)
```

```
14
15          # Button Physical Layer Simulation
16          button_b = Gtk.Button(label="Physical Layer Simulation")
17          button_b.connect("clicked", self.run_script, "
                final_physical_layer_snr_research_simulation.py")
18          box.pack_start(button_b, True, True, 0)
19
20          # Exit Button
21          exit_button = Gtk.Button(label="Exit")
22          exit_button.connect("clicked", self.on_exit_clicked)
23          box.pack_start(exit_button, True, True, 0)
24
25      def run_script(self, widget, script_name):
26          python_exe = sys.executable
27          script_path = os.path.join(os.path.dirname(__file__), script_name)
28
29          if os.path.exists(script_path):
30              subprocess.Popen([python_exe, script_path])
31          else:
32              print(f"{script_name} not found.")
33
34      def on_exit_clicked(self, widget):
35          Gtk.main_quit()
36
37  win = Launcher()
38  win.connect("destroy", Gtk.main_quit)
39  win.show_all()
40  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.

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

```
1  import gi
2  gi.require_version("Gtk", "3.0")
3  from gi.repository import Gtk, GLib
4
5  import numpy as np
6  import pandas as pd
7  from scipy.special import erfc
8  from scipy.stats import bernoulli
9
10 import matplotlib
11 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()
        }
```

```
104
105         control.pack_start(Gtk.Label(label="Physical Medium"), False, False, 0)
106         control.pack_start(self.medium_combo, False, False, 0)
107         control.pack_start(Gtk.Label(label="RTT (ms)"), False, False, 0)
108         control.pack_start(self.rtt_entry, False, False, 0)
109
110         for name, cb in buttons.items():
111             b = Gtk.Button(label=name)
112             b.connect("clicked", cb)
113             control.pack_start(b, False, False, 0)
114
115         # ---------------- PLOTS ----------------
116         plotbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL)
117         main.pack_start(plotbox, True, True, 0)
118
119         self.fig = Figure(figsize=(7, 10))
120         self.ax1 = self.fig.add_subplot(311)
121         self.ax2 = self.fig.add_subplot(312)
122         self.ax3 = self.fig.add_subplot(313)
123         self.canvas = FigureCanvas(self.fig)
124         plotbox.pack_start(self.canvas, True, True, 0)
125
126         # ---------------- TABLE ----------------
127         self.store = Gtk.ListStore(int, str, int, int, float, float, float, bool)
128         self.tree = Gtk.TreeView(self.store)
129
130         headers = [
131             "RTT", "Medium", "Bits Sent", "Bits Rx",
132             "Throughput", "SNR(dB)", "BER", "Error"
133         ]
134
135         for i, h in enumerate(headers):
136             self.tree.append_column(
137                 Gtk.TreeViewColumn(h, Gtk.CellRendererText(), text=i)
138             )
139
140         scroll = Gtk.ScrolledWindow()
141         scroll.set_size_request(-1, 240)
142         scroll.add(self.tree)
143         plotbox.pack_start(scroll, False, False, 5)
144
145     # ------------------------------------------------
146     def start(self, *_):
147         medium = self.medium_combo.get_active_text()
148         rtt = float(self.rtt_entry.get_text())
149
```

```
        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()
```

```
196
197         self.canvas.draw()
198         for r in results:
199             self.store.append([
200                 r["RTT"], r["Medium"], r["Bits Sent"],
201                 r["Bits Rx"], round(r["Throughput (Mbps)"], 2),
202                 r["SNR(dB)"], r["BER"], r["Error"]
203             ])
204
205         return True
206
207     # -----------------------------------------------
208     def export_csv(self, *_):
209         if not self.model:
210             return
211
212         dialog = Gtk.FileChooserDialog(
213             title="Save CSV",
214             parent=self,
215             action=Gtk.FileChooserAction.SAVE,
216             buttons=(
217                 Gtk.STOCK_CANCEL, Gtk.ResponseType.CANCEL,
218                 Gtk.STOCK_SAVE, Gtk.ResponseType.OK
219             )
220         )
221         dialog.set_current_name("physical_layer_results.csv")
222
223         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")
3  from gi.repository import Gtk, GLib
4
5  import numpy as np
6  import pandas as pd
7  from scipy.stats import bernoulli
8
9  import matplotlib
10 matplotlib.use("GTK3Agg")
11 from matplotlib.figure import Figure
12 from matplotlib.backends.backend_gtk3agg import FigureCanvasGTK3Agg as FigureCanvas
13
14 # ============================================================
15 # GilbertElliott Burst Error Channel
16 # ============================================================
17 class GilbertElliott:
18     def __init__(self, p_good=0.995, p_bad=0.9, ber_good=1e-4, ber_bad=0.3):
19         self.p_good = p_good
20         self.p_bad = p_bad
21         self.ber_good = ber_good
22         self.ber_bad = ber_bad
23         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)
```

```
70
71              bits_rx = int(frame_bits * (0.7 if error else 1.0))
72              throughput = bits_rx / self.rtt / 1e6
73
74              record = {
75                  "RTT": self.time,
76                  "Mode": name,
77                  "Frame Bits": frame_bits,
78                  "Bits Rx": bits_rx,
79                  "Throughput (Mbps)": throughput,
80                  "BER": ber,
81                  "Error": bool(error)
82              }
83
84              results.append(record)
85              self.history.append(record)
86
87          self.time += 1
88          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)
```

```
        self.mode_combo.set_active(len(DataLinkLayerModel.MODES))

        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="Link Mode"), False, False, 0)
        control.pack_start(self.mode_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, 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
# ============================================================
```

```
253 if __name__ == "__main__":
254     win = Simulator()
255     win.connect("destroy", Gtk.main_quit)
256     win.show_all()
257     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.

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 $P(G-G) adalah p\_good$, sedangkan nilai variabel $P(G-B) adalah 1-p\_good$. Sedangkan apabila channel pada kondisi BAD, maka variabel $P(B-B) adalah p\_bad$, 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:

2.4.13 State Diagram

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

2.5 Komponen *Network Layer*

Komponen *Network Layer* merupakan aplikasi yang dikembangkan dengan bahasa pemrograman Python untuk mendeskripsikan simulasi pada *Network 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_network_layer_research_simulation.py:

Listing 2.5: Network Layer Simulation

```
#!/usr/bin/env python3
"""
NETWORK LAYER SIMULATION
========================
Python 3
GTK3 / PyGObject
NumPy
Pandas
Matplotlib

Features
--------
1. IPv4 packet generation
2. Routing-table lookup
3. Longest Prefix Match
4. Next-hop forwarding
5. TTL processing
6. Packet loss
7. Variable network delay
8. Packet delivery
9. Packet drop
10. Live simulation plotting
11. Line graphs
12. Cumulative statistics
13. PDR
14. Throughput
15. Average latency
16. Average hop count
17. Packet status analysis
18. Routing table visualization
19. Packet log
20. CSV export
Run
---
python3 final_network_layer_research_simulation.py
```

```
"""
import gi
gi.require_version("Gtk", "3.0")
from gi.repository import Gtk, GLib
import numpy as np
import pandas as pd
import matplotlib
matplotlib.use("GTK3Agg")
from matplotlib.figure import Figure
from matplotlib.backends.backend_gtk3agg import FigureCanvasGTK3Agg as FigureCanvas
import random
import time
import threading
import ipaddress
# ============================================================
# IPv4 PACKET
# ============================================================
class IPv4Packet:
    def __init__(
        self,
        packet_id,
        source,
        destination,
        payload_size,
        ttl
    ):
        self.packet_id = packet_id
        self.source = source
        self.destination = destination
        self.payload_size = payload_size
        self.ttl = ttl
        self.original_ttl = ttl
        self.next_hop = None
        self.status = "CREATED"
        self.hop_count = 0
        self.delay_ms = 0.0
        self.created_time = time.time()
        self.route = []
        self.delivered_time = None
# ============================================================
# ROUTING TABLE
# ============================================================
class RoutingTable:
    def __init__(self):
        self.routes = [
            {
```

```
                "network": "10.0.1.0/24",
                "next_hop": "10.0.1.1",
                "interface": "eth0",
                "metric": 1
            },
            {
                "network": "10.0.2.0/24",
                "next_hop": "10.0.2.1",
                "interface": "eth1",
                "metric": 1
            },
            {
                "network": "10.0.3.0/24",
                "next_hop": "10.0.3.1",
                "interface": "eth2",
                "metric": 2
            },
            {
                "network": "192.168.1.0/24",
                "next_hop": "192.168.1.1",
                "interface": "eth3",
                "metric": 1
            },
            {
                "network": "172.16.0.0/16",
                "next_hop": "172.16.0.1",
                "interface": "eth4",
                "metric": 2
            },
            {
                "network": "0.0.0.0/0",
                "next_hop": "192.168.1.254",
                "interface": "eth5",
                "metric": 10
            }
        ]
    def lookup(self, destination_ip):
        destination = ipaddress.ip_address(
            destination_ip
        )
        matches = []
        for route in self.routes:
            network = ipaddress.ip_network(
                route["network"],
                strict=False
            )
```

```
            if destination in network:
                matches.append(
                    (
                        network.prefixlen,
                        route
                    )
                )
        if not matches:
            return None
        matches.sort(
            key=lambda item: (
                item[0],
                -item[1]["metric"]
            ),
            reverse=True
        )
        return matches[0][1]
    def dataframe(self):
        return pd.DataFrame(
            self.routes
        )
# ============================================================
# NETWORK LAYER SIMULATOR
# ============================================================
class NetworkLayerSimulator:
    def __init__(
        self,
        packet_count=100,
        packet_size=1024,
        ttl=16,
        loss_probability=0.05,
        min_delay=5,
        max_delay=50
    ):
        self.packet_count = packet_count
        self.packet_size = packet_size
        self.default_ttl = ttl
        self.loss_probability = loss_probability
        self.min_delay = min_delay
        self.max_delay = max_delay
        self.routing_table = RoutingTable()
        self.packets = []
        self.statistics = []
        self.running = False
        self.start_time = None
        self.end_time = None
```

```
        self.generated = 0
        self.forwarded = 0
        self.delivered = 0
        self.dropped = 0
        self.ttl_expired = 0
        self.no_route = 0
        self.total_delay = 0.0
        self.total_hops = 0
    # ============================================================
    # RESET
    # ============================================================
    def reset(self):
        self.packets = []
        self.statistics = []
        self.running = False
        self.start_time = None
        self.end_time = None
        self.generated = 0
        self.forwarded = 0
        self.delivered = 0
        self.dropped = 0
        self.ttl_expired = 0
        self.no_route = 0
        self.total_delay = 0.0
        self.total_hops = 0
    # ============================================================
    # CREATE PACKET
    # ============================================================
    def create_packet(self, packet_id):
        source_networks = [
            "10.0.1.",
            "10.0.2.",
            "10.0.3.",
            "192.168.1."
        ]
        destination_networks = [
            "10.0.1.",
            "10.0.2.",
            "10.0.3.",
            "192.168.1.",
            "172.16.1.",
            "172.16.2.",
            "8.8.8."
        ]
        source_network = random.choice(
            source_networks
```

```
        )
        destination_network = random.choice(
            destination_networks
        )
        source = (
            source_network +
            str(random.randint(2, 250))
        )
        destination = (
            destination_network +
            str(random.randint(2, 250))
        )
        return IPv4Packet(
            packet_id=packet_id,
            source=source,
            destination=destination,
            payload_size=self.packet_size,
            ttl=self.default_ttl
        )
    # ============================================================
    # FORWARD PACKET
    # ============================================================
    def forward_packet(self, packet):
        packet.status = "ROUTING"
        route = self.routing_table.lookup(
            packet.destination
        )
        # ------------------------------------------------
        # NO ROUTE
        # ------------------------------------------------
        if route is None:
            packet.status = "NO_ROUTE"
            self.no_route += 1
            self.dropped += 1
            return packet
        packet.next_hop = route["next_hop"]
        packet.route.append(
            route["next_hop"]
        )
        # ------------------------------------------------
        # FIRST HOP
        # ------------------------------------------------
        if packet.ttl <= 1:
            packet.status = "TTL_EXPIRED"
            self.ttl_expired += 1
            self.dropped += 1
```

```
            return packet
        packet.ttl -= 1
        packet.hop_count += 1
        self.forwarded += 1
        # ----------------------------------------------------
        # PACKET LOSS
        # ----------------------------------------------------
        if random.random() < self.loss_probability:
            packet.status = "LOST"
            self.dropped += 1
            return packet
        # ----------------------------------------------------
        # MULTI-HOP ROUTING
        # ----------------------------------------------------
        number_of_hops = random.randint(
            1,
            5
        )
        for _ in range(
            number_of_hops
        ):
            if packet.ttl <= 1:
                packet.status = "TTL_EXPIRED"
                self.ttl_expired += 1
                self.dropped += 1
                return packet
            packet.ttl -= 1
            packet.hop_count += 1
            hop_delay = random.uniform(
                self.min_delay,
                self.max_delay
            )
            packet.delay_ms += hop_delay
        # ----------------------------------------------------
        # FINAL DELAY
        # ----------------------------------------------------
        packet.delay_ms += random.uniform(
            self.min_delay,
            self.max_delay
        )
        packet.status = "DELIVERED"
        packet.delivered_time = time.time()
        self.delivered += 1
        self.total_delay += packet.delay_ms
        self.total_hops += packet.hop_count
        return packet
```

```
    # ======================================================
    # RUN
    # ======================================================
    def run(
        self,
        progress_callback=None,
        packet_callback=None
    ):
        self.reset()
        self.running = True
        self.start_time = time.time()
        for packet_id in range(
            1,
            self.packet_count + 1
        ):
            if not self.running:
                break
            packet = self.create_packet(
                packet_id
            )
            self.generated += 1
            packet = self.forward_packet(
                packet
            )
            self.packets.append(
                packet
            )
            elapsed = (
                time.time() -
                self.start_time
            )
            # ------------------------------------------------
            # RECORD STATISTICS
            # ------------------------------------------------
            self.statistics.append(
                {
                    "packet_id":
                        packet.packet_id,
                    "source":
                        packet.source,
                    "destination":
                        packet.destination,
                    "payload_bytes":
                        packet.payload_size,
                    "status":
                        packet.status,
```

```
                    "next_hop":
                        packet.next_hop
                        or "-",
                    "ttl":
                        packet.ttl,
                    "hop_count":
                        packet.hop_count,
                    "delay_ms":
                        packet.delay_ms,
                    "elapsed_seconds":
                        elapsed,
                    "generated":
                        self.generated,
                    "delivered":
                        self.delivered,
                    "dropped":
                        self.dropped,
                    "forwarded":
                        self.forwarded,
                    "pdr":
                        (
                            self.delivered /
                            self.generated
                        ) * 100.0,
                    "throughput_bps":
                        (
                            self.delivered *
                            self.packet_size *
                            8
                        ) /
                        elapsed
                        if elapsed > 0
                        else 0
                }
            )
            if progress_callback:
                progress_callback(
                    self.generated,
                    self.packet_count
                )
            if packet_callback:
                packet_callback(
                    packet
                )
            # Small delay so that live plotting
            # can be observed.
```

```
            time.sleep(
                0.03
            )
        self.end_time = time.time()
        self.running = False
    # ========================================================
    # STOP
    # ========================================================
    def stop(self):
        self.running = False
    # ========================================================
    # DATAFRAME
    # ========================================================
    def dataframe(self):
        return pd.DataFrame(
            self.statistics
        )
    # ========================================================
    # METRICS
    # ========================================================
    def get_metrics(self):
        elapsed = 0
        if (
            self.start_time is not None
            and self.end_time is not None
        ):
            elapsed = (
                self.end_time -
                self.start_time
            )
        pdr = 0
        if self.generated > 0:
            pdr = (
                self.delivered /
                self.generated
            ) * 100
        average_delay = 0
        if self.delivered > 0:
            average_delay = (
                self.total_delay /
                self.delivered
            )
        average_hops = 0
        if self.delivered > 0:
            average_hops = (
                self.total_hops /
```

```
                self.delivered
            )
        throughput = 0
        if elapsed > 0:
            throughput = (
                self.delivered *
                self.packet_size *
                8
            ) / elapsed
        return {
            "generated":
                self.generated,
            "forwarded":
                self.forwarded,
            "delivered":
                self.delivered,
            "dropped":
                self.dropped,
            "ttl_expired":
                self.ttl_expired,
            "no_route":
                self.no_route,
            "pdr":
                pdr,
            "average_delay":
                average_delay,
            "average_hops":
                average_hops,
            "throughput":
                throughput,
            "elapsed":
                elapsed
        }
# ============================================================
# GTK APPLICATION
# ============================================================
class NetworkLayerApplication(Gtk.Window):
    def __init__(self):
        super().__init__(
            title="Network Layer Simulation"
        )
        self.set_default_size(
            1400,
            900
        )
        self.set_border_width(
```

```
            10
        )
        self.simulator = (
            NetworkLayerSimulator()
        )
        self.worker_thread = None
        self.build_interface()
    # ========================================================
    # BUILD INTERFACE
    # ========================================================
    def build_interface(self):
        main_box = Gtk.Box(
            orientation=Gtk.Orientation.VERTICAL,
            spacing=8
        )
        self.add(
            main_box
        )
        # --------------------------------------------------
        # TITLE
        # --------------------------------------------------
        title = Gtk.Label()
        title.set_markup(
            "<span size='xx-large' "
            "weight='bold'>"
            "IPv4 Network Layer Simulation"
            "</span>"
        )
        main_box.pack_start(
            title,
            False,
            False,
            5
        )
        subtitle = Gtk.Label(
            label=(
                "Packet generation     routing     "
                "forwarding     TTL     loss     "
                "delivery"
            )
        )
        main_box.pack_start(
            subtitle,
            False,
            False,
            2
```

```
        )
        # ------------------------------------------------
        # PARAMETERS
        # ------------------------------------------------
        frame = Gtk.Frame(
            label="Simulation Parameters"
        )
        grid = Gtk.Grid()
        grid.set_border_width(
            8
        )
        grid.set_row_spacing(
            6
        )
        grid.set_column_spacing(
            8
        )
        frame.add(
            grid
        )
        main_box.pack_start(
            frame,
            False,
            False,
            0
        )
        # Packet count
        grid.attach(
            Gtk.Label(
                label="Packets:"
            ),
            0, 0, 1, 1
        )
        self.packet_entry = Gtk.Entry()
        self.packet_entry.set_text(
            "100"
        )
        grid.attach(
            self.packet_entry,
            1, 0, 1, 1
        )
        # Packet size
        grid.attach(
            Gtk.Label(
                label="Packet Size:"
            ),
```

```
            2, 0, 1, 1
        )
        self.size_entry = Gtk.Entry()
        self.size_entry.set_text(
            "1024"
        )
        grid.attach(
            self.size_entry,
            3, 0, 1, 1
        )
        # TTL
        grid.attach(
            Gtk.Label(
                label="TTL:"
            ),
            4, 0, 1, 1
        )
        self.ttl_entry = Gtk.Entry()
        self.ttl_entry.set_text(
            "16"
        )
        grid.attach(
            self.ttl_entry,
            5, 0, 1, 1
        )
        # Loss
        grid.attach(
            Gtk.Label(
                label="Loss Probability:"
            ),
            0, 1, 1, 1
        )
        self.loss_entry = Gtk.Entry()
        self.loss_entry.set_text(
            "0.05"
        )
        grid.attach(
            self.loss_entry,
            1, 1, 1, 1
        )
        # Min delay
        grid.attach(
            Gtk.Label(
                label="Min Delay:"
            ),
            2, 1, 1, 1
```

```
        )
        self.min_delay_entry = Gtk.Entry()
        self.min_delay_entry.set_text(
            "5"
        )
        grid.attach(
            self.min_delay_entry,
            3, 1, 1, 1
        )
        # Max delay
        grid.attach(
            Gtk.Label(
                label="Max Delay:"
            ),
            4, 1, 1, 1
        )
        self.max_delay_entry = Gtk.Entry()
        self.max_delay_entry.set_text(
            "50"
        )
        grid.attach(
            self.max_delay_entry,
            5, 1, 1, 1
        )
        # ------------------------------------------------
        # BUTTONS
        # ------------------------------------------------
        buttons = Gtk.Box(
            orientation=Gtk.Orientation.HORIZONTAL,
            spacing=8
        )
        self.start_button = Gtk.Button(
            label="Start Simulation"
        )
        self.start_button.connect(
            "clicked",
            self.start_simulation
        )
        buttons.pack_start(
            self.start_button,
            False,
            False,
            0
        )
        self.stop_button = Gtk.Button(
            label="Stop"
```

```
        )
        self.stop_button.connect(
            "clicked",
            self.stop_simulation
        )
        buttons.pack_start(
            self.stop_button,
            False,
            False,
            0
        )
        export_button = Gtk.Button(
            label="Export CSV"
        )
        export_button.connect(
            "clicked",
            self.export_csv
        )
        buttons.pack_start(
            export_button,
            False,
            False,
            0
        )
        routing_button = Gtk.Button(
            label="Routing Table"
        )
        routing_button.connect(
            "clicked",
            self.show_routing_table
        )
        buttons.pack_start(
            routing_button,
            False,
            False,
            0
        )
        grid.attach(
            buttons,
            0, 2, 6, 1
        )
        # ---------------------------------------------------
        # PROGRESS
        # ---------------------------------------------------
        self.progress = Gtk.ProgressBar()
        main_box.pack_start(
```

```
            self.progress,
            False,
            False,
            0
        )
        # ------------------------------------------------------
        # METRICS
        # ------------------------------------------------------
        metrics_frame = Gtk.Frame(
            label="Simulation Metrics"
        )
        metrics_grid = Gtk.Grid()
        metrics_grid.set_border_width(
            8
        )
        metrics_grid.set_column_spacing(
            15
        )
        metrics_grid.set_row_spacing(
            5
        )
        metrics_frame.add(
            metrics_grid
        )
        main_box.pack_start(
            metrics_frame,
            False,
            False,
            0
        )
        self.metric_labels = {}
        metric_names = [
            "Generated",
            "Forwarded",
            "Delivered",
            "Dropped",
            "PDR",
            "Avg Delay",
            "Avg Hops",
            "Throughput"
        ]
        metric_keys = [
            "generated",
            "forwarded",
            "delivered",
            "dropped",
```

```
            "pdr",
            "average_delay",
            "average_hops",
            "throughput"
        ]
        for i in range(
            len(metric_names)
        ):
            box = Gtk.Box(
                orientation=Gtk.Orientation.VERTICAL,
                spacing=2
            )
            caption = Gtk.Label(
                label=metric_names[i]
            )
            value = Gtk.Label(
                label="0"
            )
            value.set_markup(
                "<b>0</b>"
            )
            box.pack_start(
                caption,
                False,
                False,
                0
            )
            box.pack_start(
                value,
                False,
                False,
                0
            )
            metrics_grid.attach(
                box,
                i,
                0,
                1,
                1
            )
            self.metric_labels[
                metric_keys[i]
            ] = value
        # ----------------------------------------------------
        # NOTEBOOK
        # ----------------------------------------------------
```

```
        notebook = Gtk.Notebook()
        main_box.pack_start(
            notebook,
            True,
            True,
            0
        )
        # ====================================================
        # LIVE GRAPH TAB
        # ====================================================
        graph_box = Gtk.Box(
            orientation=Gtk.Orientation.VERTICAL
        )
        notebook.append_page(
            graph_box,
            Gtk.Label(
                label="Live Simulation Graph"
            )
        )
        self.figure = Figure(
            figsize=(12, 8),
            dpi=100
        )
        self.canvas = FigureCanvas(
            self.figure
        )
        graph_box.pack_start(
            self.canvas,
            True,
            True,
            0
        )
        # ====================================================
        # PACKET LOG TAB
        # ====================================================
        packet_box = Gtk.Box(
            orientation=Gtk.Orientation.VERTICAL
        )
        notebook.append_page(
            packet_box,
            Gtk.Label(
                label="Packet Log"
            )
        )
        self.packet_store = Gtk.ListStore(
            int,
```

```
            str,
            str,
            int,
            str,
            str,
            int,
            int,
            float
        )
        tree = Gtk.TreeView(
            model=self.packet_store
        )
        columns = [
            ("ID", 0),
            ("Source", 1),
            ("Destination", 2),
            ("Bytes", 3),
            ("Status", 4),
            ("Next Hop", 5),
            ("TTL", 6),
            ("Hops", 7),
            ("Delay ms", 8)
        ]
        for title, index in columns:
            renderer = (
                Gtk.CellRendererText()
            )
            column = (
                Gtk.TreeViewColumn(
                    title,
                    renderer,
                    text=index
                )
            )
            tree.append_column(
                column
            )
        scroll = Gtk.ScrolledWindow()
        scroll.add(
            tree
        )
        packet_box.pack_start(
            scroll,
            True,
            True,
            0
```

```
        )
        # ================================================
        # ROUTING PROCESS TAB
        # ================================================
        process_box = Gtk.Box(
            orientation=Gtk.Orientation.VERTICAL,
            spacing=8
        )
        notebook.append_page(
            process_box,
            Gtk.Label(
                label="Network Layer Process"
            )
        )
        process_text = Gtk.Label()
        process_text.set_markup(
            "<b>IPv4 Network Layer Processing</b>\n\n"
            "1. Packet generation\n"
            "      \n"
            "2. Source and destination IP assignment\n"
            "      \n"
            "3. Routing-table lookup\n"
            "      \n"
            "4. Longest Prefix Match\n"
            "      \n"
            "5. Next-hop selection\n"
            "      \n"
            "6. TTL decrement\n"
            "      \n"
            "7. Packet forwarding\n"
            "      \n"
            "8. Network delay model\n"
            "      \n"
            "9. Packet-loss model\n"
            "      \n"
            "10. Destination delivery\n"
            "      \n"
            "11. Performance measurement\n\n"
            "<b>Measured Parameters</b>\n\n"
            "    Packet Delivery Ratio\n"
            "    Packet Loss\n"
            "    Throughput\n"
            "    End-to-end delay\n"
            "    Hop count\n"
            "    TTL expiration\n"
            "    Routing failure"
```

```
        )
        process_text.set_xalign(
            0
        )
        process_text.set_yalign(
            0
        )
        process_box.pack_start(
            process_text,
            False,
            False,
            15
        )
        # ------------------------------------------------
        # STATUS
        # ------------------------------------------------
        self.status = Gtk.Label(
            label="Ready."
        )
        main_box.pack_start(
            self.status,
            False,
            False,
            0
        )
    # ========================================================
    # PARAMETERS
    # ========================================================
    def get_parameters(self):
        try:
            packet_count = int(
                self.packet_entry.get_text()
            )
            packet_size = int(
                self.size_entry.get_text()
            )
            ttl = int(
                self.ttl_entry.get_text()
            )
            loss = float(
                self.loss_entry.get_text()
            )
            min_delay = float(
                self.min_delay_entry.get_text()
            )
            max_delay = float(
```

```
                self.max_delay_entry.get_text()
            )
            if packet_count <= 0:
                raise ValueError(
                    "Packet count must be greater than 0."
                )
            if packet_size <= 0:
                raise ValueError(
                    "Packet size must be greater than 0."
                )
            if ttl <= 0:
                raise ValueError(
                    "TTL must be greater than 0."
                )
            if not 0 <= loss <= 1:
                raise ValueError(
                    "Loss probability must be between 0 and 1."
                )
            if min_delay < 0:
                raise ValueError(
                    "Minimum delay cannot be negative."
                )
            if max_delay < min_delay:
                raise ValueError(
                    "Maximum delay must be >= minimum delay."
                )
            return (
                packet_count,
                packet_size,
                ttl,
                loss,
                min_delay,
                max_delay
            )
        except ValueError as error:
            self.error_dialog(
                str(error)
            )
            return None
    # ======================================================
    # START SIMULATION
    # ======================================================
    def start_simulation(
        self,
        button
    ):
```

```
        parameters = (
            self.get_parameters()
        )
        if parameters is None:
            return
        (
            packet_count,
            packet_size,
            ttl,
            loss,
            min_delay,
            max_delay
        ) = parameters
        self.simulator = (
            NetworkLayerSimulator(
                packet_count=
                    packet_count,
                packet_size=
                    packet_size,
                ttl=
                    ttl,
                loss_probability=
                    loss,
                min_delay=
                    min_delay,
                max_delay=
                    max_delay
            )
        )
        self.packet_store.clear()
        self.figure.clear()
        self.canvas.draw()
        self.progress.set_fraction(
            0
        )
        self.start_button.set_sensitive(
            False
        )
        self.status.set_text(
            "Simulation running..."
        )
        self.worker_thread = threading.Thread(
            target=
                self.run_worker,
            daemon=True
        )
```

```
        self.worker_thread.start()
    # ========================================================
    # WORKER
    # ========================================================
    def run_worker(self):
        self.simulator.run(
            progress_callback=
                self.progress_callback,
            packet_callback=
                self.packet_callback
        )
        GLib.idle_add(
            self.simulation_finished
        )
    # ========================================================
    # PACKET CALLBACK
    # ========================================================
    def packet_callback(
        self,
        packet
    ):
        GLib.idle_add(
            self.update_live_view
        )
    # ========================================================
    # PROGRESS
    # ========================================================
    def progress_callback(
        self,
        current,
        total
    ):
        GLib.idle_add(
            self.update_progress,
            current,
            total
        )
    def update_progress(
        self,
        current,
        total
    ):
        self.progress.set_fraction(
            current / total
        )
        self.status.set_text(
```

```
            f"Processing packet "
            f"{current}/{total}"
        )
        self.update_live_view()
        return False
    # ========================================================
    # LIVE VIEW
    # ========================================================
    def update_live_view(self):
        dataframe = (
            self.simulator.dataframe()
        )
        if dataframe.empty:
            return False
        self.update_packet_table()
        self.update_metrics()
        self.update_live_graph()
        return False
    # ========================================================
    # LIVE GRAPH
    # ========================================================
    def update_live_graph(self):
        dataframe = (
            self.simulator.dataframe()
        )
        if dataframe.empty:
            return
        self.figure.clear()
        # ----------------------------------------------------
        # 1. DELAY
        # ----------------------------------------------------
        ax1 = self.figure.add_subplot(
            331
        )
        ax1.plot(
            dataframe["packet_id"],
            dataframe["delay_ms"],
            linewidth=1.5
        )
        ax1.set_title(
            "Delay / Packet"
        )
        ax1.set_xlabel(
            "Packet"
        )
        ax1.set_ylabel(
```

```
            "ms"
        )
        ax1.grid(
            True,
            alpha=0.3
        )
        # ------------------------------------------------------
        # 2. TTL
        # ------------------------------------------------------
        ax2 = self.figure.add_subplot(
            332
        )
        ax2.plot(
            dataframe["packet_id"],
            dataframe["ttl"],
            linewidth=1.5
        )
        ax2.set_title(
            "TTL"
        )
        ax2.set_xlabel(
            "Packet"
        )
        ax2.set_ylabel(
            "TTL"
        )
        ax2.grid(
            True,
            alpha=0.3
        )
        # ------------------------------------------------------
        # 3. HOPS
        # ------------------------------------------------------
        ax3 = self.figure.add_subplot(
            333
        )
        ax3.plot(
            dataframe["packet_id"],
            dataframe["hop_count"],
            linewidth=1.5
        )
        ax3.set_title(
            "Hop Count"
        )
        ax3.set_xlabel(
            "Packet"
```

```
        )
        ax3.set_ylabel(
            "Hops"
        )
        ax3.grid(
            True,
            alpha=0.3
        )
        # ----------------------------------------------------
        # 4. PDR
        # ----------------------------------------------------
        ax4 = self.figure.add_subplot(
            334
        )
        ax4.plot(
            dataframe["packet_id"],
            dataframe["pdr"],
            linewidth=2
        )
        ax4.set_title(
            "Packet Delivery Ratio"
        )
        ax4.set_xlabel(
            "Packet"
        )
        ax4.set_ylabel(
            "PDR (%)"
        )
        ax4.set_ylim(
            0,
            105
        )
        ax4.grid(
            True,
            alpha=0.3
        )
        # ----------------------------------------------------
        # 5. THROUGHPUT
        # ----------------------------------------------------
        ax5 = self.figure.add_subplot(
            335
        )
        throughput_mbps = (
            dataframe[
                "throughput_bps"
            ].to_numpy()
```

```
            / 1_000_000
        )
        ax5.plot(
            dataframe["packet_id"],
            throughput_mbps,
            linewidth=2
        )
        ax5.set_title(
            "Throughput"
        )
        ax5.set_xlabel(
            "Packet"
        )
        ax5.set_ylabel(
            "Mbps"
        )
        ax5.grid(
            True,
            alpha=0.3
        )
        # ----------------------------------------------------
        # 6. CUMULATIVE DELIVERED
        # ----------------------------------------------------
        ax6 = self.figure.add_subplot(
            336
        )
        ax6.plot(
            dataframe["packet_id"],
            dataframe["delivered"],
            linewidth=2
        )
        ax6.set_title(
            "Cumulative Delivered"
        )
        ax6.set_xlabel(
            "Packet"
        )
        ax6.set_ylabel(
            "Packets"
        )
        ax6.grid(
            True,
            alpha=0.3
        )
        # ----------------------------------------------------
        # 7. CUMULATIVE DROPPED
```

```
        # ---------------------------------------------------
        ax7 = self.figure.add_subplot(
            337
        )
        ax7.plot(
            dataframe["packet_id"],
            dataframe["dropped"],
            linewidth=2
        )
        ax7.set_title(
            "Cumulative Dropped"
        )
        ax7.set_xlabel(
            "Packet"
        )
        ax7.set_ylabel(
            "Packets"
        )
        ax7.grid(
            True,
            alpha=0.3
        )
        # ---------------------------------------------------
        # 8. STATUS DISTRIBUTION
        # ---------------------------------------------------
        ax8 = self.figure.add_subplot(
            338
        )
        status_counts = (
            dataframe[
                "status"
            ]
            .value_counts()
        )
        ax8.bar(
            status_counts.index,
            status_counts.values
        )
        ax8.set_title(
            "Packet Status"
        )
        ax8.set_ylabel(
            "Count"
        )
        ax8.tick_params(
            axis="x",
```

```
            rotation=40
        )
        ax8.grid(
            True,
            axis="y",
            alpha=0.3
        )
        # ------------------------------------------------
        # 9. PROCESS TIME
        # ------------------------------------------------
        ax9 = self.figure.add_subplot(
            339
        )
        ax9.plot(
            dataframe["packet_id"],
            dataframe[
                "elapsed_seconds"
            ],
            linewidth=2
        )
        ax9.set_title(
            "Simulation Time"
        )
        ax9.set_xlabel(
            "Packet"
        )
        ax9.set_ylabel(
            "Seconds"
        )
        ax9.grid(
            True,
            alpha=0.3
        )
        self.figure.suptitle(
            "Network Layer Simulation Process",
            fontsize=14,
            fontweight="bold"
        )
        self.figure.tight_layout(
            rect=[
                0,
                0,
                1,
                0.96
            ]
        )
```

```
        self.canvas.draw_idle()
    # ========================================================
    # PACKET TABLE
    # ========================================================
    def update_packet_table(self):
        self.packet_store.clear()
        for packet in (
            self.simulator.packets
        ):
            self.packet_store.append(
                [
                    packet.packet_id,
                    packet.source,
                    packet.destination,
                    packet.payload_size,
                    packet.status,
                    packet.next_hop or "-",
                    packet.ttl,
                    packet.hop_count,
                    round(
                        packet.delay_ms,
                        2
                    )
                ]
            )
    # ========================================================
    # METRICS
    # ========================================================
    def update_metrics(self):
        metrics = (
            self.simulator.get_metrics()
        )
        self.metric_labels[
            "generated"
        ].set_markup(
            f"<b>{metrics['generated']}</b>"
        )
        self.metric_labels[
            "forwarded"
        ].set_markup(
            f"<b>{metrics['forwarded']}</b>"
        )
        self.metric_labels[
            "delivered"
        ].set_markup(
            f"<b>{metrics['delivered']}</b>"
```

```
        )
        self.metric_labels[
            "dropped"
        ].set_markup(
            f"<b>{metrics['dropped']}</b>"
        )
        self.metric_labels[
            "pdr"
        ].set_markup(
            f"<b>{metrics['pdr']:.2f}%</b>"
        )
        self.metric_labels[
            "average_delay"
        ].set_markup(
            f"<b>{metrics['average_delay']:.2f} ms</b>"
        )
        self.metric_labels[
            "average_hops"
        ].set_markup(
            f"<b>{metrics['average_hops']:.2f}</b>"
        )
        throughput_mbps = (
            metrics["throughput"] /
            1_000_000
        )
        self.metric_labels[
            "throughput"
        ].set_markup(
            f"<b>{throughput_mbps:.3f} Mbps</b>"
        )
    # ========================================================
    # SIMULATION FINISHED
    # ========================================================
    def simulation_finished(self):
        self.start_button.set_sensitive(
            True
        )
        self.update_packet_table()
        self.update_metrics()
        self.update_live_graph()
        metrics = (
            self.simulator.get_metrics()
        )
        self.status.set_text(
            "Simulation completed: "
            f"{metrics['delivered']} delivered, "
```

```
            f"{metrics['dropped']} dropped, "
            f"PDR = "
            f"{metrics['pdr']:.2f}%"
        )
        return False
    # ========================================================
    # STOP
    # ========================================================
    def stop_simulation(
        self,
        button
    ):
        self.simulator.stop()
        self.status.set_text(
            "Simulation stopped."
        )
    # ========================================================
    # CSV EXPORT
    # ========================================================
    def export_csv(
        self,
        button
    ):
        dataframe = (
            self.simulator.dataframe()
        )
        if dataframe.empty:
            self.error_dialog(
                "No simulation data available."
            )
            return
        dialog = Gtk.FileChooserDialog(
            title=
                "Export Network Layer CSV",
            parent=
                self,
            action=
                Gtk.FileChooserAction.SAVE
        )
        dialog.add_buttons(
            Gtk.STOCK_CANCEL,
            Gtk.ResponseType.CANCEL,
            Gtk.STOCK_SAVE,
            Gtk.ResponseType.OK
        )
        dialog.set_current_name(
```

```
            "network_layer_results.csv"
        )
        response = dialog.run()
        if response == Gtk.ResponseType.OK:
            filename = (
                dialog.get_filename()
            )
            try:
                dataframe.to_csv(
                    filename,
                    index=False
                )
                self.status.set_text(
                    "CSV exported: " +
                    filename
                )
            except Exception as error:
                self.error_dialog(
                    str(error)
                )
        dialog.destroy()
    # ========================================================
    # ROUTING TABLE
    # ========================================================
    def show_routing_table(
        self,
        button
    ):
        window = Gtk.Window(
            title="IPv4 Routing Table"
        )
        window.set_default_size(
            700,
            400
        )
        store = Gtk.ListStore(
            str,
            str,
            str,
            int
        )
        for route in (
            self.simulator.routing_table.routes
        ):
            store.append(
                [
```

```
                    route["network"],
                    route["next_hop"],
                    route["interface"],
                    route["metric"]
                ]
            )
        tree = Gtk.TreeView(
            model=store
        )
        columns = [
            ("Network", 0),
            ("Next Hop", 1),
            ("Interface", 2),
            ("Metric", 3)
        ]
        for title, index in columns:
            renderer = (
                Gtk.CellRendererText()
            )
            column = (
                Gtk.TreeViewColumn(
                    title,
                    renderer,
                    text=index
                )
            )
            tree.append_column(
                column
            )
        scroll = Gtk.ScrolledWindow()
        scroll.add(
            tree
        )
        window.add(
            scroll
        )
        window.show_all()
    # ========================================================
    # ERROR
    # ========================================================
    def error_dialog(
        self,
        message
    ):
        dialog = Gtk.MessageDialog(
            parent=self,
```

```
            flags=
                Gtk.DialogFlags.MODAL,
            message_type=
                Gtk.MessageType.ERROR,
            buttons=
                Gtk.ButtonsType.OK,
            text=
                "Simulation Error"
        )
        dialog.format_secondary_text(
            message
        )
        dialog.run()
        dialog.destroy()
# ============================================================
# MAIN
# ============================================================
def main():
    application = (
        NetworkLayerApplication()
    )
    application.connect(
        "destroy",
        Gtk.main_quit
    )
    application.show_all()
    Gtk.main()
if __name__ == "__main__":
    main()
```

Aplikasi ini adalah simulator *Network Layer* yang menampilkan hasil simulasi dalam antar muka grafis pustaka GTK dan grafik dengan pustaka Matplotlib. Arsitektur aplikasi simulator *Network Layer* dapat kita lihat pada gambar 2.10: Parameter yang digunakan pada simulator *Network Layer* adalah *Packet Delivery Ratio* (PDR), *throughput, delay, hop-count, Time To Live* (TTL) *expiration, packet loss* dan *routing failure*. Sedangkan mekanisme plot pada simulator *Network Layer* meliputi:

1. Packet processing over time
2. Cumulative delivered packets
3. Cumulative dropped packets
4. Packet Delivery Ratio (PDR)
5. Delay per packet
6. TTL per packet
7. Hop count per packet
8. Throughput over simulation time

9. Packet status distribution
10. Live simulation progression

Pembaca dapat melihat penjelasan kode sumber pada keterangan simulator Network Layer.

2.5.1 Keterangan Hasil Plot Simulator Network Layer

Hasil *plotting* simulator Network Layer memberikan keterangan yang meliputi:

Plot	**Fungsi atau Tujuan**
Delay / Packet	Menunjukkan *latency* jaringan per-packet
Time to Live (TTL)	Menunjukkan reduksi TTL melalui mekanisme *forwarding*
Hop Count	Menunjukkan panjang lintasan routing
PDR	Menunjukkan hasil penjumlahan Packet Delivery Ratio
Throughput	Menunjukkan hasil throughput jaringan yang tercapai
Cumulative Delivered	Menunjukkan akumulasi paket yang berhasil dikirimkan
Cumulative Dropped	Menunjukkan akumulasi paket yang hilang sebelum mencapai tujuannya
Packet Status	Menunjukkan keterangan DELIVERED, LOST, TTL EXPIRED, dan lainnya.
Simulation Time	Menunjukkan waktu yang dibutuhkan untuk melakukan simulasi.

2.5.2 Tampilan Simulator Network Layer

Tampilan dari simulator Network Layer dapat pembaca lihat pada Gambar 2.11:
Tampilan dari Packet Log dari simulator Network Layer dapat pembaca lihat pada Gambar 2.12:
Tampilan dari Routing Table dari simulator Network Layer dapat pembaca lihat pada Gambar 2.13:

2.6 Komponen *Transport Layer*

Komponen *Transport Layer* merupakan aplikasi yang dikembangkan dengan bahasa pemrograman Python untuk mendeskripsikan simulasi pada *Transport Layer* [13] 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_transport_layer_research_simulation.py:

Listing 2.6: Transport Layer Simulation

```
#!/usr/bin/env python3
"""
================================================================
TRANSPORT LAYER SIMULATION
================================================================
Single-file Python 3 application using:

    GTK3 / PyGObject
    Matplotlib
    NumPy
```

```
    Pandas
Protocols:
    TCP
    UDP
Transport-layer simulation features:
    TCP:
        - Three-way handshake
        - SYN
        - SYN-ACK
        - ACK
        - Sequence numbers
        - Acknowledgement numbers
        - RTT
        - RTO
        - Retransmission
        - Congestion Window
        - Slow Start
        - Congestion Avoidance
    UDP:
        - Connectionless transmission
        - Datagram transmission
        - Packet loss
General:
    - Segmentation
    - Packet transmission
    - Packet loss
    - Network delay
    - PDR
    - Loss rate
    - Throughput
    - Goodput
    - Average RTT
    - Average delay
    - Retransmissions
    - Live simulation graph
    - Scrollable graph area
    - Packet log
    - CSV export
GUI:
    - Resizable
    - Maximizable
    - Vertical graph scrolling
    - No horizontal graph scrolling
Run:
    python3 final_transport_layer_research_simulation.py
======================================================================
```

```
"""
import gi
gi.require_version("Gtk", "3.0")
from gi.repository import Gtk, GLib
import matplotlib
matplotlib.use("GTK3Agg")
from matplotlib.figure import Figure
from matplotlib.backends.backend_gtk3agg import FigureCanvasGTK3Agg
import numpy as np
import pandas as pd
import random
import time
import threading
# ============================================================================
# TRANSPORT SEGMENT
# ============================================================================
class TransportSegment:
    def __init__(
        self,
        segment_id,
        protocol,
        source_port,
        destination_port,
        sequence_number,
        payload_size
    ):
        self.segment_id = segment_id
        self.protocol = protocol
        self.source_port = source_port
        self.destination_port = destination_port
        self.sequence_number = sequence_number
        self.ack_number = 0
        self.payload_size = payload_size
        self.status = "CREATED"
        self.delay_ms = 0.0
        self.rtt_ms = 0.0
        self.retransmissions = 0
        self.send_time = 0.0
        self.ack_time = 0.0
# ============================================================================
# TCP CONNECTION
# ============================================================================
class TCPConnection:
    def __init__(self):
        self.state = "CLOSED"
        self.client_sequence = 1000
```

```
        self.server_sequence = 5000
        self.ack_number = 0
        self.cwnd = 1.0
        self.ssthresh = 16.0
        self.rtt_ms = 50.0
        self.rto_ms = 100.0
        self.total_ack = 0
        self.total_retransmission = 0
        self.handshake_time_ms = 0.0
    # ------------------------------------------------------------
    # TCP THREE-WAY HANDSHAKE
    # ------------------------------------------------------------
    def three_way_handshake(
        self,
        min_delay,
        max_delay
    ):
        handshake_start = time.time()
        # ------------------------------------------------------------
        # SYN
        # ------------------------------------------------------------
        self.state = "SYN_SENT"
        syn_delay = random.uniform(
            min_delay,
            max_delay
        )
        # ------------------------------------------------------------
        # SYN-ACK
        # ------------------------------------------------------------
        self.state = "SYN_RECEIVED"
        syn_ack_delay = random.uniform(
            min_delay,
            max_delay
        )
        # ------------------------------------------------------------
        # ACK
        # ------------------------------------------------------------
        self.state = "ESTABLISHED"
        ack_delay = random.uniform(
            min_delay,
            max_delay
        )
        measured_time = (
            time.time() -
            handshake_start
        ) * 1000.0
```

```
        self.handshake_time_ms = (
            measured_time
            +
            syn_delay
            +
            syn_ack_delay
            +
            ack_delay
        )
        return self.handshake_time_ms
# ======================================================================
# TRANSPORT LAYER SIMULATOR
# ======================================================================
class TransportLayerSimulator:
    def __init__(
        self,
        protocol="TCP",
        segment_count=100,
        segment_size=1024,
        loss_probability=0.05,
        min_delay=5.0,
        max_delay=50.0
    ):
        self.protocol = protocol
        self.segment_count = segment_count
        self.segment_size = segment_size
        self.loss_probability = loss_probability
        self.min_delay = min_delay
        self.max_delay = max_delay
        self.source_port = 50000
        self.destination_port = 443
        self.tcp = TCPConnection()
        self.segments = []
        self.statistics = []
        self.running = False
        self.start_time = None
        self.end_time = None
        self.generated = 0
        self.transmitted = 0
        self.delivered = 0
        self.lost = 0
        self.acknowledged = 0
        self.retransmissions = 0
        self.total_delay = 0.0
        self.total_rtt = 0.0
        self.total_payload_bytes = 0
```

```
        self.total_goodput_bytes = 0
    # ------------------------------------------------------------------
    # RESET
    # ------------------------------------------------------------------
    def reset(self):
        self.segments = []
        self.statistics = []
        self.running = False
        self.start_time = None
        self.end_time = None
        self.generated = 0
        self.transmitted = 0
        self.delivered = 0
        self.lost = 0
        self.acknowledged = 0
        self.retransmissions = 0
        self.total_delay = 0.0
        self.total_rtt = 0.0
        self.total_payload_bytes = 0
        self.total_goodput_bytes = 0
        self.tcp = TCPConnection()
    # ------------------------------------------------------------------
    # TCP HANDSHAKE
    # ------------------------------------------------------------------
    def perform_tcp_handshake(self):
        if self.protocol != "TCP":
            return 0.0
        return self.tcp.three_way_handshake(
            self.min_delay,
            self.max_delay
        )
    # ------------------------------------------------------------------
    # CREATE SEGMENT
    # ------------------------------------------------------------------
    def create_segment(
        self,
        segment_id
    ):
        if self.protocol == "TCP":
            sequence_number = (
                self.tcp.client_sequence
                +
                (
                    segment_id - 1
                )
                *
```

```
                self.segment_size
            )
        else:
            sequence_number = (
                segment_id
                *
                self.segment_size
            )
        return TransportSegment(
            segment_id=segment_id,
            protocol=self.protocol,
            source_port=self.source_port,
            destination_port=self.destination_port,
            sequence_number=sequence_number,
            payload_size=self.segment_size
        )
    # ------------------------------------------------------------
    # TRANSMIT SEGMENT
    # ------------------------------------------------------------
    def transmit_segment(
        self,
        segment
    ):
        segment.status = "TRANSMITTING"
        self.transmitted += 1
        segment.send_time = time.time()
        # ------------------------------------------------------------
        # TRANSMISSION DELAY
        # ------------------------------------------------------------
        segment.delay_ms = random.uniform(
            self.min_delay,
            self.max_delay
        )
        # ------------------------------------------------------------
        # PACKET LOSS
        # ------------------------------------------------------------
        lost = (
            random.random()
            <
            self.loss_probability
        )
        if lost:
            segment.status = "LOST"
            self.lost += 1
            # ========================================================
            # TCP RETRANSMISSION
```

```
            # ============================================================
            if self.protocol == "TCP":
                segment.retransmissions += 1
                self.retransmissions += 1
                self.tcp.total_retransmission += 1
                retry_delay = min(
                    self.tcp.rto_ms,
                    1000.0
                )
                segment.delay_ms += retry_delay
                recovered = (
                    random.random()
                    >
                    self.loss_probability
                )
                if recovered:
                    segment.status = "RETRANSMITTED"
                    segment.delay_ms += random.uniform(
                        self.min_delay,
                        self.max_delay
                    )
                else:
                    return segment
            # ============================================================
            # UDP DOES NOT RETRANSMIT
            # ============================================================
            else:
                return segment
        # ============================================================
        # TCP ACK
        # ============================================================
        if self.protocol == "TCP":
            segment.ack_number = (
                segment.sequence_number
                +
                segment.payload_size
            )
            ack_delay = random.uniform(
                self.min_delay,
                self.max_delay
            )
            segment.rtt_ms = (
                segment.delay_ms
                +
                ack_delay
            )
```

```
            self.total_rtt += segment.rtt_ms
            self.acknowledged += 1
            self.tcp.total_ack += 1
            # ------------------------------------------------------
            # TCP CONGESTION CONTROL
            # ------------------------------------------------------
            if self.tcp.cwnd < self.tcp.ssthresh:
                # Slow Start
                self.tcp.cwnd += 1.0
            else:
                # Congestion Avoidance
                self.tcp.cwnd += (
                    1.0
                    /
                    max(
                        self.tcp.cwnd,
                        1.0
                    )
                )
            # ------------------------------------------------------
            # RTT ESTIMATION
            # ------------------------------------------------------
            self.tcp.rtt_ms = (
                0.875
                *
                self.tcp.rtt_ms
                +
                0.125
                *
                segment.rtt_ms
            )
            # ------------------------------------------------------
            # RTO ESTIMATION
            # ------------------------------------------------------
            self.tcp.rto_ms = max(
                100.0,
                2.0
                *
                self.tcp.rtt_ms
            )
            segment.status = "ACKED"
        # ============================================================
        # UDP
        # ============================================================
        else:
            segment.status = "DELIVERED"
```

```
        # ============================================================
        # SUCCESSFUL DELIVERY
        # ============================================================
        self.delivered += 1
        self.total_delay += segment.delay_ms
        self.total_payload_bytes += segment.payload_size
        self.total_goodput_bytes += segment.payload_size
        return segment
    # ------------------------------------------------------------------
    # RUN SIMULATION
    # ------------------------------------------------------------------
    def run(
        self,
        progress_callback=None,
        segment_callback=None
    ):
        self.reset()
        self.running = True
        self.start_time = time.time()
        handshake_ms = 0.0
        if self.protocol == "TCP":
            handshake_ms = (
                self.perform_tcp_handshake()
            )
        # ------------------------------------------------------------
        # SEGMENT LOOP
        # ------------------------------------------------------------
        for segment_id in range(
            1,
            self.segment_count + 1
        ):
            if not self.running:
                break
            segment = self.create_segment(
                segment_id
            )
            self.generated += 1
            segment = self.transmit_segment(
                segment
            )
            self.segments.append(
                segment
            )
            elapsed = (
                time.time()
                -
```

```
                self.start_time
            )
            # ------------------------------------------------------------
            # PDR
            # ------------------------------------------------------------
            pdr = 0.0
            if self.generated > 0:
                pdr = (
                    self.delivered
                    /
                    self.generated
                ) * 100.0
            # ------------------------------------------------------------
            # LOSS RATE
            # ------------------------------------------------------------
            loss_rate = 0.0
            if self.generated > 0:
                loss_rate = (
                    self.lost
                    /
                    self.generated
                ) * 100.0
            # ------------------------------------------------------------
            # THROUGHPUT
            # ------------------------------------------------------------
            throughput_bps = 0.0
            if elapsed > 0:
                throughput_bps = (
                    self.total_goodput_bytes
                    *
                    8
                ) / elapsed
            # ------------------------------------------------------------
            # AVERAGE RTT
            # ------------------------------------------------------------
            average_rtt = 0.0
            if self.acknowledged > 0:
                average_rtt = (
                    self.total_rtt
                    /
                    self.acknowledged
                )
            # ------------------------------------------------------------
            # AVERAGE DELAY
            # ------------------------------------------------------------
            average_delay = 0.0
```

```
            if self.delivered > 0:
                average_delay = (
                    self.total_delay
                    /
                    self.delivered
                )
            # ------------------------------------------------------------
            # RECORD
            # ------------------------------------------------------------
            self.statistics.append(
                {
                    "segment_id":
                        segment.segment_id,
                    "protocol":
                        segment.protocol,
                    "source_port":
                        segment.source_port,
                    "destination_port":
                        segment.destination_port,
                    "sequence_number":
                        segment.sequence_number,
                    "ack_number":
                        segment.ack_number,
                    "payload_bytes":
                        segment.payload_size,
                    "status":
                        segment.status,
                    "delay_ms":
                        segment.delay_ms,
                    "rtt_ms":
                        segment.rtt_ms,
                    "retransmissions":
                        segment.retransmissions,
                    "cwnd":
                        (
                            self.tcp.cwnd
                            if
                            self.protocol == "TCP"
                            else
                            0.0
                        ),
                    "rto_ms":
                        (
                            self.tcp.rto_ms
                            if
                            self.protocol == "TCP"
```

```
                            else
                            0.0
                        ),
                    "generated":
                        self.generated,
                    "transmitted":
                        self.transmitted,
                    "delivered":
                        self.delivered,
                    "lost":
                        self.lost,
                    "acknowledged":
                        self.acknowledged,
                    "total_retransmissions":
                        self.retransmissions,
                    "pdr":
                        pdr,
                    "loss_rate":
                        loss_rate,
                    "throughput_bps":
                        throughput_bps,
                    "average_rtt_ms":
                        average_rtt,
                    "average_delay_ms":
                        average_delay,
                    "elapsed_seconds":
                        elapsed,
                    "handshake_ms":
                        handshake_ms
                }
            )
            # ------------------------------------------------------------
            # CALLBACKS
            # ------------------------------------------------------------
            if progress_callback:
                progress_callback(
                    self.generated,
                    self.segment_count
                )
            if segment_callback:
                segment_callback(
                    segment
                )
            # ------------------------------------------------------------
            # SLOW DOWN SIMULATION FOR LIVE GRAPH
            # ------------------------------------------------------------
```

```
            time.sleep(
                0.03
            )
        self.end_time = time.time()
        self.running = False
    # ------------------------------------------------------------------
    # STOP
    # ------------------------------------------------------------------
    def stop(self):
        self.running = False
    # ------------------------------------------------------------------
    # DATAFRAME
    # ------------------------------------------------------------------
    def dataframe(self):
        return pd.DataFrame(
            self.statistics
        )
    # ------------------------------------------------------------------
    # METRICS
    # ------------------------------------------------------------------
    def get_metrics(self):
        elapsed = 0.0
        if (
            self.start_time is not None
            and
            self.end_time is not None
        ):
            elapsed = (
                self.end_time
                -
                self.start_time
            )
        pdr = 0.0
        if self.generated > 0:
            pdr = (
                self.delivered
                /
                self.generated
            ) * 100.0
        loss_rate = 0.0
        if self.generated > 0:
            loss_rate = (
                self.lost
                /
                self.generated
            ) * 100.0
```

```
        average_delay = 0.0
        if self.delivered > 0:
            average_delay = (
                self.total_delay
                /
                self.delivered
            )
        average_rtt = 0.0
        if self.acknowledged > 0:
            average_rtt = (
                self.total_rtt
                /
                self.acknowledged
            )
        throughput_bps = 0.0
        if elapsed > 0:
            throughput_bps = (
                self.total_goodput_bytes
                *
                8
            ) / elapsed
        return {
            "generated":
                self.generated,
            "transmitted":
                self.transmitted,
            "delivered":
                self.delivered,
            "lost":
                self.lost,
            "acknowledged":
                self.acknowledged,
            "retransmissions":
                self.retransmissions,
            "pdr":
                pdr,
            "loss_rate":
                loss_rate,
            "average_delay":
                average_delay,
            "average_rtt":
                average_rtt,
            "throughput":
                throughput_bps,
            "elapsed":
                elapsed,
```

```
            "handshake":
                self.tcp.handshake_time_ms,
            "cwnd":
                self.tcp.cwnd,
            "rto":
                self.tcp.rto_ms
        }
# ================================================================
# GTK APPLICATION
# ================================================================
class TransportLayerGTK(Gtk.Window):
    def __init__(self):
        super().__init__(
            title="Transport Layer Simulation"
        )
        # ============================================================
        # WINDOW CONFIGURATION
        # ============================================================
        self.set_default_size(
            1280,
            800
        )
        self.set_size_request(
            900,
            600
        )
        self.set_border_width(
            8
        )
        self.set_resizable(
            True
        )
        self.set_position(
            Gtk.WindowPosition.CENTER
        )
        self.simulator = (
            TransportLayerSimulator()
        )
        self.worker_thread = None
        self.build_ui()
    # ================================================================
    # BUILD UI
    # ================================================================
    def build_ui(self):
        main_box = Gtk.Box(
            orientation=Gtk.Orientation.VERTICAL,
```

```
            spacing=6
        )
        self.add(
            main_box
        )
        # ============================================================
        # TITLE
        # ============================================================
        title = Gtk.Label()
        title.set_markup(
            "<span size='xx-large' "
            "weight='bold'>"
            "Transport Layer Simulation"
            "</span>"
        )
        main_box.pack_start(
            title,
            False,
            False,
            3
        )
        subtitle = Gtk.Label(
            label=(
                "TCP / UDP     segmentation     "
                "transmission     ACK     "
                "retransmission     throughput"
            )
        )
        main_box.pack_start(
            subtitle,
            False,
            False,
            2
        )
        # ============================================================
        # PARAMETERS
        # ============================================================
        parameter_frame = Gtk.Frame(
            label="Simulation Parameters"
        )
        parameter_grid = Gtk.Grid()
        parameter_grid.set_border_width(
            6
        )
        parameter_grid.set_row_spacing(
            5
```

```
        )
        parameter_grid.set_column_spacing(
            7
        )
        parameter_frame.add(
            parameter_grid
        )
        main_box.pack_start(
            parameter_frame,
            False,
            False,
            0
        )
        # ------------------------------------------------------------
        # Protocol
        # ------------------------------------------------------------
        parameter_grid.attach(
            Gtk.Label(
                label="Protocol:"
            ),
            0,
            0,
            1,
            1
        )
        self.protocol_combo = (
            Gtk.ComboBoxText()
        )
        self.protocol_combo.append_text(
            "TCP"
        )
        self.protocol_combo.append_text(
            "UDP"
        )
        self.protocol_combo.set_active(
            0
        )
        parameter_grid.attach(
            self.protocol_combo,
            1,
            0,
            1,
            1
        )
        # ------------------------------------------------------------
        # Segment Count
```

```
        # ------------------------------------------------------------
        parameter_grid.attach(
            Gtk.Label(
                label="Segments:"
            ),
            2,
            0,
            1,
            1
        )
        self.segment_entry = (
            Gtk.Entry()
        )
        self.segment_entry.set_text(
            "100"
        )
        parameter_grid.attach(
            self.segment_entry,
            3,
            0,
            1,
            1
        )
        # ------------------------------------------------------------
        # Payload
        # ------------------------------------------------------------
        parameter_grid.attach(
            Gtk.Label(
                label="Payload Bytes:"
            ),
            4,
            0,
            1,
            1
        )
        self.size_entry = (
            Gtk.Entry()
        )
        self.size_entry.set_text(
            "1024"
        )
        parameter_grid.attach(
            self.size_entry,
            5,
            0,
            1,
```

```
            1
        )
        # ------------------------------------------------------------
        # Loss
        # ------------------------------------------------------------
        parameter_grid.attach(
            Gtk.Label(
                label="Loss Probability:"
            ),
            0,
            1,
            1,
            1
        )
        self.loss_entry = (
            Gtk.Entry()
        )
        self.loss_entry.set_text(
            "0.05"
        )
        parameter_grid.attach(
            self.loss_entry,
            1,
            1,
            1,
            1
        )
        # ------------------------------------------------------------
        # Minimum Delay
        # ------------------------------------------------------------
        parameter_grid.attach(
            Gtk.Label(
                label="Min Delay (ms):"
            ),
            2,
            1,
            1,
            1
        )
        self.min_delay_entry = (
            Gtk.Entry()
        )
        self.min_delay_entry.set_text(
            "5"
        )
        parameter_grid.attach(
```

```
            self.min_delay_entry,
            3,
            1,
            1,
            1
        )
        # ------------------------------------------------------------
        # Maximum Delay
        # ------------------------------------------------------------
        parameter_grid.attach(
            Gtk.Label(
                label="Max Delay (ms):"
            ),
            4,
            1,
            1,
            1
        )
        self.max_delay_entry = (
            Gtk.Entry()
        )
        self.max_delay_entry.set_text(
            "50"
        )
        parameter_grid.attach(
            self.max_delay_entry,
            5,
            1,
            1,
            1
        )
        # ============================================================
        # BUTTONS
        # ============================================================
        button_box = Gtk.Box(
            orientation=Gtk.Orientation.HORIZONTAL,
            spacing=7
        )
        self.start_button = Gtk.Button(
            label="Start Simulation"
        )
        self.start_button.connect(
            "clicked",
            self.start_simulation
        )
        button_box.pack_start(
```

```
            self.start_button,
            False,
            False,
            0
        )
        self.stop_button = Gtk.Button(
            label="Stop"
        )
        self.stop_button.connect(
            "clicked",
            self.stop_simulation
        )
        button_box.pack_start(
            self.stop_button,
            False,
            False,
            0
        )
        export_button = Gtk.Button(
            label="Export CSV"
        )
        export_button.connect(
            "clicked",
            self.export_csv
        )
        button_box.pack_start(
            export_button,
            False,
            False,
            0
        )
        parameter_grid.attach(
            button_box,
            0,
            2,
            6,
            1
        )
        # ============================================================
        # PROGRESS
        # ============================================================
        self.progress = (
            Gtk.ProgressBar()
        )
        self.progress.set_show_text(
            True
```

```
        )
        main_box.pack_start(
            self.progress,
            False,
            False,
            0
        )
        # ============================================================
        # METRICS
        # ============================================================
        metric_frame = Gtk.Frame(
            label="Transport Layer Metrics"
        )
        metric_grid = Gtk.Grid()
        metric_grid.set_border_width(
            5
        )
        metric_grid.set_column_spacing(
            10
        )
        metric_grid.set_row_spacing(
            4
        )
        metric_frame.add(
            metric_grid
        )
        main_box.pack_start(
            metric_frame,
            False,
            False,
            0
        )
        self.metric_labels = {}
        metrics = [
            ("Generated", "generated"),
            ("Delivered", "delivered"),
            ("Lost", "lost"),
            ("ACK", "acknowledged"),
            ("Retrans.", "retransmissions"),
            ("PDR", "pdr"),
            ("Loss", "loss_rate"),
            ("RTT", "average_rtt"),
            ("Delay", "average_delay"),
            ("Throughput", "throughput")
        ]
        for index, (
```

```
            caption,
            key
        ) in enumerate(metrics):
            metric_box = Gtk.Box(
                orientation=
                    Gtk.Orientation.VERTICAL,
                spacing=1
            )
            caption_label = Gtk.Label(
                label=caption
            )
            value_label = Gtk.Label(
                label="0"
            )
            value_label.set_markup(
                "<b>0</b>"
            )
            metric_box.pack_start(
                caption_label,
                False,
                False,
                0
            )
            metric_box.pack_start(
                value_label,
                False,
                False,
                0
            )
            metric_grid.attach(
                metric_box,
                index,
                0,
                1,
                1
            )
            self.metric_labels[
                key
            ] = value_label
        # ============================================================
        # NOTEBOOK
        # ============================================================
        notebook = Gtk.Notebook()
        notebook.set_vexpand(
            True
        )
```

```
        notebook.set_hexpand(
            True
        )
        main_box.pack_start(
            notebook,
            True,
            True,
            0
        )
        # ============================================================
        # LIVE SIMULATION GRAPH
        # ============================================================
        graph_box = Gtk.Box(
            orientation=
                Gtk.Orientation.VERTICAL,
            spacing=4
        )
        notebook.append_page(
            graph_box,
            Gtk.Label(
                label="Live Simulation Graph"
            )
        )
        # ------------------------------------------------------------
        # GRAPH SCROLL WINDOW
        # ------------------------------------------------------------
        self.graph_scroll = (
            Gtk.ScrolledWindow()
        )
        # Vertical scrolling only.
        self.graph_scroll.set_policy(
            Gtk.PolicyType.NEVER,
            Gtk.PolicyType.AUTOMATIC
        )
        self.graph_scroll.set_shadow_type(
            Gtk.ShadowType.IN
        )
        self.graph_scroll.set_vexpand(
            True
        )
        self.graph_scroll.set_hexpand(
            True
        )
        graph_box.pack_start(
            self.graph_scroll,
            True,
```

```
            True,
            0
        )
        # ------------------------------------------------------------
        # GRAPH CONTAINER
        # ------------------------------------------------------------
        self.graph_container = Gtk.Box(
            orientation=
                Gtk.Orientation.VERTICAL
        )
        self.graph_scroll.add(
            self.graph_container
        )
        # ------------------------------------------------------------
        # MATPLOTLIB FIGURE
        # ------------------------------------------------------------
        self.figure = Figure(
            figsize=(13, 16),
            dpi=100
        )
        self.canvas = (
            FigureCanvasGTK3Agg(
                self.figure
            )
        )
        # Large vertical canvas.
        # Gtk.ScrolledWindow handles scrolling.
        self.canvas.set_size_request(
            1100,
            1600
        )
        self.graph_container.pack_start(
            self.canvas,
            False,
            False,
            0
        )
        # ============================================================
        # SEGMENT LOG
        # ============================================================
        packet_box = Gtk.Box(
            orientation=
                Gtk.Orientation.VERTICAL
        )
        notebook.append_page(
            packet_box,
```

```
            Gtk.Label(
                label="Segment Log"
            )
        )
        self.packet_store = Gtk.ListStore(
            int,       # ID
            str,       # Protocol
            int,       # Source port
            int,       # Destination port
            int,       # Sequence
            str,       # Status
            float,     # Delay
            float,     # RTT
            int        # Retransmissions
        )
        tree = Gtk.TreeView(
            model=self.packet_store
        )
        columns = [
            ("ID", 0),
            ("Protocol", 1),
            ("Src Port", 2),
            ("Dst Port", 3),
            ("Sequence", 4),
            ("Status", 5),
            ("Delay ms", 6),
            ("RTT ms", 7),
            ("Retrans.", 8)
        ]
        for title, index in columns:
            renderer = (
                Gtk.CellRendererText()
            )
            column = (
                Gtk.TreeViewColumn(
                    title,
                    renderer,
                    text=index
                )
            )
            tree.append_column(
                column
            )
        packet_scroll = (
            Gtk.ScrolledWindow()
        )
```

```
        packet_scroll.set_policy(
            Gtk.PolicyType.AUTOMATIC,
            Gtk.PolicyType.AUTOMATIC
        )
        packet_scroll.add(
            tree
        )
        packet_box.pack_start(
            packet_scroll,
            True,
            True,
            0
        )
        # ============================================================
        # TRANSPORT PROCESS
        # ============================================================
        process_box = Gtk.Box(
            orientation=
                Gtk.Orientation.VERTICAL,
            spacing=10
        )
        notebook.append_page(
            process_box,
            Gtk.Label(
                label="Transport Process"
            )
        )
        process_scroll = (
            Gtk.ScrolledWindow()
        )
        process_scroll.set_policy(
            Gtk.PolicyType.NEVER,
            Gtk.PolicyType.AUTOMATIC
        )
        process_box.pack_start(
            process_scroll,
            True,
            True,
            0
        )
        process_label = Gtk.Label()
        process_label.set_markup(
            "<b>TCP TRANSPORT PROCESS</b>\n\n"
            "Application Data\n"
            "           \n"
            "Segmentation\n"
```

```
            "           \n"
            "TCP Header Creation\n"
            "           \n"
            "Three-Way Handshake\n"
            "           \n"
            "SYN\n"
            "           \n"
            "SYN-ACK\n"
            "           \n"
            "ACK\n"
            "           \n"
            "ESTABLISHED\n"
            "           \n"
            "Sequence Number Assignment\n"
            "           \n"
            "Segment Transmission\n"
            "           \n"
            "ACK Reception\n"
            "           \n"
            "RTT Measurement\n"
            "           \n"
            "Congestion Window Update\n"
            "           \n"
            "Slow Start / Congestion Avoidance\n"
            "           \n"
            "Packet Loss Detection\n"
            "           \n"
            "Retransmission\n"
            "           \n"
            "Reliable Delivery\n\n"
            "<b>UDP TRANSPORT PROCESS</b>\n\n"
            "Application Data\n"
            "           \n"
            "Datagram Creation\n"
            "           \n"
            "UDP Header Creation\n"
            "           \n"
            "Source Port / Destination Port\n"
            "           \n"
            "Datagram Transmission\n"
            "           \n"
            "Delivery or Loss\n\n"
            "<b>SIMULATION METRICS</b>\n\n"
            "    Packet Delivery Ratio\n"
            "    Packet Loss Rate\n"
            "    Throughput\n"
```

```
            "    Goodput\n"
            "    RTT\n"
            "    Delay\n"
            "    TCP Retransmissions\n"
            "    TCP Congestion Window\n"
            "    TCP RTO\n"
            "    ACK Progression"
        )
        process_label.set_xalign(
            0
        )
        process_label.set_yalign(
            0
        )
        process_label.set_margin_start(
            20
        )
        process_label.set_margin_top(
            20
        )
        process_scroll.add(
            process_label
        )
        # ============================================================
        # STATUS
        # ============================================================
        self.status = Gtk.Label(
            label="Ready."
        )
        main_box.pack_start(
            self.status,
            False,
            False,
            0
        )
    # ================================================================
    # READ PARAMETERS
    # ================================================================
    def get_parameters(self):
        try:
            protocol = (
                self.protocol_combo
                .get_active_text()
            )
            segment_count = int(
                self.segment_entry
```

```
                .get_text()
            )
            segment_size = int(
                self.size_entry
                .get_text()
            )
            loss_probability = float(
                self.loss_entry
                .get_text()
            )
            min_delay = float(
                self.min_delay_entry
                .get_text()
            )
            max_delay = float(
                self.max_delay_entry
                .get_text()
            )
            # ------------------------------------------------------------
            # VALIDATION
            # ------------------------------------------------------------
            if protocol not in (
                "TCP",
                "UDP"
            ):
                raise ValueError(
                    "Protocol must be TCP or UDP."
                )
            if segment_count <= 0:
                raise ValueError(
                    "Segment count must be greater than zero."
                )
            if segment_size <= 0:
                raise ValueError(
                    "Payload size must be greater than zero."
                )
            if not (
                0.0
                <=
                loss_probability
                <=
                1.0
            ):
                raise ValueError(
                    "Loss probability must be between 0 and 1."
                )
```

```
            if min_delay < 0:
                raise ValueError(
                    "Minimum delay cannot be negative."
                )
            if max_delay < min_delay:
                raise ValueError(
                    "Maximum delay must be greater than "
                    "or equal to minimum delay."
                )
            return (
                protocol,
                segment_count,
                segment_size,
                loss_probability,
                min_delay,
                max_delay
            )
        except ValueError as error:
            self.error_dialog(
                str(error)
            )
            return None
    # ================================================================
    # START SIMULATION
    # ================================================================
    def start_simulation(
        self,
        button
    ):
        parameters = (
            self.get_parameters()
        )
        if parameters is None:
            return
        (
            protocol,
            segment_count,
            segment_size,
            loss_probability,
            min_delay,
            max_delay
        ) = parameters
        self.simulator = (
            TransportLayerSimulator(
                protocol=
                    protocol,
```

```
                segment_count=
                    segment_count,
                segment_size=
                    segment_size,
                loss_probability=
                    loss_probability,
                min_delay=
                    min_delay,
                max_delay=
                    max_delay
            )
        )
        # ------------------------------------------------------------
        # RESET GUI
        # ------------------------------------------------------------
        self.packet_store.clear()
        self.figure.clear()
        self.canvas.draw()
        self.progress.set_fraction(
            0.0
        )
        self.progress.set_text(
            "0%"
        )
        self.start_button.set_sensitive(
            False
        )
        self.status.set_text(
            f"{protocol} simulation running..."
        )
        # ------------------------------------------------------------
        # START WORKER THREAD
        # ------------------------------------------------------------
        self.worker_thread = (
            threading.Thread(
                target=
                    self.run_worker,
                daemon=True
            )
        )
        self.worker_thread.start()
    # ================================================================
    # WORKER
    # ================================================================
    def run_worker(self):
        self.simulator.run(
```

```
            progress_callback=
                self.progress_callback,
            segment_callback=
                self.segment_callback
        )
        GLib.idle_add(
            self.simulation_finished
        )
    # =================================================================
    # SEGMENT CALLBACK
    # =================================================================
    def segment_callback(
        self,
        segment
    ):
        GLib.idle_add(
            self.update_live_view
        )
    # =================================================================
    # PROGRESS CALLBACK
    # =================================================================
    def progress_callback(
        self,
        current,
        total
    ):
        GLib.idle_add(
            self.update_progress,
            current,
            total
        )
    # =================================================================
    # UPDATE PROGRESS
    # =================================================================
    def update_progress(
        self,
        current,
        total
    ):
        fraction = (
            current
            /
            total
        )
        self.progress.set_fraction(
            fraction
```

```
        )
        self.progress.set_text(
            f"{current}/{total} "
            f"({fraction * 100:.1f}%)"
        )
        self.status.set_text(
            f"Processing segment "
            f"{current}/{total}"
        )
        self.update_live_view()
        return False
    # ================================================================
    # UPDATE LIVE VIEW
    # ================================================================
    def update_live_view(self):
        dataframe = (
            self.simulator.dataframe()
        )
        if dataframe.empty:
            return False
        self.update_packet_table()
        self.update_metrics()
        self.update_graph()
        return False
    # ================================================================
    # UPDATE GRAPH
    # ================================================================
    def update_graph(self):
        dataframe = (
            self.simulator.dataframe()
        )
        if dataframe.empty:
            return
        # ------------------------------------------------------------
        # CLEAR PREVIOUS FIGURE
        # ------------------------------------------------------------
        self.figure.clear()
        x = (
            dataframe["segment_id"]
        )
        # ============================================================
        # 1. RTT
        # ============================================================
        ax1 = self.figure.add_subplot(
            5,
            2,
```

```
            1
        )
        ax1.plot(
            x,
            dataframe["rtt_ms"],
            linewidth=1.5
        )
        ax1.set_title(
            "Round Trip Time (RTT)",
            fontsize=11,
            fontweight="bold"
        )
        ax1.set_xlabel(
            "Segment"
        )
        ax1.set_ylabel(
            "RTT (ms)"
        )
        ax1.grid(
            True,
            alpha=0.3
        )
        # ============================================================
        # 2. TRANSPORT DELAY
        # ============================================================
        ax2 = self.figure.add_subplot(
            5,
            2,
            2
        )
        ax2.plot(
            x,
            dataframe["delay_ms"],
            linewidth=1.5
        )
        ax2.set_title(
            "Transport Delay",
            fontsize=11,
            fontweight="bold"
        )
        ax2.set_xlabel(
            "Segment"
        )
        ax2.set_ylabel(
            "Delay (ms)"
        )
```

```
        ax2.grid(
            True,
            alpha=0.3
        )
        # ============================================================
        # 3. TCP CONGESTION WINDOW
        # ============================================================
        ax3 = self.figure.add_subplot(
            5,
            2,
            3
        )
        if self.simulator.protocol == "TCP":
            ax3.plot(
                x,
                dataframe["cwnd"],
                linewidth=2
            )
            ax3.set_title(
                "TCP Congestion Window (cwnd)",
                fontsize=11,
                fontweight="bold"
            )
            ax3.set_ylabel(
                "cwnd"
            )
        else:
            ax3.text(
                0.5,
                0.5,
                "UDP does not use TCP cwnd",
                ha="center",
                va="center",
                transform=ax3.transAxes
            )
            ax3.set_title(
                "Congestion Window"
            )
        ax3.set_xlabel(
            "Segment"
        )
        ax3.grid(
            True,
            alpha=0.3
        )
        # ============================================================
```

```
        # 4. PACKET DELIVERY RATIO
        # ============================================================
        ax4 = self.figure.add_subplot(
            5,
            2,
            4
        )
        ax4.plot(
            x,
            dataframe["pdr"],
            linewidth=2
        )
        ax4.set_title(
            "Packet Delivery Ratio (PDR)",
            fontsize=11,
            fontweight="bold"
        )
        ax4.set_xlabel(
            "Segment"
        )
        ax4.set_ylabel(
            "PDR (%)"
        )
        ax4.set_ylim(
            0,
            105
        )
        ax4.grid(
            True,
            alpha=0.3
        )
        # ============================================================
        # 5. THROUGHPUT
        # ============================================================
        ax5 = self.figure.add_subplot(
            5,
            2,
            5
        )
        throughput = (
            dataframe[
                "throughput_bps"
            ].to_numpy()
            /
            1_000_000.0
        )
```

```
        ax5.plot(
            x,
            throughput,
            linewidth=2
        )
        ax5.set_title(
            "Transport Layer Throughput",
            fontsize=11,
            fontweight="bold"
        )
        ax5.set_xlabel(
            "Segment"
        )
        ax5.set_ylabel(
            "Throughput (Mbps)"
        )
        ax5.grid(
            True,
            alpha=0.3
        )
        # ============================================================
        # 6. RETRANSMISSIONS
        # ============================================================
        ax6 = self.figure.add_subplot(
            5,
            2,
            6
        )
        ax6.plot(
            x,
            dataframe[
                "total_retransmissions"
            ],
            linewidth=2
        )
        ax6.set_title(
            "Cumulative TCP Retransmissions",
            fontsize=11,
            fontweight="bold"
        )
        ax6.set_xlabel(
            "Segment"
        )
        ax6.set_ylabel(
            "Retransmissions"
        )
```

```
        ax6.grid(
            True,
            alpha=0.3
        )
        # ============================================================
        # 7. DELIVERED VS LOST
        # ============================================================
        ax7 = self.figure.add_subplot(
            5,
            2,
            7
        )
        ax7.plot(
            x,
            dataframe["delivered"],
            linewidth=2,
            label="Delivered"
        )
        ax7.plot(
            x,
            dataframe["lost"],
            linewidth=2,
            label="Lost"
        )
        ax7.set_title(
            "Delivered vs Lost",
            fontsize=11,
            fontweight="bold"
        )
        ax7.set_xlabel(
            "Segment"
        )
        ax7.set_ylabel(
            "Count"
        )
        ax7.legend(
            fontsize=8
        )
        ax7.grid(
            True,
            alpha=0.3
        )
        # ============================================================
        # 8. ACK
        # ============================================================
        ax8 = self.figure.add_subplot(
```

```
            5,
            2,
            8
        )
        ax8.plot(
            x,
            dataframe["acknowledged"],
            linewidth=2
        )
        ax8.set_title(
            "Cumulative TCP ACK",
            fontsize=11,
            fontweight="bold"
        )
        ax8.set_xlabel(
            "Segment"
        )
        ax8.set_ylabel(
            "ACK Count"
        )
        ax8.grid(
            True,
            alpha=0.3
        )
        # ============================================================
        # 9. RTO
        # ============================================================
        ax9 = self.figure.add_subplot(
            5,
            2,
            9
        )
        if self.simulator.protocol == "TCP":
            ax9.plot(
                x,
                dataframe["rto_ms"],
                linewidth=2
            )
            ax9.set_title(
                "TCP Retransmission Timeout (RTO)",
                fontsize=11,
                fontweight="bold"
            )
            ax9.set_ylabel(
                "RTO (ms)"
            )
```

```
        else:
            ax9.text(
                0.5,
                0.5,
                "UDP does not use RTO",
                ha="center",
                va="center",
                transform=ax9.transAxes
            )
            ax9.set_title(
                "Retransmission Timeout"
            )
        ax9.set_xlabel(
            "Segment"
        )
        ax9.grid(
            True,
            alpha=0.3
        )
        # ============================================================
        # 10. EMPTY AREA
        # ============================================================
        ax10 = self.figure.add_subplot(
            5,
            2,
            10
        )
        ax10.axis(
            "off"
        )
        # ============================================================
        # GLOBAL TITLE
        # ============================================================
        self.figure.suptitle(
            f"{self.simulator.protocol} "
            "Transport Layer Simulation",
            fontsize=15,
            fontweight="bold"
        )
        # ============================================================
        # LAYOUT
        # ============================================================
        self.figure.tight_layout(
            rect=[
                0.03,
                0.02,
```

```
                0.98,
                0.96
            ],
            h_pad=2.5,
            w_pad=2.0
        )
        self.canvas.draw_idle()
    # ================================================================
    # UPDATE PACKET TABLE
    # ================================================================
    def update_packet_table(self):
        self.packet_store.clear()
        for segment in (
            self.simulator.segments
        ):
            self.packet_store.append(
                [
                    segment.segment_id,
                    segment.protocol,
                    segment.source_port,
                    segment.destination_port,
                    segment.sequence_number,
                    segment.status,
                    round(
                        segment.delay_ms,
                        2
                    ),
                    round(
                        segment.rtt_ms,
                        2
                    ),
                    segment.retransmissions
                ]
            )
    # ================================================================
    # UPDATE METRICS
    # ================================================================
    def update_metrics(self):
        metrics = (
            self.simulator.get_metrics()
        )
        # ------------------------------------------------------------
        # GENERATED
        # ------------------------------------------------------------
        self.metric_labels[
            "generated"
```

```
        ].set_markup(
            f"<b>{metrics['generated']}</b>"
        )
        # ------------------------------------------------------------
        # DELIVERED
        # ------------------------------------------------------------
        self.metric_labels[
            "delivered"
        ].set_markup(
            f"<b>{metrics['delivered']}</b>"
        )
        # ------------------------------------------------------------
        # LOST
        # ------------------------------------------------------------
        self.metric_labels[
            "lost"
        ].set_markup(
            f"<b>{metrics['lost']}</b>"
        )
        # ------------------------------------------------------------
        # ACK
        # ------------------------------------------------------------
        self.metric_labels[
            "acknowledged"
        ].set_markup(
            f"<b>{metrics['acknowledged']}</b>"
        )
        # ------------------------------------------------------------
        # RETRANSMISSION
        # ------------------------------------------------------------
        self.metric_labels[
            "retransmissions"
        ].set_markup(
            f"<b>{metrics['retransmissions']}</b>"
        )
        # ------------------------------------------------------------
        # PDR
        # ------------------------------------------------------------
        self.metric_labels[
            "pdr"
        ].set_markup(
            f"<b>{metrics['pdr']:.2f}%</b>"
        )
        # ------------------------------------------------------------
        # LOSS RATE
        # ------------------------------------------------------------
```

```
        self.metric_labels[
            "loss_rate"
        ].set_markup(
            f"<b>{metrics['loss_rate']:.2f}%</b>"
        )
        # ------------------------------------------------------------
        # RTT
        # ------------------------------------------------------------
        self.metric_labels[
            "average_rtt"
        ].set_markup(
            f"<b>{metrics['average_rtt']:.2f} ms</b>"
        )
        # ------------------------------------------------------------
        # DELAY
        # ------------------------------------------------------------
        self.metric_labels[
            "average_delay"
        ].set_markup(
            f"<b>{metrics['average_delay']:.2f} ms</b>"
        )
        # ------------------------------------------------------------
        # THROUGHPUT
        # ------------------------------------------------------------
        throughput_mbps = (
            metrics["throughput"]
            /
            1_000_000.0
        )
        self.metric_labels[
            "throughput"
        ].set_markup(
            f"<b>{throughput_mbps:.3f} Mbps</b>"
        )
    # ================================================================
    # SIMULATION FINISHED
    # ================================================================
    def simulation_finished(self):
        self.start_button.set_sensitive(
            True
        )
        self.update_packet_table()
        self.update_metrics()
        self.update_graph()
        metrics = (
            self.simulator.get_metrics()
```

```
        )
        protocol = (
            self.simulator.protocol
        )
        self.status.set_text(
            f"{protocol} simulation completed | "
            f"Generated={metrics['generated']} | "
            f"Delivered={metrics['delivered']} | "
            f"Lost={metrics['lost']} | "
            f"Retransmissions="
            f"{metrics['retransmissions']} | "
            f"PDR={metrics['pdr']:.2f}%"
        )
        return False
    # ================================================================
    # STOP SIMULATION
    # ================================================================
    def stop_simulation(
        self,
        button
    ):
        self.simulator.stop()
        self.status.set_text(
            "Simulation stopped."
        )
        self.start_button.set_sensitive(
            True
        )
    # ================================================================
    # EXPORT CSV
    # ================================================================
    def export_csv(
        self,
        button
    ):
        dataframe = (
            self.simulator.dataframe()
        )
        if dataframe.empty:
            self.error_dialog(
                "No simulation data available."
            )
            return

        dialog = Gtk.FileChooserDialog(
            title=
```

```
                "Export Transport Layer CSV",
            parent=
                self,
            action=
                Gtk.FileChooserAction.SAVE
        )
        dialog.add_buttons(
            Gtk.STOCK_CANCEL,
            Gtk.ResponseType.CANCEL,
            Gtk.STOCK_SAVE,
            Gtk.ResponseType.OK
        )
        dialog.set_current_name(
            "transport_layer_results.csv"
        )
        response = dialog.run()
        if response == Gtk.ResponseType.OK:
            filename = (
                dialog.get_filename()
            )
            try:
                dataframe.to_csv(
                    filename,
                    index=False
                )
                self.status.set_text(
                    "CSV exported: "
                    +
                    filename
                )
            except Exception as error:
                self.error_dialog(
                    str(error)
                )
        dialog.destroy()
    # ================================================================
    # ERROR DIALOG
    # ================================================================
    def error_dialog(
        self,
        message
    ):
        dialog = Gtk.MessageDialog(
            parent=self,
            flags=
                Gtk.DialogFlags.MODAL,
```

```
            message_type=
                Gtk.MessageType.ERROR,
            buttons=
                Gtk.ButtonsType.OK,
            text=
                "Simulation Error"
        )
        dialog.format_secondary_text(
            message
        )
        dialog.run()
        dialog.destroy()
# ======================================================================
# MAIN
# ======================================================================
def main():
    window = TransportLayerGTK()
    window.connect(
        "destroy",
        Gtk.main_quit
    )
    window.show_all()
    # ================================================================
    # MAXIMIZE WINDOW
    # ================================================================
    window.maximize()
    Gtk.main()
# ======================================================================
# ENTRY POINT
# ======================================================================
if __name__ == "__main__":
    main()
```

Aplikasi ini adalah simulator *Transport Layer* yang menampilkan hasil simulasi dalam antar muka grafis pustaka GTK dan grafik dengan pustaka Matplotlib. Arsitektur aplikasi simulator *Transport Layer* dapat kita lihat pada gambar 2.14: Pembaca dapat melihat penjelasan kode sumber pada keterangan simulator Transport Layer.

2.6.1 Tampilan Simulator Transport Layer

Tampilan dari pilihan protokol TCP pada simulator Transport Layer dapat pembaca lihat pada Gambar 2.15:

Tampilan dari pilihan protokol UDP pada simulator Transport Layer dapat pembaca lihat pada Gambar 2.16:

2.7 Komponen *Session Layer*

Komponen *Session Layer* merupakan aplikasi yang dikembangkan dengan bahasa pemrograman Python untuk mendeskripsikan simulasi pada *Session Layer* [14] 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_session_layer_research_simulation.py:

Listing 2.7: Transport Layer Simulation

```
#!/usr/bin/env python3

"""
================================================================================
SESSION LAYER SIMULATION
================================================================================

Python 3
GTK3 / PyGObject
Matplotlib
NumPy
Pandas

OSI SESSION LAYER SIMULATION

Main concepts modeled:

    1. Session establishment
    2. Session authentication
    3. Session synchronization
    4. Dialog control
    5. Data exchange
    6. Checkpoint creation
    7. Keep-alive
    8. Session timeout
    9. Synchronization loss
   10. Session recovery
   11. Recovery from checkpoint
   12. Session termination

Simulation metrics:

    - Session state
    - Session duration
    - Response time
    - Synchronization point
    - Checkpoint count
    - Synchronization loss
    - Recovery attempts
```

```
    - Successful recovery
    - Session reliability
    - Recovery rate
    - Keep-alive count
    - Timeout count
    - Data exchanged
    - Events processed

GUI:

    - GTK3
    - Resizable
    - Maximizable
    - Responsive parameter area
    - Live Matplotlib graph
    - Vertical graph scrolling
    - Event log
    - CSV export
    - Process explanation

Run:

    python3 final_session_layer_research_simulation.py

======================================================================
"""

import gi

gi.require_version("Gtk", "3.0")

from gi.repository import Gtk, GLib

import matplotlib

matplotlib.use("GTK3Agg")

from matplotlib.figure import Figure
from matplotlib.backends.backend_gtk3agg import FigureCanvasGTK3Agg

import numpy as np
import pandas as pd

import random
import time
import threading
```

```


# ======================================================================
# SESSION EVENT
# ======================================================================

class SessionEvent:

    def __init__(
        self,
        event_id,
        event_type,
        state,
        sequence_number,
        checkpoint,
        response_time_ms,
        status,
        data_bytes=0
    ):

        self.event_id = event_id

        self.event_type = event_type

        self.state = state

        self.sequence_number = sequence_number

        self.checkpoint = checkpoint

        self.response_time_ms = response_time_ms

        self.status = status

        self.data_bytes = data_bytes

        self.timestamp = time.time()


# ======================================================================
# SESSION STATE MACHINE
# ======================================================================

class SessionStateMachine:

    CLOSED = "CLOSED"
```

```
    ESTABLISHING = "ESTABLISHING"

    AUTHENTICATING = "AUTHENTICATING"

    ESTABLISHED = "ESTABLISHED"

    SYNCHRONIZING = "SYNCHRONIZING"

    DATA_TRANSFER = "DATA_TRANSFER"

    RECOVERING = "RECOVERING"

    TERMINATING = "TERMINATING"

    TIMEOUT = "TIMEOUT"

    RECOVERY_FAILED = "RECOVERY_FAILED"


# ======================================================================
# SESSION LAYER SIMULATOR
# ======================================================================

class SessionLayerSimulator:

    def __init__(
        self,
        event_count=100,
        checkpoint_interval=10,
        synchronization_probability=0.02,
        timeout_probability=0.01,
        recovery_probability=0.90,
        min_response_time=5.0,
        max_response_time=50.0,
        data_size=1024,
        keepalive_interval=10
    ):

        self.event_count = event_count

        self.checkpoint_interval = (
            checkpoint_interval
        )

        self.synchronization_probability = (
```

```
            synchronization_probability
        )

        self.timeout_probability = (
            timeout_probability
        )

        self.recovery_probability = (
            recovery_probability
        )

        self.min_response_time = (
            min_response_time
        )

        self.max_response_time = (
            max_response_time
        )

        self.data_size = data_size

        self.keepalive_interval = (
            keepalive_interval
        )

        # ------------------------------------------------------------
        # SESSION STATE
        # ------------------------------------------------------------

        self.state = (
            SessionStateMachine.CLOSED
        )

        self.previous_state = (
            SessionStateMachine.CLOSED
        )

        self.session_id = None

        self.sequence_number = 0

        self.last_checkpoint = 0

        self.checkpoint_count = 0

        # ------------------------------------------------------------
```

```
        # EVENT STORAGE
        # ------------------------------------------------------------

        self.events = []

        self.statistics = []

        # ------------------------------------------------------------
        # EXECUTION
        # ------------------------------------------------------------

        self.running = False

        self.start_time = None

        self.end_time = None

        # ------------------------------------------------------------
        # COUNTERS
        # ------------------------------------------------------------

        self.generated_events = 0

        self.processed_events = 0

        self.successful_events = 0

        self.failed_events = 0

        self.synchronization_losses = 0

        self.recovery_attempts = 0

        self.successful_recoveries = 0

        self.keepalive_count = 0

        self.timeout_count = 0

        self.total_data_bytes = 0

        self.total_response_time = 0.0

        self.session_establishment_time = 0.0

        self.session_termination_time = 0.0
```

```
270
271         # ------------------------------------------------------------
272         # INTERNAL FLAGS
273         # ------------------------------------------------------------
274
275         self.session_established = False
276
277         self.authenticated = False
278
279         self.synchronized = False
280
281     # ====================================================================
282     # RESET
283     # ====================================================================
284
285     def reset(self):
286
287         self.state = (
288             SessionStateMachine.CLOSED
289         )
290
291         self.previous_state = (
292             SessionStateMachine.CLOSED
293         )
294
295         self.session_id = None
296
297         self.sequence_number = 0
298
299         self.last_checkpoint = 0
300
301         self.checkpoint_count = 0
302
303         self.events = []
304
305         self.statistics = []
306
307         self.running = False
308
309         self.start_time = None
310
311         self.end_time = None
312
313         self.generated_events = 0
314
315         self.processed_events = 0
```

```
316
317         self.successful_events = 0
318
319         self.failed_events = 0
320
321         self.synchronization_losses = 0
322
323         self.recovery_attempts = 0
324
325         self.successful_recoveries = 0
326
327         self.keepalive_count = 0
328
329         self.timeout_count = 0
330
331         self.total_data_bytes = 0
332
333         self.total_response_time = 0.0
334
335         self.session_establishment_time = 0.0
336
337         self.session_termination_time = 0.0
338
339         self.session_established = False
340
341         self.authenticated = False
342
343         self.synchronized = False
344
345     # ================================================================
346     # GENERATE SESSION ID
347     # ================================================================
348
349     def generate_session_id(self):
350
351         self.session_id = (
352
353             "SES-"
354             +
355             str(
356                 random.randint(
357                     100000,
358                     999999
359                 )
360             )
361
```

```
        )

        return self.session_id

    # ================================================================
    # CHANGE STATE
    # ================================================================

    def change_state(
        self,
        new_state
    ):

        self.previous_state = self.state

        self.state = new_state

    # ================================================================
    # RESPONSE TIME
    # ================================================================

    def random_response_time(self):

        return random.uniform(
            self.min_response_time,
            self.max_response_time
        )

    # ================================================================
    # ESTABLISH SESSION
    # ================================================================

    def establish_session(self):

        start = time.time()

        self.change_state(
            SessionStateMachine.ESTABLISHING
        )

        self.generate_session_id()

        response_1 = (
            self.random_response_time()
        )

```

```
        self.change_state(
            SessionStateMachine.AUTHENTICATING
        )

        response_2 = (
            self.random_response_time()
        )

        self.authenticated = True

        self.change_state(
            SessionStateMachine.SYNCHRONIZING
        )

        response_3 = (
            self.random_response_time()
        )

        self.synchronized = True

        self.change_state(
            SessionStateMachine.ESTABLISHED
        )

        self.session_established = True

        self.session_establishment_time = (

            time.time()
            -
            start

        ) * 1000.0

        self.session_establishment_time += (

            response_1
            +
            response_2
            +
            response_3

        )

        return self.session_establishment_time

```

```
    # ================================================================
    # CREATE CHECKPOINT
    # ================================================================

    def create_checkpoint(self):

        self.last_checkpoint = (
            self.sequence_number
        )

        self.checkpoint_count += 1

        return self.last_checkpoint

    # ================================================================
    # KEEP ALIVE
    # ================================================================

    def keepalive(self):

        self.keepalive_count += 1

        response = (
            self.random_response_time()
        )

        self.total_response_time += response

        return response

    # ================================================================
    # SYNCHRONIZATION LOSS
    # ================================================================

    def synchronization_loss(self):

        self.synchronization_losses += 1

        self.synchronized = False

        self.change_state(
            SessionStateMachine.RECOVERING
        )

    # ================================================================
    # SESSION RECOVERY
```

```
    # ================================================================

    def recover_session(self):

        self.recovery_attempts += 1

        recovery_response = (
            self.random_response_time()
        )

        success = (

            random.random()
            <
            self.recovery_probability

        )

        if success:

            self.successful_recoveries += 1

            self.synchronized = True

            self.change_state(
                SessionStateMachine.ESTABLISHED
            )

            self.sequence_number = (
                self.last_checkpoint
            )

            return (
                True,
                recovery_response
            )

        self.change_state(
            SessionStateMachine.RECOVERY_FAILED
        )

        return (
            False,
            recovery_response
        )

```

```
    # ================================================================
    # SESSION TIMEOUT
    # ================================================================

    def timeout(self):

        self.timeout_count += 1

        self.change_state(
            SessionStateMachine.TIMEOUT
        )

        self.synchronized = False

    # ================================================================
    # DATA EXCHANGE
    # ================================================================

    def exchange_data(
        self,
        event_id
    ):

        self.change_state(
            SessionStateMachine.DATA_TRANSFER
        )

        response = (
            self.random_response_time()
        )

        self.sequence_number += 1

        self.total_data_bytes += (
            self.data_size
        )

        # ------------------------------------------------------------
        # CHECKPOINT
        # ------------------------------------------------------------

        checkpoint_created = False

        if (

            self.sequence_number
```

```
            %
            self.checkpoint_interval

            ==

            0

        ):

            self.create_checkpoint()

            checkpoint_created = True

        # ------------------------------------------------------------
        # KEEP ALIVE
        # ------------------------------------------------------------

        if (

            event_id
            %
            self.keepalive_interval

            ==

            0

        ):

            self.keepalive()

        self.change_state(
            SessionStateMachine.ESTABLISHED
        )

        return (
            response,
            checkpoint_created
        )

    # ============================================================
    # TERMINATE SESSION
    # ============================================================

    def terminate_session(self):

```

```
        start = time.time()

        self.change_state(
            SessionStateMachine.TERMINATING
        )

        response = (
            self.random_response_time()
        )

        self.change_state(
            SessionStateMachine.CLOSED
        )

        self.session_termination_time = (

            time.time()
            -
            start

        ) * 1000.0

        self.session_termination_time += response

        self.session_established = False

        self.authenticated = False

        self.synchronized = False

        return self.session_termination_time

    # ================================================================
    # PROCESS ONE EVENT
    # ================================================================

    def process_event(
        self,
        event_id
    ):

        self.generated_events += 1

        # ------------------------------------------------------------
        # SESSION MUST EXIST
        # ------------------------------------------------------------
```

```
        if not self.session_established:
            self.failed_events += 1
            return SessionEvent(
                event_id,
                "SESSION_NOT_ESTABLISHED",
                self.state,
                self.sequence_number,
                self.last_checkpoint,
                0.0,
                "FAILED"
            )
        # ------------------------------------------------------------
        # SYNCHRONIZATION LOSS
        # ------------------------------------------------------------
        synchronization_lost = (
            random.random()
            <
            self.synchronization_probability
        )
        if synchronization_lost:
            self.synchronization_loss()
            response = (
                self.random_response_time()
            )
            self.failed_events += 1
            event = SessionEvent(
```

```
                event_id,

                "SYNCHRONIZATION_LOSS",

                self.state,

                self.sequence_number,

                self.last_checkpoint,

                response,

                "RECOVERY_REQUIRED"

            )

            self.events.append(
                event
            )

            return event

        # ------------------------------------------------------------
        # TIMEOUT
        # ------------------------------------------------------------

        timeout_event = (

            random.random()
            <
            self.timeout_probability

        )

        if timeout_event:

            self.timeout()

            response = (
                self.random_response_time()
            )

            self.failed_events += 1

            event = SessionEvent(
```

```
                event_id,

                "SESSION_TIMEOUT",

                self.state,

                self.sequence_number,

                self.last_checkpoint,

                response,

                "TIMEOUT"

            )

            self.events.append(
                event
            )

            return event

        # ------------------------------------------------------------
        # DATA EXCHANGE
        # ------------------------------------------------------------

        response, checkpoint_created = (
            self.exchange_data(
                event_id
            )
        )

        self.processed_events += 1

        self.successful_events += 1

        self.total_response_time += response

        event_type = (
            "DATA_EXCHANGE"
        )

        if checkpoint_created:

            event_type = (
```

```
                "DATA_EXCHANGE_CHECKPOINT"
            )

        event = SessionEvent(

            event_id,

            event_type,

            self.state,

            self.sequence_number,

            self.last_checkpoint,

            response,

            "SUCCESS",

            self.data_size

        )

        self.events.append(
            event
        )

        return event

    # ================================================================
    # RUN SIMULATION
    # ================================================================

    def run(
        self,
        progress_callback=None,
        event_callback=None
    ):

        self.reset()

        self.running = True

        self.start_time = time.time()

        # ------------------------------------------------------------
```

```
        # SESSION ESTABLISHMENT
        # ------------------------------------------------------------

        establishment_time = (
            self.establish_session()
        )

        establishment_event = SessionEvent(

            0,

            "SESSION_ESTABLISHED",

            self.state,

            self.sequence_number,

            self.last_checkpoint,

            establishment_time,

            "SUCCESS"

        )

        self.events.append(
            establishment_event
        )

        if event_callback:

            event_callback(
                establishment_event
            )

        # ------------------------------------------------------------
        # EVENT PROCESSING
        # ------------------------------------------------------------

        for event_id in range(
            1,
            self.event_count + 1
        ):

            if not self.running:

```

```
                break

            event = (
                self.process_event(
                    event_id
                )
            )

            # ------------------------------------------------------
            # RECOVERY
            # ------------------------------------------------------

            if event.status in (

                "RECOVERY_REQUIRED",

                "TIMEOUT"

            ):

                recovery_success, recovery_time = (
                    self.recover_session()
                )

                recovery_status = (
                    "RECOVERED"
                    if recovery_success
                    else
                    "RECOVERY_FAILED"
                )

                recovery_event = SessionEvent(

                    event_id,

                    "SESSION_RECOVERY",

                    self.state,

                    self.sequence_number,

                    self.last_checkpoint,

                    recovery_time,

                    recovery_status
```

```
                )

                self.events.append(
                    recovery_event
                )

                if event_callback:

                    event_callback(
                        recovery_event
                    )

            if event_callback:

                event_callback(
                    event
                )

            # ------------------------------------------------------------
            # METRICS
            # ------------------------------------------------------------

            elapsed = (

                time.time()
                -
                self.start_time

            )

            total_events = (
                self.generated_events
            )

            reliability = 0.0

            if total_events > 0:

                reliability = (

                    self.successful_events
                    /
                    total_events

                ) * 100.0
```

```
            recovery_rate = 0.0

            if self.recovery_attempts > 0:

                recovery_rate = (

                    self.successful_recoveries
                    /
                    self.recovery_attempts

                ) * 100.0

            average_response = 0.0

            if self.processed_events > 0:

                average_response = (

                    self.total_response_time
                    /
                    self.processed_events

                )

            self.statistics.append(

                {

                    "event_id":
                        event.event_id,

                    "event_type":
                        event.event_type,

                    "session_id":
                        self.session_id,

                    "state":
                        self.state,

                    "sequence_number":
                        self.sequence_number,

                    "checkpoint":
                        self.last_checkpoint,
```

```
                    "checkpoint_count":
                        self.checkpoint_count,

                    "response_time_ms":
                        event.response_time_ms,

                    "status":
                        event.status,

                    "data_bytes":
                        event.data_bytes,

                    "generated_events":
                        self.generated_events,

                    "processed_events":
                        self.processed_events,

                    "successful_events":
                        self.successful_events,

                    "failed_events":
                        self.failed_events,

                    "synchronization_losses":
                        self.synchronization_losses,

                    "recovery_attempts":
                        self.recovery_attempts,

                    "successful_recoveries":
                        self.successful_recoveries,

                    "keepalive_count":
                        self.keepalive_count,

                    "timeout_count":
                        self.timeout_count,

                    "reliability":
                        reliability,

                    "recovery_rate":
                        recovery_rate,

```

```
                    "average_response_ms":
                        average_response,

                    "elapsed_seconds":
                        elapsed,

                    "total_data_bytes":
                        self.total_data_bytes

                }

            )

            # ------------------------------------------------------------
            # GUI CALLBACK
            # ------------------------------------------------------------

            if progress_callback:

                progress_callback(

                    self.generated_events,

                    self.event_count

                )

            time.sleep(
                0.03
            )

        # ------------------------------------------------------------
        # SESSION TERMINATION
        # ------------------------------------------------------------

        if self.running:

            termination_time = (
                self.terminate_session()
            )

            termination_event = SessionEvent(

                self.event_count + 1,

                "SESSION_TERMINATED",
```

```
                self.state,

                self.sequence_number,

                self.last_checkpoint,

                termination_time,

                "SUCCESS"

            )

            self.events.append(
                termination_event
            )

            if event_callback:

                event_callback(
                    termination_event
                )

        self.end_time = time.time()

        self.running = False

    # ================================================================
    # STOP
    # ================================================================

    def stop(self):

        self.running = False

    # ================================================================
    # DATAFRAME
    # ================================================================

    def dataframe(self):

        return pd.DataFrame(
            self.statistics
        )

    # ================================================================
```

```
    # METRICS
    # ================================================================

    def get_metrics(self):

        elapsed = 0.0

        if (

            self.start_time is not None

            and

            self.end_time is not None

        ):

            elapsed = (

                self.end_time
                -
                self.start_time

            )

        reliability = 0.0

        if self.generated_events > 0:

            reliability = (

                self.successful_events
                /
                self.generated_events

            ) * 100.0

        recovery_rate = 0.0

        if self.recovery_attempts > 0:

            recovery_rate = (

                self.successful_recoveries
                /
                self.recovery_attempts
```

```
            ) * 100.0

        average_response = 0.0

        if self.processed_events > 0:

            average_response = (

                self.total_response_time
                /
                self.processed_events

            )

        return {

            "session_id":
                self.session_id,

            "state":
                self.state,

            "generated_events":
                self.generated_events,

            "processed_events":
                self.processed_events,

            "successful_events":
                self.successful_events,

            "failed_events":
                self.failed_events,

            "synchronization_losses":
                self.synchronization_losses,

            "recovery_attempts":
                self.recovery_attempts,

            "successful_recoveries":
                self.successful_recoveries,

            "keepalive_count":
                self.keepalive_count,
```

```
            "timeout_count":
                self.timeout_count,

            "checkpoint_count":
                self.checkpoint_count,

            "last_checkpoint":
                self.last_checkpoint,

            "reliability":
                reliability,

            "recovery_rate":
                recovery_rate,

            "average_response":
                average_response,

            "data_bytes":
                self.total_data_bytes,

            "elapsed":
                elapsed,

            "establishment":
                self.session_establishment_time,

            "termination":
                self.session_termination_time

        }


# ======================================================================
# GTK APPLICATION
# ======================================================================

class SessionLayerGTK(Gtk.Window):

    def __init__(self):

        super().__init__(
            title="Session Layer Simulation"
        )

```

```
        # ============================================================
        # WINDOW
        # ============================================================

        self.set_default_size(
            1280,
            800
        )

        self.set_size_request(
            900,
            600
        )

        self.set_border_width(
            8
        )

        self.set_resizable(
            True
        )

        self.set_position(
            Gtk.WindowPosition.CENTER
        )

        # ============================================================
        # SIMULATOR
        # ============================================================

        self.simulator = (
            SessionLayerSimulator()
        )

        self.worker_thread = None

        self.build_ui()

    # ================================================================
    # BUILD UI
    # ================================================================

    def build_ui(self):

        main_box = Gtk.Box(

```

```
            orientation=
                Gtk.Orientation.VERTICAL,

            spacing=6

        )

        self.add(
            main_box
        )

        # ============================================================
        # TITLE
        # ============================================================

        title = Gtk.Label()

        title.set_markup(

            "<span size='xx-large' "
            "weight='bold'>"
            "Session Layer Simulation"
            "</span>"

        )

        main_box.pack_start(

            title,

            False,
            False,
            3

        )

        subtitle = Gtk.Label(

            label=(
                "Session establishment     "
                "authentication     synchronization     "
                "data exchange     checkpoint     "
                "recovery     termination"
            )

        )
```

```
        main_box.pack_start(
            subtitle,
            False,
            False,
            2
        )
        # ============================================================
        # PARAMETERS
        # ============================================================
        parameter_frame = Gtk.Frame(
            label="Session Simulation Parameters"
        )
        parameter_grid = Gtk.Grid()
        parameter_grid.set_border_width(
            6
        )
        parameter_grid.set_row_spacing(
            5
        )
        parameter_grid.set_column_spacing(
            7
        )
        parameter_frame.add(
            parameter_grid
        )
        main_box.pack_start(
            parameter_frame,
            False,
            False,
            0
```

```
        )

        # ------------------------------------------------------------
        # EVENT COUNT
        # ------------------------------------------------------------

        parameter_grid.attach(

            Gtk.Label(
                label="Events:"
            ),

            0,
            0,
            1,
            1

        )

        self.event_entry = (
            Gtk.Entry()
        )

        self.event_entry.set_text(
            "100"
        )

        parameter_grid.attach(

            self.event_entry,

            1,
            0,
            1,
            1

        )

        # ------------------------------------------------------------
        # CHECKPOINT
        # ------------------------------------------------------------

        parameter_grid.attach(

            Gtk.Label(
```

```
                label="Checkpoint Interval:"
            ),

            2,
            0,
            1,
            1

        )

        self.checkpoint_entry = (
            Gtk.Entry()
        )

        self.checkpoint_entry.set_text(
            "10"
        )

        parameter_grid.attach(

            self.checkpoint_entry,

            3,
            0,
            1,
            1

        )

        # ------------------------------------------------------------
        # SYNCHRONIZATION LOSS
        # ------------------------------------------------------------

        parameter_grid.attach(

            Gtk.Label(
                label="Sync Loss Probability:"
            ),

            4,
            0,
            1,
            1

        )

```

```
        self.sync_loss_entry = (
            Gtk.Entry()
        )

        self.sync_loss_entry.set_text(
            "0.02"
        )

        parameter_grid.attach(

            self.sync_loss_entry,

            5,
            0,
            1,
            1

        )

        # ------------------------------------------------------------
        # TIMEOUT
        # ------------------------------------------------------------

        parameter_grid.attach(

            Gtk.Label(
                label="Timeout Probability:"
            ),

            0,
            1,
            1,
            1

        )

        self.timeout_entry = (
            Gtk.Entry()
        )

        self.timeout_entry.set_text(
            "0.01"
        )

        parameter_grid.attach(

```

```
            self.timeout_entry,

            1,
            1,
            1,
            1

        )

        # ------------------------------------------------------------
        # RECOVERY
        # ------------------------------------------------------------

        parameter_grid.attach(

            Gtk.Label(
                label="Recovery Probability:"
            ),

            2,
            1,
            1,
            1

        )

        self.recovery_entry = (
            Gtk.Entry()
        )

        self.recovery_entry.set_text(
            "0.90"
        )

        parameter_grid.attach(

            self.recovery_entry,

            3,
            1,
            1,
            1

        )

        # ------------------------------------------------------------
```

```
        # DATA SIZE
        # ------------------------------------------------------------

        parameter_grid.attach(

            Gtk.Label(
                label="Data Bytes/Event:"
            ),

            4,
            1,
            1,
            1

        )

        self.data_size_entry = (
            Gtk.Entry()
        )

        self.data_size_entry.set_text(
            "1024"
        )

        parameter_grid.attach(

            self.data_size_entry,

            5,
            1,
            1,
            1

        )

        # ============================================================
        # BUTTONS
        # ============================================================

        button_box = Gtk.Box(

            orientation=
                Gtk.Orientation.HORIZONTAL,

            spacing=7

```

```
        )

        self.start_button = Gtk.Button(

            label="Start Simulation"

        )

        self.start_button.connect(

            "clicked",

            self.start_simulation

        )

        button_box.pack_start(

            self.start_button,

            False,
            False,
            0

        )

        self.stop_button = Gtk.Button(

            label="Stop"

        )

        self.stop_button.connect(

            "clicked",

            self.stop_simulation

        )

        button_box.pack_start(

            self.stop_button,

            False,
            False,
```

```
            0

        )

        export_button = Gtk.Button(

            label="Export CSV"

        )

        export_button.connect(

            "clicked",

            self.export_csv

        )

        button_box.pack_start(

            export_button,

            False,
            False,
            0

        )

        parameter_grid.attach(

            button_box,

            0,
            2,
            6,
            1

        )

        # ============================================================
        # PROGRESS
        # ============================================================

        self.progress = (
            Gtk.ProgressBar()
        )
```

```
1788
1789            self.progress.set_show_text(
1790                True
1791            )
1792
1793            main_box.pack_start(
1794
1795                self.progress,
1796
1797                False,
1798                False,
1799                0
1800
1801            )
1802
1803            # ============================================================
1804            # METRICS
1805            # ============================================================
1806
1807            metric_frame = Gtk.Frame(
1808
1809                label="Session Layer Metrics"
1810
1811            )
1812
1813            metric_grid = Gtk.Grid()
1814
1815            metric_grid.set_border_width(
1816                5
1817            )
1818
1819            metric_grid.set_column_spacing(
1820                9
1821            )
1822
1823            metric_grid.set_row_spacing(
1824                4
1825            )
1826
1827            metric_frame.add(
1828                metric_grid
1829            )
1830
1831            main_box.pack_start(
1832
1833                metric_frame,
```

```
            False,
            False,
            0

        )

        self.metric_labels = {}

        metrics = [

            ("Events", "generated_events"),

            ("Processed", "processed_events"),

            ("Success", "successful_events"),

            ("Failed", "failed_events"),

            ("Sync Loss", "synchronization_losses"),

            ("Recovery", "recovery_attempts"),

            ("Recovered", "successful_recoveries"),

            ("Checkpoint", "checkpoint_count"),

            ("Keepalive", "keepalive_count"),

            ("Timeout", "timeout_count"),

            ("Reliability", "reliability"),

            ("Recovery Rate", "recovery_rate")

        ]

        for index, (
            caption,
            key
        ) in enumerate(metrics):

            metric_box = Gtk.Box(

                orientation=
                    Gtk.Orientation.VERTICAL,
```

```
                spacing=1

            )

            caption_label = Gtk.Label(

                label=caption

            )

            value_label = Gtk.Label(

                label="0"

            )

            value_label.set_markup(

                "<b>0</b>"

            )

            metric_box.pack_start(

                caption_label,

                False,
                False,
                0

            )

            metric_box.pack_start(

                value_label,

                False,
                False,
                0

            )

            metric_grid.attach(

                metric_box,
```

```
                index,
                0,
                1,
                1

            )

            self.metric_labels[key] = (
                value_label
            )

        # ============================================================
        # NOTEBOOK
        # ============================================================

        notebook = Gtk.Notebook()

        notebook.set_vexpand(
            True
        )

        notebook.set_hexpand(
            True
        )

        main_box.pack_start(

            notebook,

            True,
            True,
            0

        )

        # ============================================================
        # LIVE GRAPH TAB
        # ============================================================

        graph_box = Gtk.Box(

            orientation=
                Gtk.Orientation.VERTICAL,

            spacing=4
```

```
        )

        notebook.append_page(

            graph_box,

            Gtk.Label(

                label="Live Simulation Graph"

            )

        )

        # ------------------------------------------------------------
        # SCROLL WINDOW
        # ------------------------------------------------------------

        self.graph_scroll = (
            Gtk.ScrolledWindow()
        )

        self.graph_scroll.set_policy(

            Gtk.PolicyType.NEVER,

            Gtk.PolicyType.AUTOMATIC

        )

        self.graph_scroll.set_shadow_type(

            Gtk.ShadowType.IN

        )

        self.graph_scroll.set_vexpand(
            True
        )

        self.graph_scroll.set_hexpand(
            True
        )

        graph_box.pack_start(
```

```
            self.graph_scroll,

            True,
            True,
            0

        )

        # ------------------------------------------------------------
        # GRAPH CONTAINER
        # ------------------------------------------------------------

        self.graph_container = Gtk.Box(

            orientation=
                Gtk.Orientation.VERTICAL

        )

        self.graph_scroll.add(

            self.graph_container

        )

        # ------------------------------------------------------------
        # MATPLOTLIB
        # ------------------------------------------------------------

        self.figure = Figure(

            figsize=(13, 16),

            dpi=100

        )

        self.canvas = (

            FigureCanvasGTK3Agg(

                self.figure

            )

```

```
        )

        # Intentionally taller than viewport.
        # Gtk.ScrolledWindow provides vertical scrolling.

        self.canvas.set_size_request(

            1100,

            1600

        )

        self.graph_container.pack_start(

            self.canvas,

            False,
            False,
            0

        )

        # ============================================================
        # EVENT LOG TAB
        # ============================================================

        event_box = Gtk.Box(

            orientation=
                Gtk.Orientation.VERTICAL

        )

        notebook.append_page(

            event_box,

            Gtk.Label(

                label="Session Event Log"

            )

        )

```

```
        self.event_store = Gtk.ListStore(

            int,
            str,
            str,
            int,
            int,
            float,
            str,
            int

        )

        event_tree = Gtk.TreeView(

            model=self.event_store

        )

        event_columns = [

            ("ID", 0),

            ("Event", 1),

            ("State", 2),

            ("Sequence", 3),

            ("Checkpoint", 4),

            ("Response ms", 5),

            ("Status", 6),

            ("Data Bytes", 7)

        ]

        for title, index in event_columns:

            renderer = (
                Gtk.CellRendererText()
            )

            column = (
```

```
                Gtk.TreeViewColumn(

                    title,

                    renderer,

                    text=index

                )

            )

            event_tree.append_column(
                column
            )

        event_scroll = (
            Gtk.ScrolledWindow()
        )

        event_scroll.set_policy(

            Gtk.PolicyType.AUTOMATIC,

            Gtk.PolicyType.AUTOMATIC

        )

        event_scroll.add(
            event_tree
        )

        event_box.pack_start(

            event_scroll,

            True,
            True,
            0

        )

        # ============================================================
        # SESSION PROCESS TAB
        # ============================================================
```

```
2202
2203        process_box = Gtk.Box(
2204
2205            orientation=
2206                Gtk.Orientation.VERTICAL,
2207
2208            spacing=10
2209
2210        )
2211
2212        notebook.append_page(
2213
2214            process_box,
2215
2216            Gtk.Label(
2217
2218                label="Session Process"
2219
2220            )
2221
2222        )
2223
2224        process_scroll = (
2225            Gtk.ScrolledWindow()
2226        )
2227
2228        process_scroll.set_policy(
2229
2230            Gtk.PolicyType.NEVER,
2231
2232            Gtk.PolicyType.AUTOMATIC
2233
2234        )
2235
2236        process_box.pack_start(
2237
2238            process_scroll,
2239
2240            True,
2241            True,
2242            0
2243
2244        )
2245
2246        process_label = Gtk.Label()
2247
```

```
        process_label.set_markup(

            "<b>OSI SESSION LAYER PROCESS</b>\n\n"

            "Application Layer\n"
            "           \n"
            "Session Request\n"
            "           \n"
            "Session Establishment\n"
            "           \n"
            "Session ID Assignment\n"
            "           \n"
            "Authentication\n"
            "           \n"
            "Synchronization\n"
            "           \n"
            "Session Established\n"
            "           \n"
            "Dialog Control\n"
            "           \n"
            "Data Exchange\n"
            "           \n"
            "Checkpoint Creation\n"
            "           \n"
            "Keep-Alive\n"
            "           \n"
            "Synchronization Monitoring\n"
            "           \n"
            "Synchronization Loss?\n"
            "           \n"
            "Session Recovery\n"
            "           \n"
            "Restore from Checkpoint\n"
            "           \n"
            "Resume Data Exchange\n"
            "           \n"
            "Session Termination\n"
            "           \n"
            "CLOSED\n\n"

            "<b>SESSION LAYER FUNCTIONS MODELED</b>\n\n"

            "    Session establishment\n"
            "    Session maintenance\n"
            "    Session termination\n"
            "    Dialog control\n"
```

```
            "    Synchronization\n"
            "    Checkpointing\n"
            "    Session recovery\n"
            "    Keep-alive\n"
            "    Timeout detection\n"
            "    State management\n\n"

            "<b>SIMULATION METRICS</b>\n\n"

            "    Session reliability\n"
            "    Recovery rate\n"
            "    Synchronization losses\n"
            "    Recovery attempts\n"
            "    Checkpoint count\n"
            "    Keep-alive count\n"
            "    Timeout count\n"
            "    Response time\n"
            "    Session duration\n"
            "    Data exchanged"

        )

        process_label.set_xalign(
            0
        )

        process_label.set_yalign(
            0
        )

        process_label.set_margin_start(
            20
        )

        process_label.set_margin_top(
            20
        )

        process_scroll.add(
            process_label
        )

        # ============================================================
        # STATUS
        # ============================================================

```

```
        self.status = Gtk.Label(

            label="Ready."

        )

        main_box.pack_start(

            self.status,

            False,
            False,
            0

        )

    # ================================================================
    # GET PARAMETERS
    # ================================================================

    def get_parameters(self):

        try:

            event_count = int(

                self.event_entry
                .get_text()

            )

            checkpoint_interval = int(

                self.checkpoint_entry
                .get_text()

            )

            sync_loss_probability = float(

                self.sync_loss_entry
                .get_text()

            )

            timeout_probability = float(
```

```
                self.timeout_entry
                .get_text()

            )

            recovery_probability = float(

                self.recovery_entry
                .get_text()

            )

            data_size = int(

                self.data_size_entry
                .get_text()

            )

            # ---------------------------------------------------------
            # VALIDATION
            # ---------------------------------------------------------

            if event_count <= 0:

                raise ValueError(

                    "Event count must be greater than zero."

                )

            if checkpoint_interval <= 0:

                raise ValueError(

                    "Checkpoint interval must be greater than zero."

                )

            if not (

                0.0
                <=
                sync_loss_probability
                <=
```

```
                1.0

            ):

                raise ValueError(

                    "Synchronization loss probability "
                    "must be between 0 and 1."

                )

            if not (

                0.0
                <=
                timeout_probability
                <=
                1.0

            ):

                raise ValueError(

                    "Timeout probability "
                    "must be between 0 and 1."

                )

            if not (

                0.0
                <=
                recovery_probability
                <=
                1.0

            ):

                raise ValueError(

                    "Recovery probability "
                    "must be between 0 and 1."

                )

            if data_size <= 0:
```

```
                raise ValueError(
                    "Data size must be greater than zero."
                )
            return (
                event_count,
                checkpoint_interval,
                sync_loss_probability,
                timeout_probability,
                recovery_probability,
                data_size
            )
        except ValueError as error:
            self.error_dialog(
                str(error)
            )
            return None
    # ================================================================
    # START
    # ================================================================
    def start_simulation(
        self,
        button
    ):
        parameters = (
            self.get_parameters()
        )
        if parameters is None:
```

```
            return

        (

            event_count,

            checkpoint_interval,

            sync_loss_probability,

            timeout_probability,

            recovery_probability,

            data_size

        ) = parameters

        # ------------------------------------------------------------
        # CREATE SIMULATOR
        # ------------------------------------------------------------

        self.simulator = (

            SessionLayerSimulator(

                event_count=
                    event_count,

                checkpoint_interval=
                    checkpoint_interval,

                synchronization_probability=
                    sync_loss_probability,

                timeout_probability=
                    timeout_probability,

                recovery_probability=
                    recovery_probability,

                data_size=
                    data_size

            )

```

```
        )

        # ------------------------------------------------------------
        # RESET GUI
        # ------------------------------------------------------------

        self.event_store.clear()

        self.figure.clear()

        self.canvas.draw()

        self.progress.set_fraction(
            0.0
        )

        self.progress.set_text(
            "0%"
        )

        self.start_button.set_sensitive(
            False
        )

        self.status.set_text(

            "Session simulation running..."

        )

        # ------------------------------------------------------------
        # WORKER
        # ------------------------------------------------------------

        self.worker_thread = (

            threading.Thread(

                target=
                    self.run_worker,

                daemon=True

            )

        )
```

```
2616
2617            self.worker_thread.start()
2618
2619        # ================================================================
2620        # WORKER
2621        # ================================================================
2622
2623        def run_worker(self):
2624
2625            self.simulator.run(
2626
2627                progress_callback=
2628                    self.progress_callback,
2629
2630                event_callback=
2631                    self.event_callback
2632
2633            )
2634
2635            GLib.idle_add(
2636
2637                self.simulation_finished
2638
2639            )
2640
2641        # ================================================================
2642        # EVENT CALLBACK
2643        # ================================================================
2644
2645        def event_callback(
2646            self,
2647            event
2648        ):
2649
2650            GLib.idle_add(
2651
2652                self.add_event_to_table,
2653
2654                event
2655
2656            )
2657
2658            GLib.idle_add(
2659
2660                self.update_live_view
2661
```

```
        )

    # ============================================================
    # ADD EVENT
    # ============================================================

    def add_event_to_table(
        self,
        event
    ):

        self.event_store.append(

            [

                event.event_id,

                event.event_type,

                event.state,

                event.sequence_number,

                event.checkpoint,

                round(
                    event.response_time_ms,
                    2
                ),

                event.status,

                event.data_bytes

            ]

        )

        return False

    # ============================================================
    # PROGRESS CALLBACK
    # ============================================================

    def progress_callback(
        self,
```

```
        current,
        total
    ):

        GLib.idle_add(

            self.update_progress,

            current,

            total

        )

    # ================================================================
    # UPDATE PROGRESS
    # ================================================================

    def update_progress(
        self,
        current,
        total
    ):

        fraction = (

            current
            /
            total

        )

        self.progress.set_fraction(
            fraction
        )

        self.progress.set_text(

            f"{current}/{total} "
            f"({fraction * 100:.1f}%)"

        )

        self.status.set_text(

            f"Processing session event "
```

```
            f"{current}/{total}"

        )

        self.update_live_view()

        return False

    # ================================================================
    # UPDATE LIVE VIEW
    # ================================================================

    def update_live_view(self):

        dataframe = (
            self.simulator.dataframe()
        )

        if dataframe.empty:

            self.update_metrics()

            return False

        self.update_metrics()

        self.update_graph()

        return False

    # ================================================================
    # UPDATE METRICS
    # ================================================================

    def update_metrics(self):

        metrics = (
            self.simulator.get_metrics()
        )

        self.metric_labels[
            "generated_events"
        ].set_markup(

            f"<b>{metrics['generated_events']}</b>"

```

```
        )

        self.metric_labels[
            "processed_events"
        ].set_markup(

            f"<b>{metrics['processed_events']}</b>"

        )

        self.metric_labels[
            "successful_events"
        ].set_markup(

            f"<b>{metrics['successful_events']}</b>"

        )

        self.metric_labels[
            "failed_events"
        ].set_markup(

            f"<b>{metrics['failed_events']}</b>"

        )

        self.metric_labels[
            "synchronization_losses"
        ].set_markup(

            f"<b>{metrics['synchronization_losses']}</b>"

        )

        self.metric_labels[
            "recovery_attempts"
        ].set_markup(

            f"<b>{metrics['recovery_attempts']}</b>"

        )

        self.metric_labels[
            "successful_recoveries"
        ].set_markup(

```

```
            f"<b>{metrics['successful_recoveries']}</b>"

        )

        self.metric_labels[
            "checkpoint_count"
        ].set_markup(

            f"<b>{metrics['checkpoint_count']}</b>"

        )

        self.metric_labels[
            "keepalive_count"
        ].set_markup(

            f"<b>{metrics['keepalive_count']}</b>"

        )

        self.metric_labels[
            "timeout_count"
        ].set_markup(

            f"<b>{metrics['timeout_count']}</b>"

        )

        self.metric_labels[
            "reliability"
        ].set_markup(

            f"<b>{metrics['reliability']:.2f}%</b>"

        )

        self.metric_labels[
            "recovery_rate"
        ].set_markup(

            f"<b>{metrics['recovery_rate']:.2f}%</b>"

        )

    # ============================================================
    # UPDATE GRAPH
```

```
    # ================================================================

    def update_graph(self):

        dataframe = (
            self.simulator.dataframe()
        )

        if dataframe.empty:

            return

        # ------------------------------------------------------------
        # CLEAR FIGURE
        # ------------------------------------------------------------

        self.figure.clear()

        x = (
            dataframe["event_id"]
        )

        # ============================================================
        # 1. RESPONSE TIME
        # ============================================================

        ax1 = self.figure.add_subplot(
            5,
            2,
            1
        )

        ax1.plot(

            x,

            dataframe["response_time_ms"],

            linewidth=1.5

        )

        ax1.set_title(

            "Session Response Time",

```

```
            fontsize=11,

            fontweight="bold"

        )

        ax1.set_xlabel(
            "Event"
        )

        ax1.set_ylabel(
            "Response Time (ms)"
        )

        ax1.grid(
            True,
            alpha=0.3
        )

        # ============================================================
        # 2. SEQUENCE NUMBER
        # ============================================================

        ax2 = self.figure.add_subplot(
            5,
            2,
            2
        )

        ax2.plot(

            x,

            dataframe["sequence_number"],

            linewidth=2

        )

        ax2.set_title(

            "Session Sequence Progression",

            fontsize=11,

            fontweight="bold"
```

```
        )

        ax2.set_xlabel(
            "Event"
        )

        ax2.set_ylabel(
            "Sequence"
        )

        ax2.grid(
            True,
            alpha=0.3
        )

        # ============================================================
        # 3. CHECKPOINT
        # ============================================================

        ax3 = self.figure.add_subplot(
            5,
            2,
            3
        )

        ax3.plot(

            x,

            dataframe["checkpoint"],

            linewidth=2

        )

        ax3.set_title(

            "Session Synchronization Checkpoint",

            fontsize=11,

            fontweight="bold"

        )

```

```
        ax3.set_xlabel(
            "Event"
        )

        ax3.set_ylabel(
            "Checkpoint"
        )

        ax3.grid(
            True,
            alpha=0.3
        )

        # ============================================================
        # 4. CHECKPOINT COUNT
        # ============================================================

        ax4 = self.figure.add_subplot(
            5,
            2,
            4
        )

        ax4.plot(

            x,

            dataframe["checkpoint_count"],

            linewidth=2

        )

        ax4.set_title(

            "Cumulative Checkpoints",

            fontsize=11,

            fontweight="bold"

        )

        ax4.set_xlabel(
            "Event"
        )
```

```
        ax4.set_ylabel(
            "Checkpoint Count"
        )

        ax4.grid(
            True,
            alpha=0.3
        )

        # ============================================================
        # 5. SESSION RELIABILITY
        # ============================================================

        ax5 = self.figure.add_subplot(
            5,
            2,
            5
        )

        ax5.plot(

            x,

            dataframe["reliability"],

            linewidth=2

        )

        ax5.set_title(

            "Session Reliability",

            fontsize=11,

            fontweight="bold"

        )

        ax5.set_xlabel(
            "Event"
        )

        ax5.set_ylabel(
            "Reliability (%)"
```

```
        )

        ax5.set_ylim(
            0,
            105
        )

        ax5.grid(
            True,
            alpha=0.3
        )

        # ============================================================
        # 6. RECOVERY RATE
        # ============================================================

        ax6 = self.figure.add_subplot(
            5,
            2,
            6
        )

        ax6.plot(

            x,

            dataframe["recovery_rate"],

            linewidth=2

        )

        ax6.set_title(

            "Session Recovery Rate",

            fontsize=11,

            fontweight="bold"

        )

        ax6.set_xlabel(
            "Event"
        )

```

```
        ax6.set_ylabel(
            "Recovery Rate (%)"
        )

        ax6.set_ylim(
            0,
            105
        )

        ax6.grid(
            True,
            alpha=0.3
        )

        # ============================================================
        # 7. SYNCHRONIZATION LOSS
        # ============================================================

        ax7 = self.figure.add_subplot(
            5,
            2,
            7
        )

        ax7.plot(

            x,

            dataframe[
                "synchronization_losses"
            ],

            linewidth=2

        )

        ax7.set_title(

            "Cumulative Synchronization Loss",

            fontsize=11,

            fontweight="bold"

        )

```

```
        ax7.set_xlabel(
            "Event"
        )

        ax7.set_ylabel(
            "Loss Count"
        )

        ax7.grid(
            True,
            alpha=0.3
        )

        # ============================================================
        # 8. RECOVERY ATTEMPTS
        # ============================================================

        ax8 = self.figure.add_subplot(
            5,
            2,
            8
        )

        ax8.plot(

            x,

            dataframe[
                "recovery_attempts"
            ],

            linewidth=2,

            label="Attempts"

        )

        ax8.plot(

            x,

            dataframe[
                "successful_recoveries"
            ],

            linewidth=2,
```

```
            label="Successful"

        )

        ax8.set_title(

            "Session Recovery",

            fontsize=11,

            fontweight="bold"

        )

        ax8.set_xlabel(
            "Event"
        )

        ax8.set_ylabel(
            "Count"
        )

        ax8.legend(
            fontsize=8
        )

        ax8.grid(
            True,
            alpha=0.3
        )

        # ============================================================
        # 9. KEEP-ALIVE / TIMEOUT
        # ============================================================

        ax9 = self.figure.add_subplot(
            5,
            2,
            9
        )

        ax9.plot(

            x,
```

```
            dataframe[
                "keepalive_count"
            ],

            linewidth=2,

            label="Keepalive"

        )

        ax9.plot(

            x,

            dataframe[
                "timeout_count"
            ],

            linewidth=2,

            label="Timeout"

        )

        ax9.set_title(

            "Keep-Alive and Timeout",

            fontsize=11,

            fontweight="bold"

        )

        ax9.set_xlabel(
            "Event"
        )

        ax9.set_ylabel(
            "Count"
        )

        ax9.legend(
            fontsize=8
        )

```

```
        ax9.grid(
            True,
            alpha=0.3
        )

        # ============================================================
        # 10. DATA EXCHANGE
        # ============================================================

        ax10 = self.figure.add_subplot(
            5,
            2,
            10
        )

        data_kb = (

            dataframe[
                "total_data_bytes"
            ].to_numpy()

            /

            1024.0

        )

        ax10.plot(

            x,

            data_kb,

            linewidth=2

        )

        ax10.set_title(

            "Cumulative Session Data",

            fontsize=11,

            fontweight="bold"

        )
```

```
        ax10.set_xlabel(
            "Event"
        )

        ax10.set_ylabel(
            "Data (KB)"
        )

        ax10.grid(
            True,
            alpha=0.3
        )

        # ============================================================
        # GLOBAL TITLE
        # ============================================================

        self.figure.suptitle(

            "OSI Session Layer Simulation",

            fontsize=15,

            fontweight="bold"

        )

        # ============================================================
        # LAYOUT
        # ============================================================

        self.figure.tight_layout(

            rect=[

                0.03,
                0.02,
                0.98,
                0.96

            ],

            h_pad=2.5,

            w_pad=2.0
```

```
        )

        self.canvas.draw_idle()

    # ================================================================
    # SIMULATION FINISHED
    # ================================================================

    def simulation_finished(self):

        self.start_button.set_sensitive(
            True
        )

        self.update_metrics()

        self.update_graph()

        metrics = (
            self.simulator.get_metrics()
        )

        self.status.set_text(

            "Session simulation completed | "

            f"Session={metrics['session_id']} | "

            f"Events={metrics['generated_events']} | "

            f"Reliability="
            f"{metrics['reliability']:.2f}% | "

            f"Recovery="
            f"{metrics['recovery_rate']:.2f}% | "

            f"Checkpoints="
            f"{metrics['checkpoint_count']}"

        )

        return False

    # ================================================================
    # STOP
```

```
    # ================================================================

    def stop_simulation(
        self,
        button
    ):

        self.simulator.stop()

        self.start_button.set_sensitive(
            True
        )

        self.status.set_text(

            "Session simulation stopped."

        )

    # ================================================================
    # EXPORT CSV
    # ================================================================

    def export_csv(
        self,
        button
    ):

        dataframe = (
            self.simulator.dataframe()
        )

        if dataframe.empty:

            self.error_dialog(

                "No simulation data available."

            )

            return

        dialog = Gtk.FileChooserDialog(

            title=
                "Export Session Layer CSV",
```

```
            parent=
                self,

            action=
                Gtk.FileChooserAction.SAVE

        )

        dialog.add_buttons(

            Gtk.STOCK_CANCEL,

            Gtk.ResponseType.CANCEL,

            Gtk.STOCK_SAVE,

            Gtk.ResponseType.OK

        )

        dialog.set_current_name(

            "session_layer_results.csv"

        )

        response = dialog.run()

        if response == Gtk.ResponseType.OK:

            filename = (
                dialog.get_filename()
            )

            try:

                dataframe.to_csv(

                    filename,

                    index=False

                )

                self.status.set_text(
```

```
                "CSV exported: "
                +
                filename

            )

        except Exception as error:

            self.error_dialog(
                str(error)
            )

    dialog.destroy()

# ================================================================
# ERROR DIALOG
# ================================================================

def error_dialog(
    self,
    message
):

    dialog = Gtk.MessageDialog(

        parent=self,

        flags=
            Gtk.DialogFlags.MODAL,

        message_type=
            Gtk.MessageType.ERROR,

        buttons=
            Gtk.ButtonsType.OK,

        text=
            "Simulation Error"

    )

    dialog.format_secondary_text(
        message
    )

```

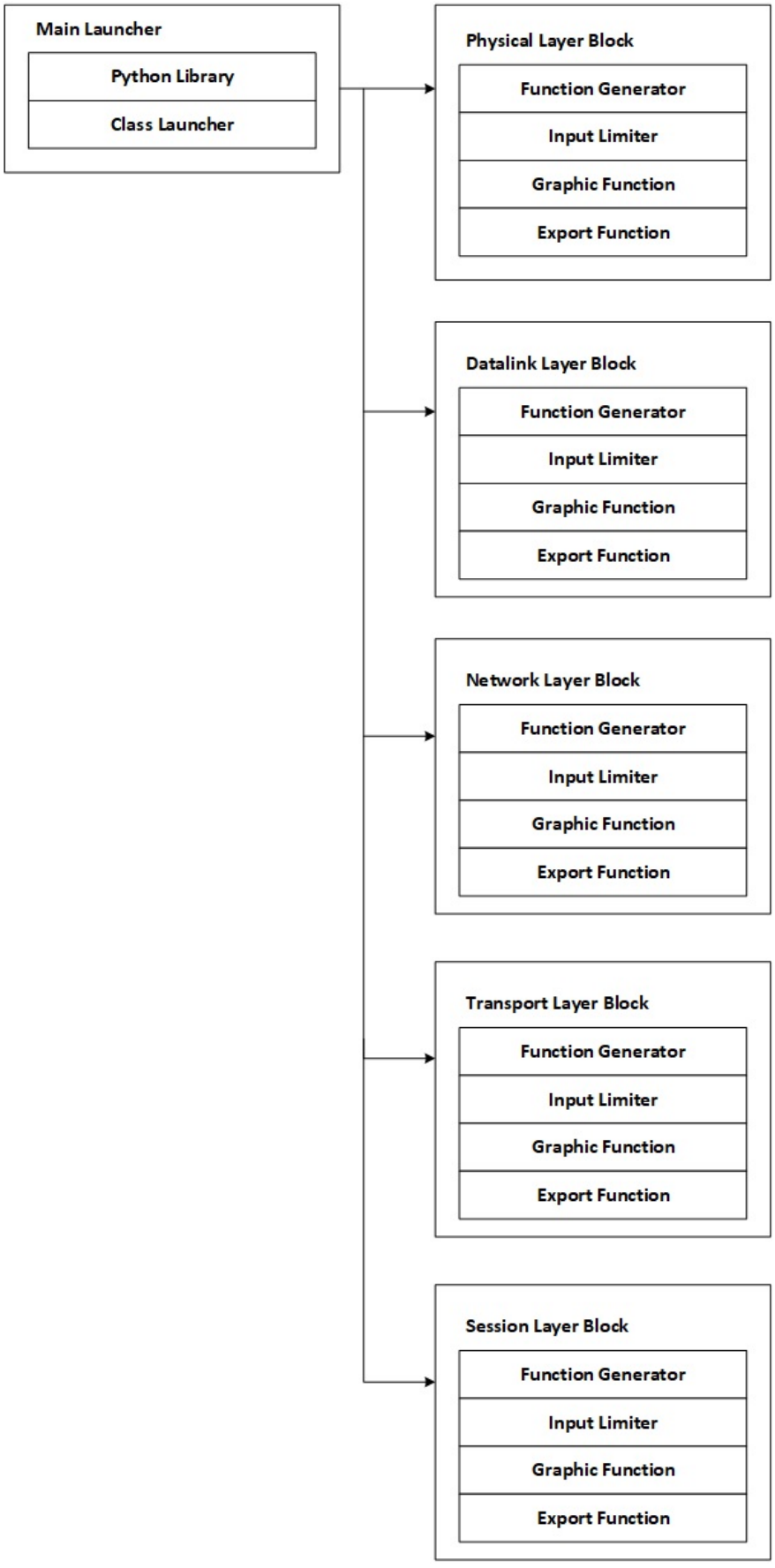
```
        dialog.run()

        dialog.destroy()


# ======================================================================
# MAIN
# ======================================================================

def main():

    window = (
        SessionLayerGTK()
    )

    window.connect(

        "destroy",

        Gtk.main_quit

    )

    window.show_all()

    # ================================================================
    # MAXIMIZE FOR BEST SCREEN FIT
    # ================================================================

    window.maximize()

    Gtk.main()


# ======================================================================
# ENTRY POINT
# ======================================================================

if __name__ == "__main__":

    main()
```

Aplikasi ini adalah simulator *Session Layer* yang menampilkan hasil simulasi dalam antar muka grafis pustaka GTK dan grafik dengan pustaka Matplotlib. Arsitektur aplikasi simulator *Session Layer* dapat kita lihat pada gambar 2.17: Pembaca dapat melihat penjelasan kode sumber pada keterangan simulator Session Layer.

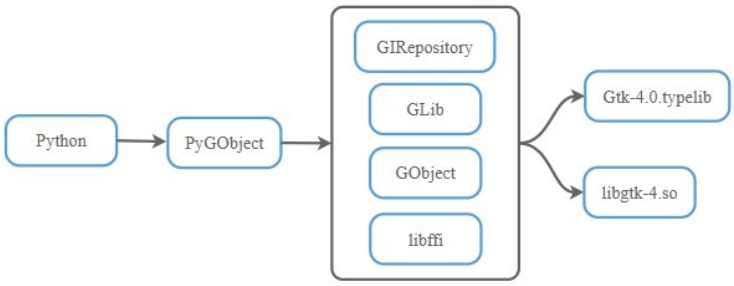
2.7.1 Tampilan Simulator Session Layer

Tampilan simulator Session Layer dapat pembaca lihat pada Gambar 2.18:

Tampilan dari Session Event Log pada simulator Session Layer dapat pembaca lihat pada Gambar 2.19:

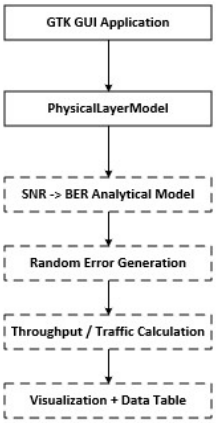


Gambar 2.1: Arsitektur SimPyNetLib v2.0

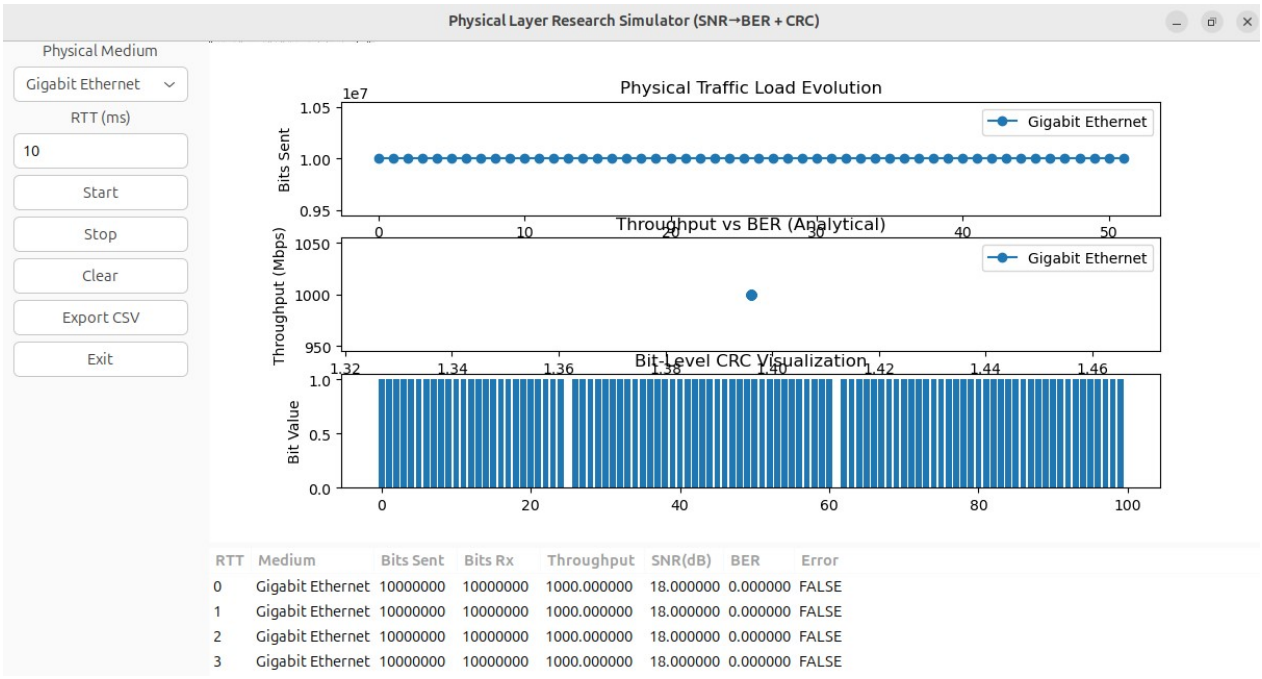


Gambar 2.2: Cara Kerja pustaka PyGObject [5]

Gambar 2.3: Tampilan aplikasi Main Launcher

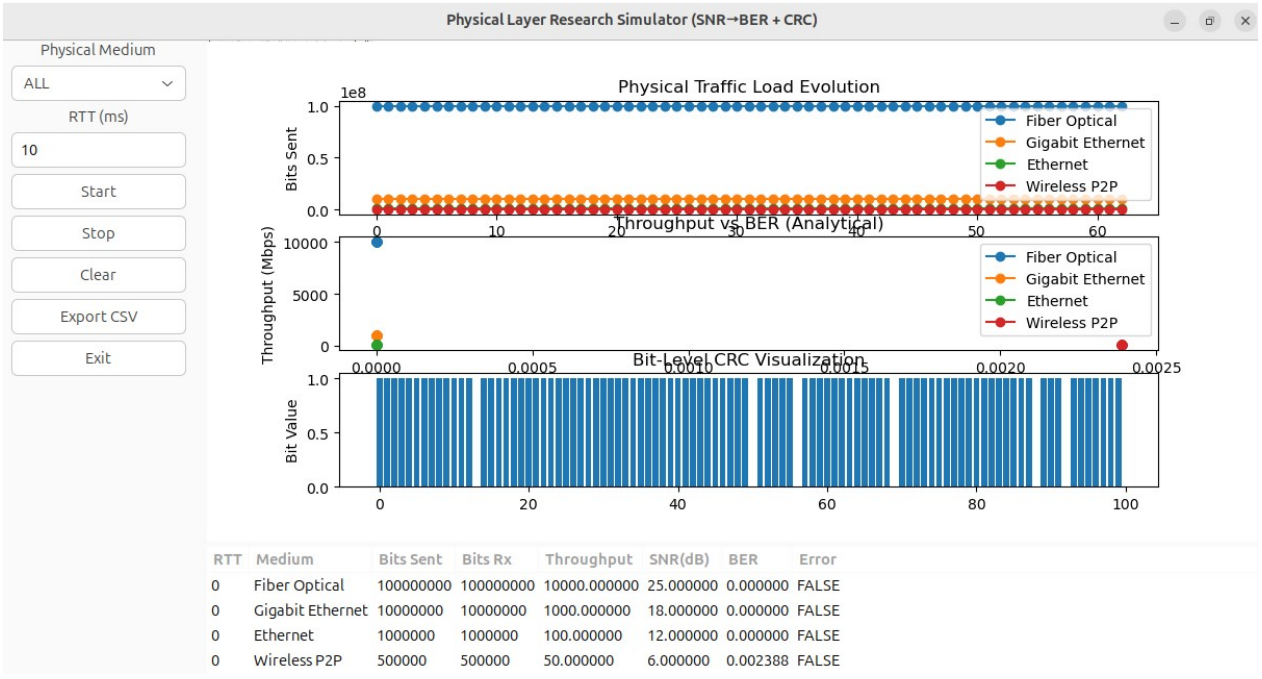


Gambar 2.4: Skema aplikasi Physical Layer Simulator



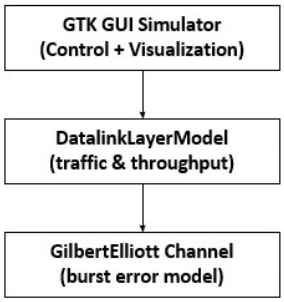
RTT	Medium	Bits Sent	Bits Rx	Throughput	SNR(dB)	BER	Error
0	Gigabit Ethernet	10000000	10000000	1000.000000	18.000000	0.000000	FALSE
1	Gigabit Ethernet	10000000	10000000	1000.000000	18.000000	0.000000	FALSE
2	Gigabit Ethernet	10000000	10000000	1000.000000	18.000000	0.000000	FALSE
3	Gigabit Ethernet	10000000	10000000	1000.000000	18.000000	0.000000	FALSE

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

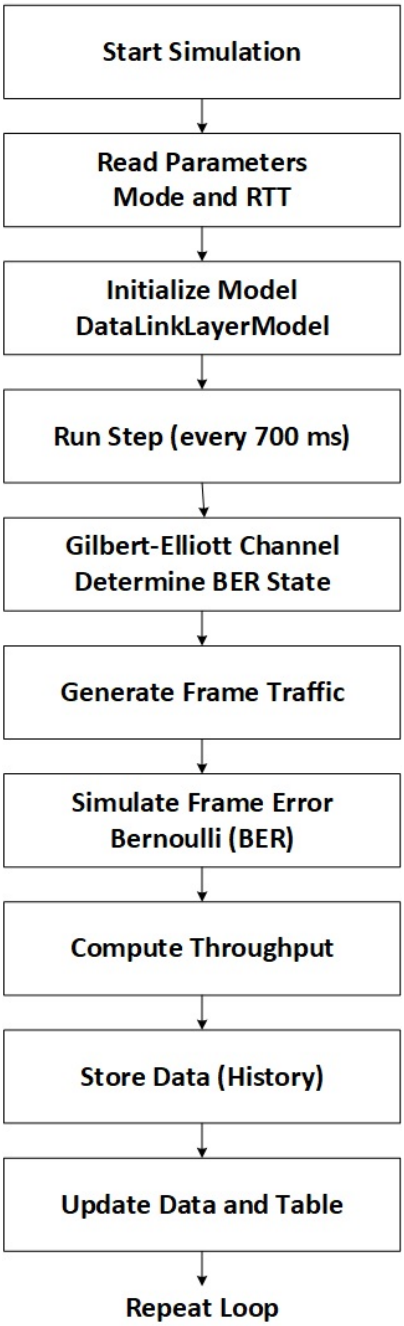


RTT	Medium	Bits Sent	Bits Rx	Throughput	SNR(dB)	BER	Error
0	Fiber Optical	100000000	100000000	10000.000000	25.000000	0.000000	FALSE
0	Gigabit Ethernet	10000000	10000000	1000.000000	18.000000	0.000000	FALSE
0	Ethernet	1000000	1000000	100.000000	12.000000	0.000000	FALSE
0	Wireless P2P	500000	500000	50.000000	6.000000	0.002388	FALSE

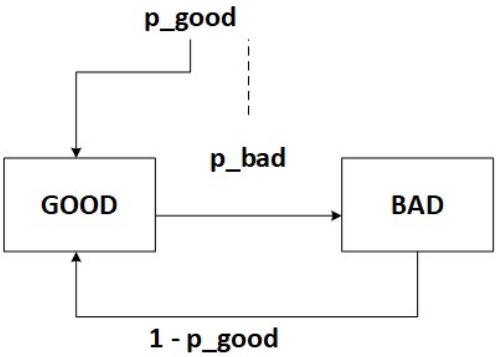
Gambar 2.6: Tampilan aplikasi Physical Layer Simulator untuk semua medium



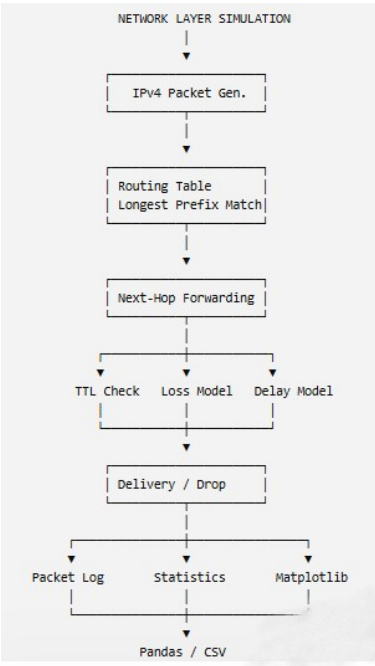
Gambar 2.7: Skema Datalink Layer Simulator



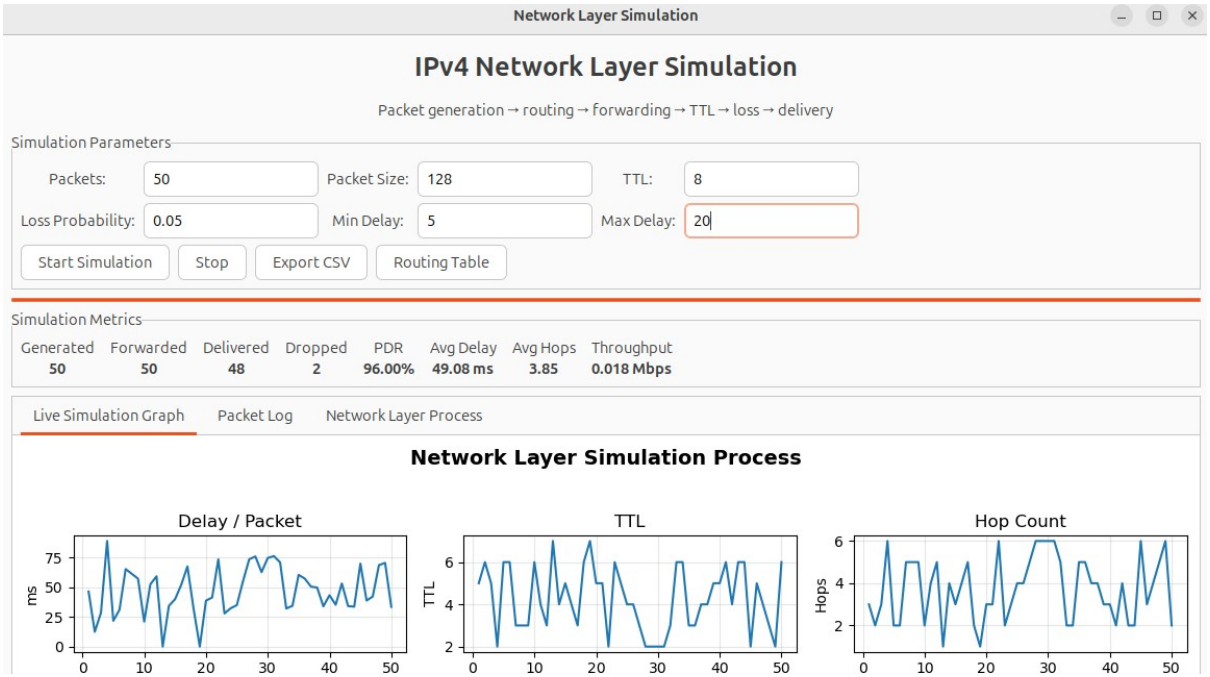
Gambar 2.8: Alur Kerja Datalink Layer Simulator



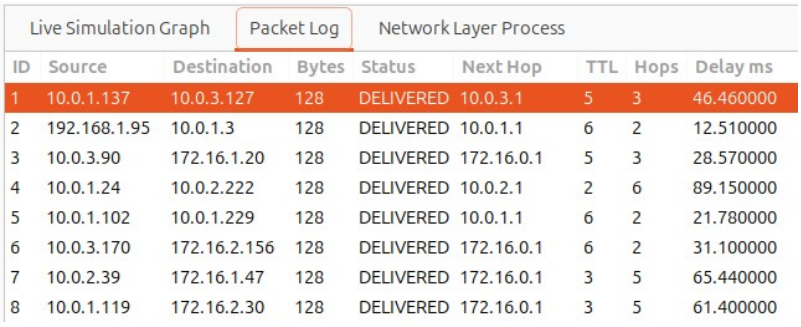
Gambar 2.9: State Diagram pada Datalink Layer Simulator



Gambar 2.10: Skema Network Layer Simulator

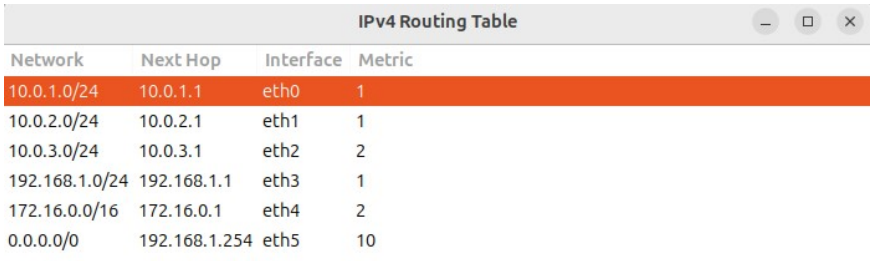


Gambar 2.11: Tampilan Network Layer Simulator



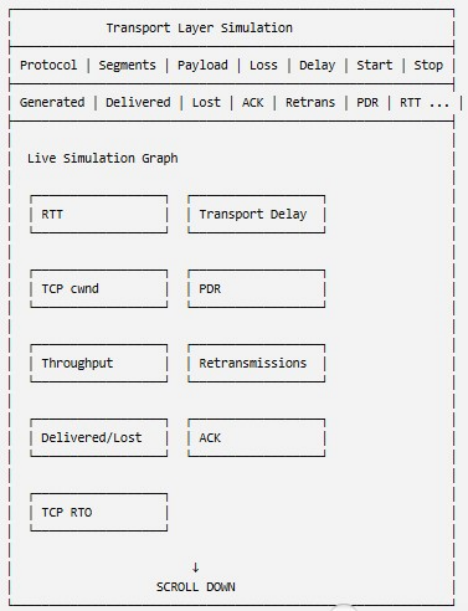
ID	Source	Destination	Bytes	Status	Next Hop	TTL	Hops	Delay ms
1	10.0.1.137	10.0.3.127	128	DELIVERED	10.0.3.1	5	3	46.460000
2	192.168.1.95	10.0.1.3	128	DELIVERED	10.0.1.1	6	2	12.510000
3	10.0.3.90	172.16.1.20	128	DELIVERED	172.16.0.1	5	3	28.570000
4	10.0.1.24	10.0.2.222	128	DELIVERED	10.0.2.1	2	6	89.150000
5	10.0.1.102	10.0.1.229	128	DELIVERED	10.0.1.1	6	2	21.780000
6	10.0.3.170	172.16.2.156	128	DELIVERED	172.16.0.1	6	2	31.100000
7	10.0.2.39	172.16.1.47	128	DELIVERED	172.16.0.1	3	5	65.440000
8	10.0.1.119	172.16.2.30	128	DELIVERED	172.16.0.1	3	5	61.400000

Gambar 2.12: Tampilan Packet Log pada Network Layer Simulator

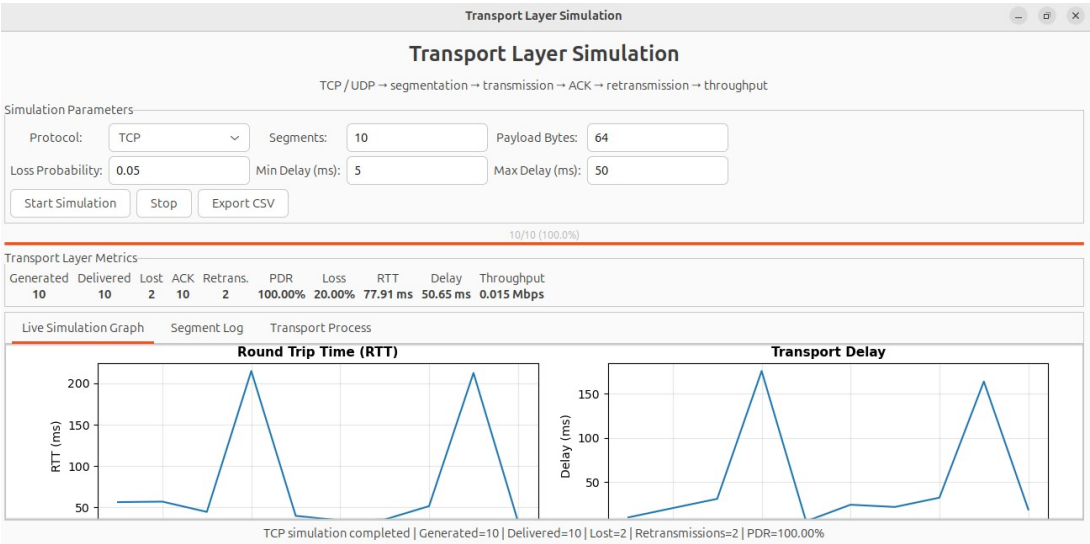


Network	Next Hop	Interface	Metric
10.0.1.0/24	10.0.1.1	eth0	1
10.0.2.0/24	10.0.2.1	eth1	1
10.0.3.0/24	10.0.3.1	eth2	2
192.168.1.0/24	192.168.1.1	eth3	1
172.16.0.0/16	172.16.0.1	eth4	2
0.0.0.0/0	192.168.1.254	eth5	10

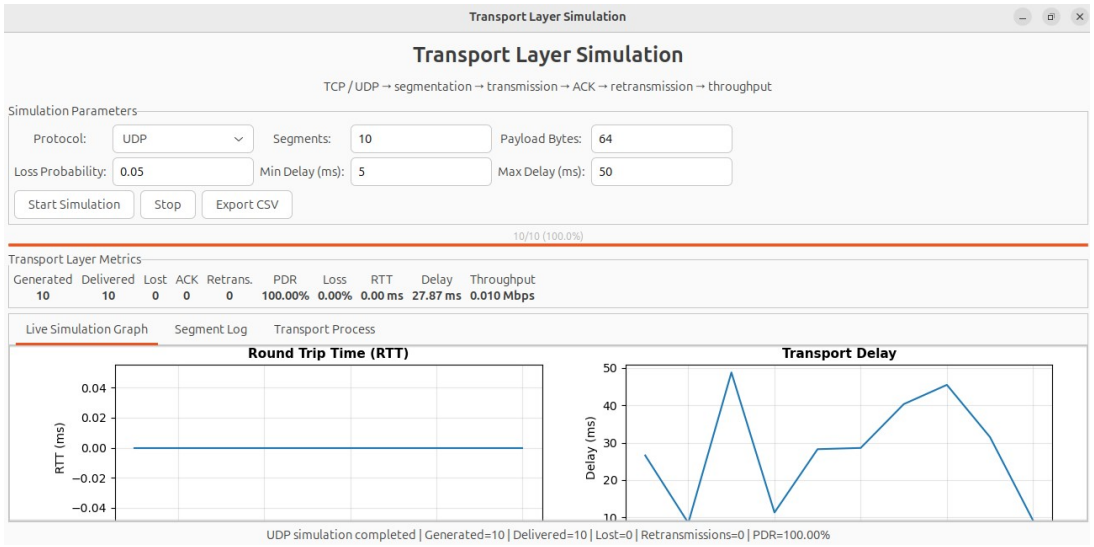
Gambar 2.13: Tampilan Routing Table pada Network Layer Simulator



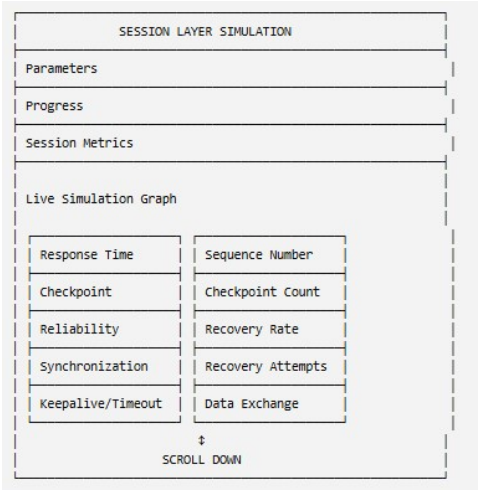
Gambar 2.14: Skema Transport Layer Simulator



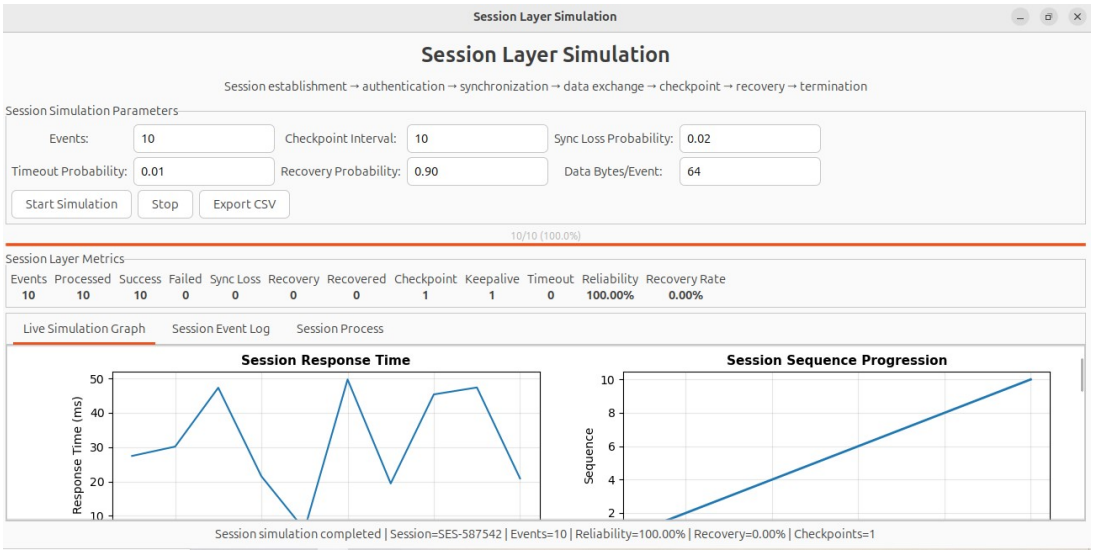
Gambar 2.15: Tampilan Opsi Protokol TCP pada Transport Layer Simulator



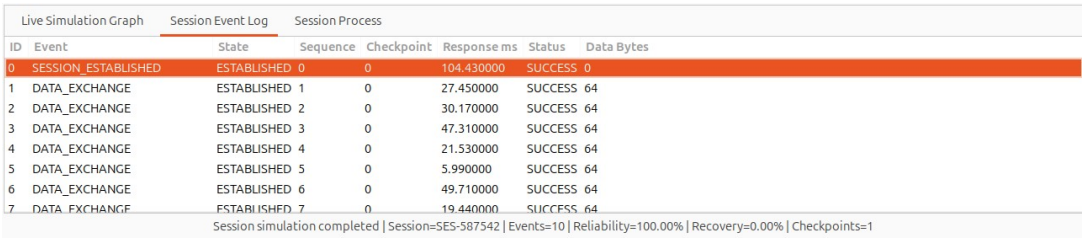
Gambar 2.16: Tampilan Opsi Protokol UDP pada Transport Layer Simulator



Gambar 2.17: Skema Session Layer Simulator



Gambar 2.18: Tampilan Session Layer Simulator



Gambar 2.19: Tampilan Session Event Log pada Sessin Layer Simulator

Bab 3

Kesimpulan

Pengembangan simulator SimPyNetLib v2.0 telah dapat mendeskripsikan simulasi lapisan jaringan komputer dalam hal ini *Physical Layer, Datalink Layer, Network Layer, Transport Layer* dan *Session 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 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 4

[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. 4

[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. 4

[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. 4

[5] Unknown, "Overview - pgobject," accessed December 19th, 2025. [Online]. Available: https://pygobject.gnome.org/index.html 7, 194

[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 7

[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 7

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

[9] ——, "Sys - python standard library," accessed December 20th, 2025. [Online]. Available: https://realpython.com/ref/stdlib/sys/ 8

[10] ——, "os - python standard library," accessed December 20th, 2025. [Online]. Available: https://realpython.com/ref/stdlib/os/ 8

[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

[13] O. Nassef, T. Mahmoodi, F. Chiariotti, and S. H. Johnson, "Drel: Dynamically assigning per-packet reliability at the transport layer," *IEEE Transactions on Network and Service Management*, vol. 22, no. 2, pp. 1276–1294, 2025. 64

[14] C. H. Koo and S. C. Burleigh, "Aggressive and proactive ltp control signal handling for minimal session delivery time: Rtt rules the world," *Journal of Communications and Networks*, vol. 25, no. 4, pp. 516–531, 2023. 112