PENGEMBANGAN PROGRAM KULIAH MODEL LAPISAN TRANSPORT DAN SESSION JARINGAN KOMPUTER BERBASIS PUSTAKA GRAFIS SEBAGAI MEDIA PERKULIAHAN LAN AND WIRELESS PADA PROGRAM STUDI INFORMATIKA

Berkah I. Santoso, ST, MTI (0309068003)

Program Studi Informatika
Fakultas Teknik dan Ilmu Komputer
Universitas Bakrie
Jakarta

2026

LEMBAR PENGESAHAN

1.	Judul Program	: Pengembangan Program Kuliah Model Lapisan Transport dan Session Jaringan Komputer Berbasis Pustaka Grafis Sebagai Media Perkuliahan LAN and Wireless Pada Program Studi Informatika
2.	Dosen Penyusun	
	a. Nama Lengkap	: Berkah Iman Santoso, S.T, MTI
	b. Jenis Kelamin	: Laki Laki
	c. NIDN	: 0309068003
	d. Pangkat/Golongan	: Asisten Ahli / III b
	e. Jabatan	: Dosen
	f. No. telepon seluler/Alamat Surel	: 08128114857 / berkah.santoso@bakrie.ac.id
3.	Anggota Tim Pengusul Kegiatan	
	a. Dosen	: -
	b. Praktisi	: -
4.	Peserta	
	a. Mahasiswa	: -
	b. Alumni	: -
5.	Biaya Kegiatan	
	a. Universitas Bakrie	: -
	b. Sumber lain	: -
6.	Tahun Pelaksanaan	: 2026

Jakarta, 16 Juli 2026

Mengetahui, Ketua Program Studi Informatika	Dosen Pengusul
	
Ir. Iwan Adhicandra, S.T, M.Sc, Ph.D., SMIEEE NIP: 0018025806	Berkah Iman Santoso, S.T, MTI NIDN : 0309068003

Daftar Isi

Bagian 1

Kata Pengantar

Atas berkat rahmat Allah Subhanahu Wa Ta'ala, selesailah Pengembangan Program Kuliah Model Lapisan Transport dan Session Jaringan Komputer Berbasis Pustaka Grafis Sebagai Media Perkuliahan LAN and Wireless pada Kelas Karyawan, Program Studi Informatika, Fakultas Teknik dan Ilmu Komputer. Oleh karena itu, penulis sebagai pengembang program kuliah model lapisan transport dan session jaringan komputer berbasis pustaka grafis untuk media perkuliahan LAN and Wireless Kelas Karyawan, memberikan penghargaan yang setinggi-tingginya dan menyampaikan ucapan terima kasih yang sebesar-besarnya atas segala bantuannya terutama kepada yang terhormat :

1. Rektor Universitas Bakrie.
2. Ketua Program Studi Informatika, FTIK, Universitas Bakrie.
3. Kepala Perpustakaan Universitas Bakrie.
4. Kolega Dosen pada Program Studi Informatika, FTIK.

Semoga kegiatan ini dapat berkembang terus sejalan dengan perkembangan Program Studi Informatika Universitas Bakrie.

Jakarta, Agustus 2026
Pelaksana Pengembangan Program Kuliah Prodi Informatika

Bagian 2

Latar Belakang

Maksud dan tujuan diselenggarakannya Pengembangan Program Kuliah Model Lapisan Transport dan Session Jaringan Komputer Berbasis Pustaka Grafis sebagai Media Perkuliahan LAN and Wireless pada Kelas Karyawan, Program Studi Informatika adalah dalam rangka memberikan alternatif perangkat bantu bagi mahasiswa kelas karyawan yang bekerja pada jam kerja, untuk tetap mempelajari materi OSI 7 Layer jaringan komputer tanpa harus tergantung pada *software* simulator seperti OpNet®yang saat ini telah berlisensi *proprietary* dan berbayar setelah diakuisisi oleh Riverbed Technology®menjadi Riverbed Modeler®. Selain itu simulator untuk model lapisan transport dan session ini memberikan terobosan penggunaan teknologi terkini untuk meningkatkan kemampuan dosen selaku salah satu mediator pembelajaran sebagai perluasan jangkauan kanal akses perkuliahan kepada mahasiswa kelas karyawan. Kegiatan Pengembangan Program Kuliah ini sejalan dengan upaya Universitas Bakrie dalam menerapkan *experience-based learning* untuk dapat terus memberikan layanan pembelajaran dalam rangka memberikan *update* teknologi terkini yang dapat dijangkau oleh mahasiswa kelas reguler dan karyawan. Penggunaan aplikasi Model Lapisan Transport dan Session Jaringan Komputer berbasis Pustaka Grafis sebagai Media Perkuliahan LAN and Wireless dirasakan tepat guna untuk Kelas Karyawan. Hal tersebut merupakan upaya penulis agar mahasiswa peserta kelas LAN and Wireless dapat mempelajari simulasi perilaku *traffic* jaringan komputer pada setiap lapisannya tanpa harus memiliki ketergantungan pada *software prioprietary* simulator berbayar. Berdasarkan kondisi tersebut maka gagasan untuk melaksanakan Pengembangan Program Kuliah melalui pengembangan aplikasi Model Lapisan Transport dan Session Jaringan Komputer Berbasis Pustaka Grafis sebagai Media Perkuliahan LAN and Wireless pada Kelas Karyawan, Program Studi Informatika ini kami susun. Aplikasi simulasi jaringan komputer yang mendeskripsikan perilaku *traffic* pada masing- masing lapisan beserta dengan protokol yang digunakan sangat diperlukan untuk pengukuran akurasi dan efektifitas algoritma pada protokol yang akan diobservasi. Permasalahan beranjak dari lapisan transport hingga lapisan *session* yang dapat diobservasi, yaitu protokol transport, baik TCP maupun UDP [1], [2].

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* [3].

Berdasarkan beberapa materi yang bersifat *up to date* tersebut, maka penulis memulai proses pengembangan program kuliah dengan metode pendekatan keterbaruan teknologi, dalam hal ini penggunaan aplikasi Model Lapisan Transport dan Session Jaringan Komputer Berbasis Pustaka Grafis sebagai Me-

dia Perkuliahan LAN and Wireless pada Kelas Karyawan, Program Studi Informatika yang dilaksanakan pada perkuliahan pada Semester Genap 2025/2026 dengan penerapan metode kelas *online learning*.

Bagian 3

Permasalahan

Beberapa aplikasi simulator seperti NS2, OMNET++, Opnet IT Guru Academic Edition [4] [5], dirasakan masih belum dapat memenuhi kebutuhan akademik untuk pemodelan dan simulasi jaringan. Ketersediaan aplikasi simulator dalam bentuk model lapisan jaringan komputer, dimulai dari lapisan fisik hingga lapisan aplikasi yang bersifat deskriptif, *free* dan *open source* untuk pembelajaran dalam rangka lebih memahami jaringan komputer masih dirasakan kurang. Pengembangan program kuliah ini mencoba membahas penggunaan model jaringan komputer dimulai dari lapisan transport hingga lapisan *session* berbasis pustaka grafis PyGTK menggunakan bahasa pemrograman Python yang bersifat *reusable* atau *use-the-battery* dari pustaka eksisting, memiliki variasi obyek pada setiap lapisan. Hal tersebut diperlukan dalam rangka studi lebih lanjut terkait karakteristik dan perilaku *traffic* jaringan komputer pada kedua lapisan tersebut untuk memenuhi kebutuhan penyediaan model agar dapat meningkatkan efisiensi waktu desain jaringan komputer.

Berdasarkan uraian beberapa aplikasi simulator pemodelan jaringan komputer tersebut, maka penulis memandang perlu untuk memberikan usulan pengembangan aplikasi dalam rangka penyediaan media belajar yang tepat dan sesuai dengan kondisi pengguna untuk mendukung proses pembelajaran secara *online*, dengan mempertimbangkan lisensi *software* simulator, aspek teknis, pengembangan lebih lanjut dan aspek kemudahan penggunaan berikut portabilitas dengan sistem operasi oleh mahasiswa karyawan peserta kelas pembelajaran *online*.

Bagian 4

Tujuan Pengembangan Program Kuliah

Pengembangan Program Kuliah Model Lapisan Transport dan Session Jaringan Komputer ini ditujukan untuk memberikan fokus studi agar dapat tercapai tujuan pembelajaran bagi mahasiswa Kelas Karyawan yang mengikuti perkuliahan *online* dan mengembangkan diri mereka pada masa depan *open source software*. Pemanfaatan aplikasi Model Lapisan Transport dan Session Jaringan Komputer Berbasis Pustaka Grafis dirasakan cukup tepat untuk menjawab kebutuhan tersebut agar mahasiswa mendapatkan alternatif solusi untuk menyerap penyampaian materi pembelajaran dengan keterbatasan lisensi aplikasi simulator sehingga tepat guna bagi peserta kelas karyawan yang mengikuti perkuliahan *online* mata kuliah LAN and Wireless.

Bagian 5

Metode Pengembangan Program Kuliah

Pengembangan program kuliah menggunakan aplikasi Model Lapisan Transport dan Session Jaringan Komputer Berbasis Pustaka Grafis untuk Media Perkuliahan LAN and Wireless pada Kelas Karyawan, maka Program Studi Informatika menggunakan metode *rapid development and hands-on experience*, yaitu penulis menggunakan bahasa pemrogrman Python, pustaka grafis PyGTK/PyGObject, numpy, pandas, scipy, matplotlib dengan memanfaatkan pustaka Python yang telah tersedia. Berdasarkan paparan pada permasalahan, para pengguna aplikasi *network modeling and simulation* hanya memanfaatkan aplikasi yang telah tersedia baik yang bersifat *proprietary* maupun *open source* untuk melakukan studi *modeling and simulation*, untuk penggunaan dasar, tanpa melakukan eksplorasi lebih lanjut mengenai pengembangan aplikasi pemodelan dan simulasi jaringan dengan bahasa pemrograman yang tersedia. Penulis melakukan eksplorasi dalam rangka membangun simulator pemodelan jaringan komputer agar penulis dapat memanfaatkan pustaka bahasa pemrograman Python dan mengajarkan mahasiswa kelas Reguler dan Karyawan. Para mahasiswa dapat melihat sekaligus mempelajari kode sumber simulator pemodelan jaringan komputer dengan menggunakan aplikasi *text editor* yang telah terinstalasi pada notebook atau PC. Mahasiswa kelas Karyawan yang mengikuti perkuliahan *online* dapat melihat kode sumber aplikasi simulator untuk mata kuliah LAN and Wireless pada portal akademik. Sehingga diharapkan semua kanal yang mereka gunakan dapat membantu dalam mengakses kode sumber dan penjelasan materi perkuliahan LAN and Wireless.

Bagian 6

Sasaran dan Luaran

Sasaran dan Luaran dari pengembangan program kuliah penggunaan Model Lapisan Transport dan Session Jaringan Komputer Berbasis Pustaka Grafis Sebagai Media Perkuliahan LAN and Wireless pada Kelas Karyawan, Program Studi Informatika, adalah sebagai berikut :

1. Sasaran dari pengembangan program kuliah ini adalah bertambahnya pengetahuan kolega dosen pada Program Studi Informatika atas alternatif usulan dalam penggunaan bahasa pemrograman Python untuk membangun aplikasi Model Lapisan Transport dan Session Jaringan Komputer Berbasis Pustaka Grafis yang disampaikan secara *online* bagi mahasiswa kelas Karyawan dalam rangka mengikuti mata kuliah LAN and Wireless pada Semester Genap 2025/2026.

2. Luaran dari pengembangan program kuliah ini adalah tersedianya aplikasi simulator Model Lapisan Transport dan Session Jaringan Komputer Berbasis Pustaka Grafis dengan memanfaatkan penggunaan bahasa pemrograman Python.

3. Kemudahan dalam akses melihat kode sumber aplikasi simulator pemodelan lapisan jaringan komputer sebagai materi pembelajaran dirasakan memiliki manfaat yang tinggi bagi mahasiswa, dalam rangka melakukan review materi perkuliahan terutama menjelang dan pada saat Ujian Tengah Semester dan Ujian Akhir Semester.

Bagian 7

Uraian Pengembangan Program Kuliah

Uraian Pengembangan Program Kuliah Model Lapisan Transport dan Session Jaringan Komputer Berbasis Pustaka Grafis Sebagai Media Perkuliahan LAN and Wireless terbagi menjadi :

- Lapisan Transport
- Lapisan Session

7.1 Lapisan Transport

Pada aplikasi model Lapisan Transport merupakan aplikasi yang dikembangkan dengan bahasa pemrograman Python untuk mendeskripsikan simulasi pada *Transport Layer* dengan antar muka grafis dengan memanfaatkan pustaka PyGTK, numpy, pandas, scipy dan matplotlib. Komponen *Transport Layer* merupakan aplikasi yang dikembangkan dengan bahasa pemrograman Python untuk mendeskripsikan simulasi pada *Transport 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_transport_layer_research_simulation.py:

Listing 7.1: 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
```

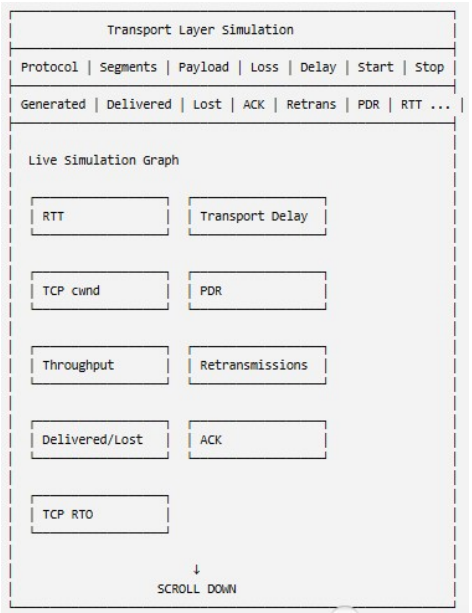
```
        # ------------------------------------------------------------
        self.metric_labels[
            "average_rtt"
        ].set_markup(
            f"<b>{metrics['average_rtt']:.2f} ms</b>"
        )
        # ------------------------------------------------------------
        # DELAY
        # ------------------------------------------------------------
        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

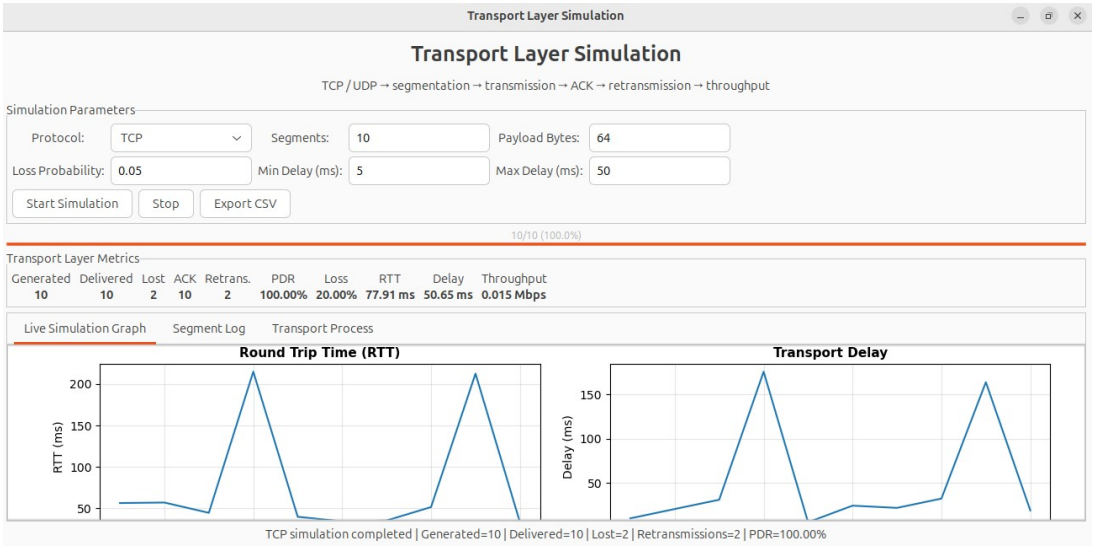


Gambar 7.1: Skema Transport Layer Simulator

Transport Layer.

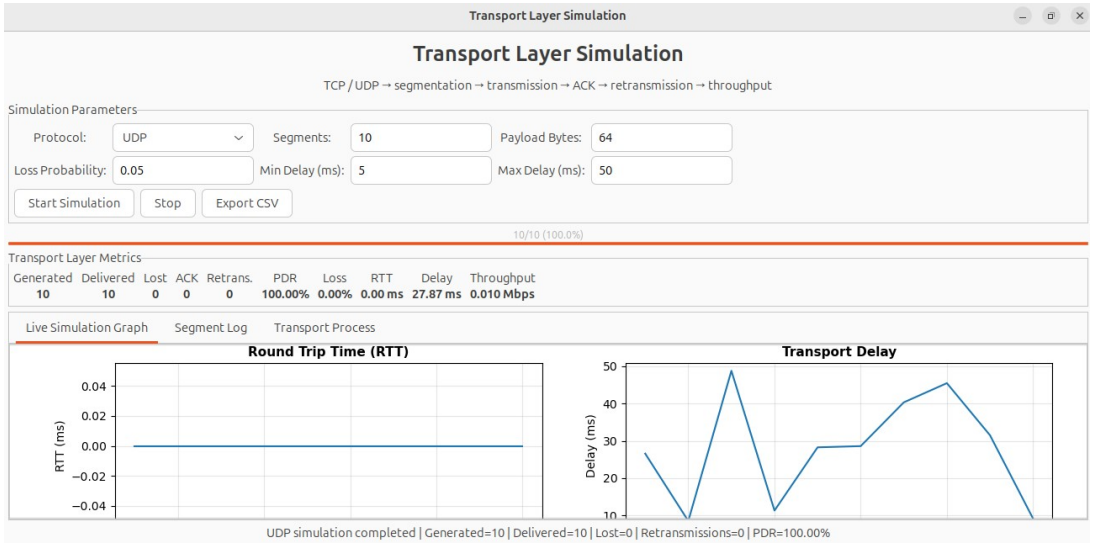
7.1.1 Tampilan Simulator Transport Layer

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



Gambar 7.2: Tampilan Opsi Protokol TCP pada Transport Layer Simulator

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



Gambar 7.3: Tampilan Opsi Protokol UDP pada Transport Layer Simulator

7.2 Lapisan Sesi

Pada aplikasi model Lapisan Sesi merupakan aplikasi yang dikembangkan dengan bahasa pemrograman Python untuk mendeskripsikan simulasi pada *Session Layer* dengan antar muka grafis dengan memanfaatkan pustaka PyGTK, numpy, pandas, scipy dan matplotlib. Komponen *Session Layer* merupakan aplikasi yang dikembangkan dengan bahasa pemrograman Python untuk mendeskripsikan simulasi pada *Session 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_session_layer_research_simulation.py:

Listing 7.2: 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

        # ------------------------------------------------------------
        # INTERNAL FLAGS
        # ------------------------------------------------------------

        self.session_established = False

        self.authenticated = False

        self.synchronized = False

    # ================================================================
    # RESET
    # ================================================================

    def reset(self):

        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

        self.events = []

        self.statistics = []

        self.running = False

        self.start_time = None

        self.end_time = None

```

```
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
```

```
                )
            )

        )

        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

        )
```

```
451
452         return self.session_establishment_time
453
454     # ================================================================
455     # CREATE CHECKPOINT
456     # ================================================================
457
458     def create_checkpoint(self):
459
460         self.last_checkpoint = (
461             self.sequence_number
462         )
463
464         self.checkpoint_count += 1
465
466         return self.last_checkpoint
467
468     # ================================================================
469     # KEEP ALIVE
470     # ================================================================
471
472     def keepalive(self):
473
474         self.keepalive_count += 1
475
476         response = (
477             self.random_response_time()
478         )
479
480         self.total_response_time += response
481
482         return response
483
484     # ================================================================
485     # SYNCHRONIZATION LOSS
486     # ================================================================
487
488     def synchronization_loss(self):
489
490         self.synchronization_losses += 1
491
492         self.synchronized = False
493
494         self.change_state(
495             SessionStateMachine.RECOVERING
496         )
```

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

```
635
636     def terminate_session(self):
637
638         start = time.time()
639
640         self.change_state(
641             SessionStateMachine.TERMINATING
642         )
643
644         response = (
645             self.random_response_time()
646         )
647
648         self.change_state(
649             SessionStateMachine.CLOSED
650         )
651
652         self.session_termination_time = (
653
654             time.time()
655             -
656             start
657
658         ) * 1000.0
659
660         self.session_termination_time += response
661
662         self.session_established = False
663
664         self.authenticated = False
665
666         self.synchronized = False
667
668         return self.session_termination_time
669
670     # ================================================================
671     # PROCESS ONE EVENT
672     # ================================================================
673
674     def process_event(
675         self,
676         event_id
677     ):
678
679         self.generated_events += 1
680
```

```
        # ------------------------------------------------------------
        # 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
        ):
```

```
911
912             if not self.running:
913
914                 break
915
916             event = (
917                 self.process_event(
918                     event_id
919                 )
920             )
921
922             # ------------------------------------------------------------
923             # RECOVERY
924             # ------------------------------------------------------------
925
926             if event.status in (
927
928                 "RECOVERY_REQUIRED",
929
930                 "TIMEOUT"
931
932             ):
933
934                 recovery_success, recovery_time = (
935                     self.recover_session()
936                 )
937
938                 recovery_status = (
939                     "RECOVERED"
940                     if recovery_success
941                     else
942                     "RECOVERY_FAILED"
943                 )
944
945                 recovery_event = SessionEvent(
946
947                     event_id,
948
949                     "SESSION_RECOVERY",
950
951                     self.state,
952
953                     self.sequence_number,
954
955                     self.last_checkpoint,
956
```

```
                    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):
```

```
1371
1372        main_box = Gtk.Box(
1373
1374            orientation=
1375                Gtk.Orientation.VERTICAL,
1376
1377            spacing=6
1378
1379        )
1380
1381        self.add(
1382            main_box
1383        )
1384
1385        # ============================================================
1386        # TITLE
1387        # ============================================================
1388
1389        title = Gtk.Label()
1390
1391        title.set_markup(
1392
1393            "<span size='xx-large' "
1394            "weight='bold'>"
1395            "Session Layer Simulation"
1396            "</span>"
1397
1398        )
1399
1400        main_box.pack_start(
1401
1402            title,
1403
1404            False,
1405            False,
1406            3
1407
1408        )
1409
1410        subtitle = Gtk.Label(
1411
1412            label=(
1413                "Session establishment     "
1414                "authentication     synchronization     "
1415                "data exchange     checkpoint     "
1416                "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"
        )
```

```
1601
1602        parameter_grid.attach(
1603
1604            self.timeout_entry,
1605
1606            1,
1607            1,
1608            1,
1609            1
1610
1611        )
1612
1613        # ------------------------------------------------------------
1614        # RECOVERY
1615        # ------------------------------------------------------------
1616
1617        parameter_grid.attach(
1618
1619            Gtk.Label(
1620                label="Recovery Probability:"
1621            ),
1622
1623            2,
1624            1,
1625            1,
1626            1
1627
1628        )
1629
1630        self.recovery_entry = (
1631            Gtk.Entry()
1632        )
1633
1634        self.recovery_entry.set_text(
1635            "0.90"
1636        )
1637
1638        parameter_grid.attach(
1639
1640            self.recovery_entry,
1641
1642            3,
1643            1,
1644            1,
1645            1
1646
```

```
        )

        # ------------------------------------------------------------
        # 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()
        )

        self.progress.set_show_text(
            True
        )

        main_box.pack_start(

            self.progress,

            False,
            False,
            0

        )

        # ============================================================
        # METRICS
        # ============================================================

        metric_frame = Gtk.Frame(

            label="Session Layer Metrics"

        )

        metric_grid = Gtk.Grid()

        metric_grid.set_border_width(
            5
        )

        metric_grid.set_column_spacing(
            9
        )

        metric_grid.set_row_spacing(
            4
        )

        metric_frame.add(
            metric_grid
        )

```

```
        main_box.pack_start(

            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
        # ============================================================

        process_box = Gtk.Box(

            orientation=
                Gtk.Orientation.VERTICAL,

            spacing=10

        )

        notebook.append_page(

            process_box,

            Gtk.Label(

                label="Session 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>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

```

```
            )

        )

        self.worker_thread.start()

    # ================================================================
    # WORKER
    # ================================================================

    def run_worker(self):

        self.simulator.run(

            progress_callback=
                self.progress_callback,

            event_callback=
                self.event_callback

        )

        GLib.idle_add(

            self.simulation_finished

        )

    # ================================================================
    # EVENT CALLBACK
    # ================================================================

    def event_callback(
        self,
        event
    ):

        GLib.idle_add(

            self.add_event_to_table,

            event

        )

        GLib.idle_add(
```

```
            self.update_live_view

        )

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

```
2705
2706    def progress_callback(
2707        self,
2708        current,
2709        total
2710    ):
2711
2712        GLib.idle_add(
2713
2714            self.update_progress,
2715
2716            current,
2717
2718            total
2719
2720        )
2721
2722    # ================================================================
2723    # UPDATE PROGRESS
2724    # ================================================================
2725
2726    def update_progress(
2727        self,
2728        current,
2729        total
2730    ):
2731
2732        fraction = (
2733
2734            current
2735            /
2736            total
2737
2738        )
2739
2740        self.progress.set_fraction(
2741            fraction
2742        )
2743
2744        self.progress.set_text(
2745
2746            f"{current}/{total} "
2747            f"({fraction * 100:.1f}%)"
2748
2749        )
2750
```

```
        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(
```

```
2797
2798            f"<b>{metrics['generated_events']}</b>"
2799
2800        )
2801
2802        self.metric_labels[
2803            "processed_events"
2804        ].set_markup(
2805
2806            f"<b>{metrics['processed_events']}</b>"
2807
2808        )
2809
2810        self.metric_labels[
2811            "successful_events"
2812        ].set_markup(
2813
2814            f"<b>{metrics['successful_events']}</b>"
2815
2816        )
2817
2818        self.metric_labels[
2819            "failed_events"
2820        ].set_markup(
2821
2822            f"<b>{metrics['failed_events']}</b>"
2823
2824        )
2825
2826        self.metric_labels[
2827            "synchronization_losses"
2828        ].set_markup(
2829
2830            f"<b>{metrics['synchronization_losses']}</b>"
2831
2832        )
2833
2834        self.metric_labels[
2835            "recovery_attempts"
2836        ].set_markup(
2837
2838            f"<b>{metrics['recovery_attempts']}</b>"
2839
2840        )
2841
2842        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>"

        )
```

```
2889
2890    # ================================================================
2891    # UPDATE GRAPH
2892    # ================================================================
2893
2894    def update_graph(self):
2895
2896        dataframe = (
2897            self.simulator.dataframe()
2898        )
2899
2900        if dataframe.empty:
2901
2902            return
2903
2904        # ------------------------------------------------------------
2905        # CLEAR FIGURE
2906        # ------------------------------------------------------------
2907
2908        self.figure.clear()
2909
2910        x = (
2911            dataframe["event_id"]
2912        )
2913
2914        # ============================================================
2915        # 1. RESPONSE TIME
2916        # ============================================================
2917
2918        ax1 = self.figure.add_subplot(
2919            5,
2920            2,
2921            1
2922        )
2923
2924        ax1.plot(
2925
2926            x,
2927
2928            dataframe["response_time_ms"],
2929
2930            linewidth=1.5
2931
2932        )
2933
2934        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
```

```
3487
3488    # ================================================================
3489    # STOP
3490    # ================================================================
3491
3492    def stop_simulation(
3493        self,
3494        button
3495    ):
3496
3497        self.simulator.stop()
3498
3499        self.start_button.set_sensitive(
3500            True
3501        )
3502
3503        self.status.set_text(
3504
3505            "Session simulation stopped."
3506
3507        )
3508
3509    # ================================================================
3510    # EXPORT CSV
3511    # ================================================================
3512
3513    def export_csv(
3514        self,
3515        button
3516    ):
3517
3518        dataframe = (
3519            self.simulator.dataframe()
3520        )
3521
3522        if dataframe.empty:
3523
3524            self.error_dialog(
3525
3526                "No simulation data available."
3527
3528            )
3529
3530            return
3531
3532        dialog = Gtk.FileChooserDialog(
```

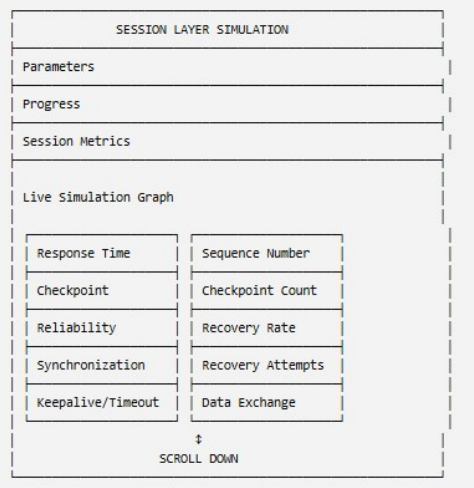
```
            title=
                "Export Session Layer CSV",

            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

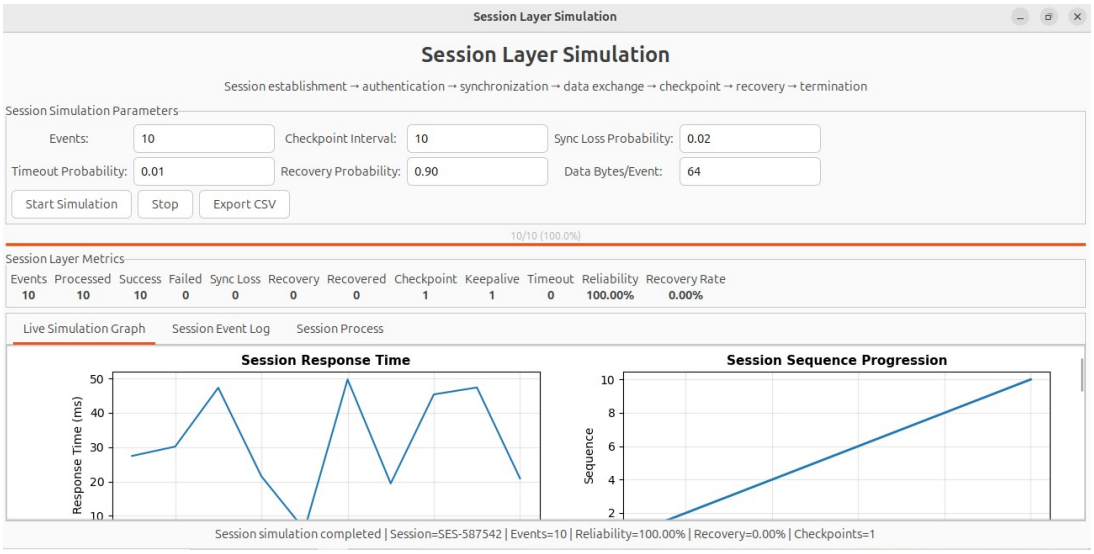


Gambar 7.4: Skema Session Layer Simulator

Session Layer.

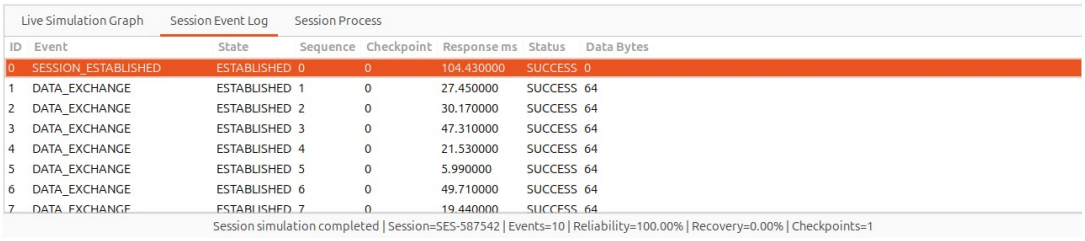
7.2.1 Tampilan Simulator Session Layer

Tampilan simulator Session Layer dapat pembaca lihat pada Gambar 2.18:



Gambar 7.5: Tampilan Session Layer Simulator

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



Gambar 7.6: Tampilan Session Event Log pada Sessin Layer Simulator

Bibliography

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

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

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

[4] M. Drajic and I. Kokic, "A comparative analysis of simulators ns2 and opnet it guru academic edition," in *2012 20th Telecommunications Forum (TELFOR)*, 2012, pp. 1780–1783. 5

[5] X. Xian, W. Shi, and H. Huang, "Comparison of omnet++ and other simulator for wsn simulation," in *2008 3rd IEEE Conference on Industrial Electronics and Applications*, 2008, pp. 1439–1443. 5