

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

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



Ir. Iwan Adhichandra, S.T, M.Sc, Ph.D., SMIEEE
NIP: 0018025806

Dosen Pengusul



Berkah Iman Santoso, S.T, MTI
NIDN : 0309068003

Daftar Isi

1	Kata Pengantar	2
2	Latar Belakang	3
3	Permasalahan	5
4	Tujuan Pengembangan Program Kuliah	6
5	Metode Pengembangan Program Kuliah	7
6	Sasaran dan Luaran	8
7	Uraian Pengembangan Program Kuliah	9
7.1	Lapisan Transport	9
7.1.1	Tampilan Simulator Transport Layer	57
7.2	Lapisan Sesi	58
7.2.1	Tampilan Simulator Session Layer	138

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 proprietary* 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 pemrograman 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

```
1 #!/usr/bin/env python3
2 """
3 =====
4 TRANSPORT LAYER SIMULATION
5 =====
6 Single-file Python 3 application using:
7
8     GTK3 / PyGObject
9     Matplotlib
10    NumPy
11    Pandas
12 Protocols:
13     TCP
14     UDP
15 Transport-layer simulation features:
16     TCP:
17         - Three-way handshake
```

- 18 - SYN
- 19 - SYN-ACK
- 20 - ACK
- 21 - Sequence numbers
- 22 - Acknowledgement numbers
- 23 - RTT
- 24 - RTO
- 25 - Retransmission
- 26 - Congestion Window
- 27 - Slow Start
- 28 - Congestion Avoidance

29 UDP:

- 30 - Connectionless transmission
- 31 - Datagram transmission
- 32 - Packet loss

33 General:

- 34 - Segmentation
- 35 - Packet transmission
- 36 - Packet loss
- 37 - Network delay
- 38 - PDR
- 39 - Loss rate
- 40 - Throughput
- 41 - Goodput
- 42 - Average RTT
- 43 - Average delay
- 44 - Retransmissions
- 45 - Live simulation graph
- 46 - Scrollable graph area
- 47 - Packet log
- 48 - CSV export

49 GUI:

- 50 - Resizable
- 51 - Maximizable
- 52 - Vertical graph scrolling
- 53 - No horizontal graph scrolling

54 Run:

55 python3 final_transport_layer_research_simulation.py

```
56 =====
57 """
58 import gi
59 gi.require_version("Gtk", "3.0")
60 from gi.repository import Gtk, GLib
61 import matplotlib
62 matplotlib.use("GTK3Agg")
63 from matplotlib.figure import Figure
```

```

64 from matplotlib.backends.backend_gtk3agg import FigureCanvasGTK3Agg
65 import numpy as np
66 import pandas as pd
67 import random
68 import time
69 import threading
70 # =====
71 # TRANSPORT SEGMENT
72 # =====
73 class TransportSegment:
74     def __init__(
75         self,
76         segment_id,
77         protocol,
78         source_port,
79         destination_port,
80         sequence_number,
81         payload_size
82     ):
83         self.segment_id = segment_id
84         self.protocol = protocol
85         self.source_port = source_port
86         self.destination_port = destination_port
87         self.sequence_number = sequence_number
88         self.ack_number = 0
89         self.payload_size = payload_size
90         self.status = "CREATED"
91         self.delay_ms = 0.0
92         self.rtt_ms = 0.0
93         self.retransmissions = 0
94         self.send_time = 0.0
95         self.ack_time = 0.0
96 # =====
97 # TCP CONNECTION
98 # =====
99 class TCPConnection:
100     def __init__(self):
101         self.state = "CLOSED"
102         self.client_sequence = 1000
103         self.server_sequence = 5000
104         self.ack_number = 0
105         self.cwnd = 1.0
106         self.ssthresh = 16.0
107         self.rtt_ms = 50.0
108         self.rto_ms = 100.0
109         self.total_ack = 0

```

```

110         self.total_retransmission = 0
111         self.handshake_time_ms = 0.0
112         # -----
113         # TCP THREE-WAY HANDSHAKE
114         # -----
115     def three_way_handshake(
116         self,
117         min_delay,
118         max_delay
119     ):
120         handshake_start = time.time()
121         # -----
122         # SYN
123         # -----
124         self.state = "SYN_SENT"
125         syn_delay = random.uniform(
126             min_delay,
127             max_delay
128         )
129         # -----
130         # SYN-ACK
131         # -----
132         self.state = "SYN_RECEIVED"
133         syn_ack_delay = random.uniform(
134             min_delay,
135             max_delay
136         )
137         # -----
138         # ACK
139         # -----
140         self.state = "ESTABLISHED"
141         ack_delay = random.uniform(
142             min_delay,
143             max_delay
144         )
145         measured_time = (
146             time.time() -
147             handshake_start
148         ) * 1000.0
149         self.handshake_time_ms = (
150             measured_time
151             +
152             syn_delay
153             +
154             syn_ack_delay
155             +

```

```

156         ack_delay
157     )
158     return self.handshake_time_ms
159 # =====
160 # TRANSPORT LAYER SIMULATOR
161 # =====
162 class TransportLayerSimulator:
163     def __init__(
164         self,
165         protocol="TCP",
166         segment_count=100,
167         segment_size=1024,
168         loss_probability=0.05,
169         min_delay=5.0,
170         max_delay=50.0
171     ):
172         self.protocol = protocol
173         self.segment_count = segment_count
174         self.segment_size = segment_size
175         self.loss_probability = loss_probability
176         self.min_delay = min_delay
177         self.max_delay = max_delay
178         self.source_port = 50000
179         self.destination_port = 443
180         self.tcp = TCPConnection()
181         self.segments = []
182         self.statistics = []
183         self.running = False
184         self.start_time = None
185         self.end_time = None
186         self.generated = 0
187         self.transmitted = 0
188         self.delivered = 0
189         self.lost = 0
190         self.acknowledged = 0
191         self.retransmissions = 0
192         self.total_delay = 0.0
193         self.total_rtt = 0.0
194         self.total_payload_bytes = 0
195         self.total_goodput_bytes = 0
196 # -----
197 # RESET
198 # -----
199     def reset(self):
200         self.segments = []
201         self.statistics = []

```

```

202     self.running = False
203     self.start_time = None
204     self.end_time = None
205     self.generated = 0
206     self.transmitted = 0
207     self.delivered = 0
208     self.lost = 0
209     self.acknowledged = 0
210     self.retransmissions = 0
211     self.total_delay = 0.0
212     self.total_rtt = 0.0
213     self.total_payload_bytes = 0
214     self.total_goodput_bytes = 0
215     self.tcp = TCPConnection()
216     # -----
217     # TCP HANDSHAKE
218     # -----
219     def perform_tcp_handshake(self):
220         if self.protocol != "TCP":
221             return 0.0
222         return self.tcp.three_way_handshake(
223             self.min_delay,
224             self.max_delay
225         )
226     # -----
227     # CREATE SEGMENT
228     # -----
229     def create_segment(
230         self,
231         segment_id
232     ):
233         if self.protocol == "TCP":
234             sequence_number = (
235                 self.tcp.client_sequence
236                 +
237                 (
238                     segment_id - 1
239                 )
240                 *
241                 self.segment_size
242             )
243         else:
244             sequence_number = (
245                 segment_id
246                 *
247                 self.segment_size

```

```

248         )
249     return TransportSegment(
250         segment_id=segment_id,
251         protocol=self.protocol,
252         source_port=self.source_port,
253         destination_port=self.destination_port,
254         sequence_number=sequence_number,
255         payload_size=self.segment_size
256     )
257     # -----
258     # TRANSMIT SEGMENT
259     # -----
260     def transmit_segment(
261         self,
262         segment
263     ):
264         segment.status = "TRANSMITTING"
265         self.transmitted += 1
266         segment.send_time = time.time()
267         # -----
268         # TRANSMISSION DELAY
269         # -----
270         segment.delay_ms = random.uniform(
271             self.min_delay,
272             self.max_delay
273         )
274         # -----
275         # PACKET LOSS
276         # -----
277         lost = (
278             random.random()
279             <
280             self.loss_probability
281         )
282         if lost:
283             segment.status = "LOST"
284             self.lost += 1
285             # =====
286             # TCP RETRANSMISSION
287             # =====
288             if self.protocol == "TCP":
289                 segment.retransmissions += 1
290                 self.retransmissions += 1
291                 self.tcp.total_retransmission += 1
292                 retry_delay = min(
293                     self.tcp.rto_ms,

```

```

294         1000.0
295     )
296     segment.delay_ms += retry_delay
297     recovered = (
298         random.random()
299         >
300         self.loss_probability
301     )
302     if recovered:
303         segment.status = "RETRANSMITTED"
304         segment.delay_ms += random.uniform(
305             self.min_delay,
306             self.max_delay
307         )
308     else:
309         return segment
310     # =====
311     # UDP DOES NOT RETRANSMIT
312     # =====
313     else:
314         return segment
315     # =====
316     # TCP ACK
317     # =====
318     if self.protocol == "TCP":
319         segment.ack_number = (
320             segment.sequence_number
321             +
322             segment.payload_size
323         )
324         ack_delay = random.uniform(
325             self.min_delay,
326             self.max_delay
327         )
328         segment.rtt_ms = (
329             segment.delay_ms
330             +
331             ack_delay
332         )
333         self.total_rtt += segment.rtt_ms
334         self.acknowledged += 1
335         self.tcp.total_ack += 1
336         # -----
337         # TCP CONGESTION CONTROL
338         # -----
339         if self.tcp.cwnd < self.tcp.ssthresh:

```

```

340         # Slow Start
341         self.tcp.cwnd += 1.0
342     else:
343         # Congestion Avoidance
344         self.tcp.cwnd += (
345             1.0
346             /
347             max(
348                 self.tcp.cwnd,
349                 1.0
350             )
351         )
352     # -----
353     # RTT ESTIMATION
354     # -----
355     self.tcp.rtt_ms = (
356         0.875
357         *
358         self.tcp.rtt_ms
359         +
360         0.125
361         *
362         segment.rtt_ms
363     )
364     # -----
365     # RTO ESTIMATION
366     # -----
367     self.tcp.rto_ms = max(
368         100.0,
369         2.0
370         *
371         self.tcp.rtt_ms
372     )
373     segment.status = "ACKED"
374     # =====
375     # UDP
376     # =====
377     else:
378         segment.status = "DELIVERED"
379     # =====
380     # SUCCESSFUL DELIVERY
381     # =====
382     self.delivered += 1
383     self.total_delay += segment.delay_ms
384     self.total_payload_bytes += segment.payload_size
385     self.total_goodput_bytes += segment.payload_size

```

```

386         return segment
387     # -----
388     # RUN SIMULATION
389     # -----
390     def run(
391         self,
392         progress_callback=None,
393         segment_callback=None
394     ):
395         self.reset()
396         self.running = True
397         self.start_time = time.time()
398         handshake_ms = 0.0
399         if self.protocol == "TCP":
400             handshake_ms = (
401                 self.perform_tcp_handshake()
402             )
403         # -----
404         # SEGMENT LOOP
405         # -----
406         for segment_id in range(
407             1,
408             self.segment_count + 1
409         ):
410             if not self.running:
411                 break
412             segment = self.create_segment(
413                 segment_id
414             )
415             self.generated += 1
416             segment = self.transmit_segment(
417                 segment
418             )
419             self.segments.append(
420                 segment
421             )
422             elapsed = (
423                 time.time()
424                 -
425                 self.start_time
426             )
427             # -----
428             # PDR
429             # -----
430             pdr = 0.0
431             if self.generated > 0:

```

```

432         pdr = (
433             self.delivered
434             /
435             self.generated
436         ) * 100.0
437     # -----
438     # LOSS RATE
439     # -----
440     loss_rate = 0.0
441     if self.generated > 0:
442         loss_rate = (
443             self.lost
444             /
445             self.generated
446         ) * 100.0
447     # -----
448     # THROUGHPUT
449     # -----
450     throughput_bps = 0.0
451     if elapsed > 0:
452         throughput_bps = (
453             self.total_goodput_bytes
454             *
455             8
456         ) / elapsed
457     # -----
458     # AVERAGE RTT
459     # -----
460     average_rtt = 0.0
461     if self.acknowledged > 0:
462         average_rtt = (
463             self.total_rtt
464             /
465             self.acknowledged
466         )
467     # -----
468     # AVERAGE DELAY
469     # -----
470     average_delay = 0.0
471     if self.delivered > 0:
472         average_delay = (
473             self.total_delay
474             /
475             self.delivered
476         )
477     # -----

```

```

478     # RECORD
479     # -----
480     self.statistics.append(
481         {
482             "segment_id":
483                 segment.segment_id,
484             "protocol":
485                 segment.protocol,
486             "source_port":
487                 segment.source_port,
488             "destination_port":
489                 segment.destination_port,
490             "sequence_number":
491                 segment.sequence_number,
492             "ack_number":
493                 segment.ack_number,
494             "payload_bytes":
495                 segment.payload_size,
496             "status":
497                 segment.status,
498             "delay_ms":
499                 segment.delay_ms,
500             "rtt_ms":
501                 segment.rtt_ms,
502             "retransmissions":
503                 segment.retransmissions,
504             "cwnd":
505                 (
506                     self.tcp.cwnd
507                     if
508                     self.protocol == "TCP"
509                     else
510                     0.0
511                 ),
512             "rto_ms":
513                 (
514                     self.tcp.rto_ms
515                     if
516                     self.protocol == "TCP"
517                     else
518                     0.0
519                 ),
520             "generated":
521                 self.generated,
522             "transmitted":
523                 self.transmitted,

```

```

524         "delivered":
525             self.delivered,
526         "lost":
527             self.lost,
528         "acknowledged":
529             self.acknowledged,
530         "total_retransmissions":
531             self.retransmissions,
532         "pdr":
533             pdr,
534         "loss_rate":
535             loss_rate,
536         "throughput_bps":
537             throughput_bps,
538         "average_rtt_ms":
539             average_rtt,
540         "average_delay_ms":
541             average_delay,
542         "elapsed_seconds":
543             elapsed,
544         "handshake_ms":
545             handshake_ms
546     }
547 )
548 # -----
549 # CALLBACKS
550 # -----
551 if progress_callback:
552     progress_callback(
553         self.generated,
554         self.segment_count
555     )
556 if segment_callback:
557     segment_callback(
558         segment
559     )
560 # -----
561 # SLOW DOWN SIMULATION FOR LIVE GRAPH
562 # -----
563 time.sleep(
564     0.03
565 )
566 self.end_time = time.time()
567 self.running = False
568 # -----
569 # STOP

```

```

570 # -----
571 def stop(self):
572     self.running = False
573 # -----
574 # DATAFRAME
575 # -----
576 def dataframe(self):
577     return pd.DataFrame(
578         self.statistics
579     )
580 # -----
581 # METRICS
582 # -----
583 def get_metrics(self):
584     elapsed = 0.0
585     if (
586         self.start_time is not None
587         and
588         self.end_time is not None
589     ):
590         elapsed = (
591             self.end_time
592             -
593             self.start_time
594         )
595     pdr = 0.0
596     if self.generated > 0:
597         pdr = (
598             self.delivered
599             /
600             self.generated
601         ) * 100.0
602     loss_rate = 0.0
603     if self.generated > 0:
604         loss_rate = (
605             self.lost
606             /
607             self.generated
608         ) * 100.0
609     average_delay = 0.0
610     if self.delivered > 0:
611         average_delay = (
612             self.total_delay
613             /
614             self.delivered
615         )

```

```

616         average_rtt = 0.0
617         if self.acknowledged > 0:
618             average_rtt = (
619                 self.total_rtt
620                 /
621                 self.acknowledged
622             )
623         throughput_bps = 0.0
624         if elapsed > 0:
625             throughput_bps = (
626                 self.total_goodput_bytes
627                 *
628                 8
629             ) / elapsed
630         return {
631             "generated":
632                 self.generated,
633             "transmitted":
634                 self.transmitted,
635             "delivered":
636                 self.delivered,
637             "lost":
638                 self.lost,
639             "acknowledged":
640                 self.acknowledged,
641             "retransmissions":
642                 self.retransmissions,
643             "pdr":
644                 pdr,
645             "loss_rate":
646                 loss_rate,
647             "average_delay":
648                 average_delay,
649             "average_rtt":
650                 average_rtt,
651             "throughput":
652                 throughput_bps,
653             "elapsed":
654                 elapsed,
655             "handshake":
656                 self.tcp.handshake_time_ms,
657             "cwnd":
658                 self.tcp.cwnd,
659             "rto":
660                 self.tcp.rto_ms
661         }

```

```

662 # =====
663 # GTK APPLICATION
664 # =====
665 class TransportLayerGTK(Gtk.Window):
666     def __init__(self):
667         super().__init__(
668             title="Transport Layer Simulation"
669         )
670         # =====
671         # WINDOW CONFIGURATION
672         # =====
673         self.set_default_size(
674             1280,
675             800
676         )
677         self.set_size_request(
678             900,
679             600
680         )
681         self.set_border_width(
682             8
683         )
684         self.set_resizable(
685             True
686         )
687         self.set_position(
688             Gtk.WindowPosition.CENTER
689         )
690         self.simulator = (
691             TransportLayerSimulator()
692         )
693         self.worker_thread = None
694         self.build_ui()
695         # =====
696         # BUILD UI
697         # =====
698         def build_ui(self):
699             main_box = Gtk.Box(
700                 orientation=Gtk.Orientation.VERTICAL,
701                 spacing=6
702             )
703             self.add(
704                 main_box
705             )
706             # =====
707             # TITLE

```

```

708 # =====
709 title = Gtk.Label()
710 title.set_markup(
711     "<span size='xx-large' "
712     "weight='bold'>"
713     "Transport Layer Simulation"
714     "</span>"
715 )
716 main_box.pack_start(
717     title,
718     False,
719     False,
720     3
721 )
722 subtitle = Gtk.Label(
723     label=(
724         "TCP / UDP      segmentation      "
725         "transmission    ACK      "
726         "retransmission    throughput"
727     )
728 )
729 main_box.pack_start(
730     subtitle,
731     False,
732     False,
733     2
734 )
735 # =====
736 # PARAMETERS
737 # =====
738 parameter_frame = Gtk.Frame(
739     label="Simulation Parameters"
740 )
741 parameter_grid = Gtk.Grid()
742 parameter_grid.set_border_width(
743     6
744 )
745 parameter_grid.set_row_spacing(
746     5
747 )
748 parameter_grid.set_column_spacing(
749     7
750 )
751 parameter_frame.add(
752     parameter_grid
753 )

```

```

754     main_box.pack_start(
755         parameter_frame,
756         False,
757         False,
758         0
759     )
760     # -----
761     # Protocol
762     # -----
763     parameter_grid.attach(
764         Gtk.Label(
765             label="Protocol:"
766         ),
767         0,
768         0,
769         1,
770         1
771     )
772     self.protocol_combo = (
773         Gtk.ComboBoxText()
774     )
775     self.protocol_combo.append_text(
776         "TCP"
777     )
778     self.protocol_combo.append_text(
779         "UDP"
780     )
781     self.protocol_combo.set_active(
782         0
783     )
784     parameter_grid.attach(
785         self.protocol_combo,
786         1,
787         0,
788         1,
789         1
790     )
791     # -----
792     # Segment Count
793     # -----
794     parameter_grid.attach(
795         Gtk.Label(
796             label="Segments:"
797         ),
798         2,
799         0,

```

```

800         1,
801         1
802     )
803     self.segment_entry = (
804         Gtk.Entry()
805     )
806     self.segment_entry.set_text(
807         "100"
808     )
809     parameter_grid.attach(
810         self.segment_entry,
811         3,
812         0,
813         1,
814         1
815     )
816     # -----
817     # Payload
818     # -----
819     parameter_grid.attach(
820         Gtk.Label(
821             label="Payload Bytes:"
822         ),
823         4,
824         0,
825         1,
826         1
827     )
828     self.size_entry = (
829         Gtk.Entry()
830     )
831     self.size_entry.set_text(
832         "1024"
833     )
834     parameter_grid.attach(
835         self.size_entry,
836         5,
837         0,
838         1,
839         1
840     )
841     # -----
842     # Loss
843     # -----
844     parameter_grid.attach(
845         Gtk.Label(

```

```

846         label="Loss Probability:"
847     ),
848     0,
849     1,
850     1,
851     1
852 )
853 self.loss_entry = (
854     Gtk.Entry()
855 )
856 self.loss_entry.set_text(
857     "0.05"
858 )
859 parameter_grid.attach(
860     self.loss_entry,
861     1,
862     1,
863     1,
864     1
865 )
866 # -----
867 # Minimum Delay
868 # -----
869 parameter_grid.attach(
870     Gtk.Label(
871         label="Min Delay (ms):"
872     ),
873     2,
874     1,
875     1,
876     1
877 )
878 self.min_delay_entry = (
879     Gtk.Entry()
880 )
881 self.min_delay_entry.set_text(
882     "5"
883 )
884 parameter_grid.attach(
885     self.min_delay_entry,
886     3,
887     1,
888     1,
889     1
890 )
891 # -----

```

```

892     # Maximum Delay
893     # -----
894     parameter_grid.attach(
895         Gtk.Label(
896             label="Max Delay (ms):"
897         ),
898         4,
899         1,
900         1,
901         1
902     )
903     self.max_delay_entry = (
904         Gtk.Entry()
905     )
906     self.max_delay_entry.set_text(
907         "50"
908     )
909     parameter_grid.attach(
910         self.max_delay_entry,
911         5,
912         1,
913         1,
914         1
915     )
916     # =====
917     # BUTTONS
918     # =====
919     button_box = Gtk.Box(
920         orientation=Gtk.Orientation.HORIZONTAL,
921         spacing=7
922     )
923     self.start_button = Gtk.Button(
924         label="Start Simulation"
925     )
926     self.start_button.connect(
927         "clicked",
928         self.start_simulation
929     )
930     button_box.pack_start(
931         self.start_button,
932         False,
933         False,
934         0
935     )
936     self.stop_button = Gtk.Button(
937         label="Stop"

```

```

938     )
939     self.stop_button.connect(
940         "clicked",
941         self.stop_simulation
942     )
943     button_box.pack_start(
944         self.stop_button,
945         False,
946         False,
947         0
948     )
949     export_button = Gtk.Button(
950         label="Export CSV"
951     )
952     export_button.connect(
953         "clicked",
954         self.export_csv
955     )
956     button_box.pack_start(
957         export_button,
958         False,
959         False,
960         0
961     )
962     parameter_grid.attach(
963         button_box,
964         0,
965         2,
966         6,
967         1
968     )
969     # =====
970     # PROGRESS
971     # =====
972     self.progress = (
973         Gtk.ProgressBar()
974     )
975     self.progress.set_show_text(
976         True
977     )
978     main_box.pack_start(
979         self.progress,
980         False,
981         False,
982         0
983     )

```

```

984 # =====
985 # METRICS
986 # =====
987 metric_frame = Gtk.Frame(
988     label="Transport Layer Metrics"
989 )
990 metric_grid = Gtk.Grid()
991 metric_grid.set_border_width(
992     5
993 )
994 metric_grid.set_column_spacing(
995     10
996 )
997 metric_grid.set_row_spacing(
998     4
999 )
1000 metric_frame.add(
1001     metric_grid
1002 )
1003 main_box.pack_start(
1004     metric_frame,
1005     False,
1006     False,
1007     0
1008 )
1009 self.metric_labels = {}
1010 metrics = [
1011     ("Generated", "generated"),
1012     ("Delivered", "delivered"),
1013     ("Lost", "lost"),
1014     ("ACK", "acknowledged"),
1015     ("Retrans.", "retransmissions"),
1016     ("PDR", "pdr"),
1017     ("Loss", "loss_rate"),
1018     ("RTT", "average_rtt"),
1019     ("Delay", "average_delay"),
1020     ("Throughput", "throughput")
1021 ]
1022 for index, (
1023     caption,
1024     key
1025 ) in enumerate(metrics):
1026     metric_box = Gtk.Box(
1027         orientation=
1028             Gtk.Orientation.VERTICAL,
1029         spacing=1

```

```

1030         )
1031         caption_label = Gtk.Label(
1032             label=caption
1033         )
1034         value_label = Gtk.Label(
1035             label="0"
1036         )
1037         value_label.set_markup(
1038             "<b>0</b>"
1039         )
1040         metric_box.pack_start(
1041             caption_label,
1042             False,
1043             False,
1044             0
1045         )
1046         metric_box.pack_start(
1047             value_label,
1048             False,
1049             False,
1050             0
1051         )
1052         metric_grid.attach(
1053             metric_box,
1054             index,
1055             0,
1056             1,
1057             1
1058         )
1059         self.metric_labels[
1060             key
1061         ] = value_label
1062         # =====
1063         # NOTEBOOK
1064         # =====
1065         notebook = Gtk.Notebook()
1066         notebook.set_vexpand(
1067             True
1068         )
1069         notebook.set_hexpand(
1070             True
1071         )
1072         main_box.pack_start(
1073             notebook,
1074             True,
1075             True,

```

```

1076         0
1077     )
1078     # =====
1079     # LIVE SIMULATION GRAPH
1080     # =====
1081     graph_box = Gtk.Box(
1082         orientation=
1083             Gtk.Orientation.VERTICAL,
1084         spacing=4
1085     )
1086     notebook.append_page(
1087         graph_box,
1088         Gtk.Label(
1089             label="Live Simulation Graph"
1090         )
1091     )
1092     # -----
1093     # GRAPH SCROLL WINDOW
1094     # -----
1095     self.graph_scroll = (
1096         Gtk.ScrolledWindow()
1097     )
1098     # Vertical scrolling only.
1099     self.graph_scroll.set_policy(
1100         Gtk.PolicyType.NEVER,
1101         Gtk.PolicyType.AUTOMATIC
1102     )
1103     self.graph_scroll.set_shadow_type(
1104         Gtk.ShadowType.IN
1105     )
1106     self.graph_scroll.set_vexpand(
1107         True
1108     )
1109     self.graph_scroll.set_hexpand(
1110         True
1111     )
1112     graph_box.pack_start(
1113         self.graph_scroll,
1114         True,
1115         True,
1116         0
1117     )
1118     # -----
1119     # GRAPH CONTAINER
1120     # -----
1121     self.graph_container = Gtk.Box(

```

```

1122         orientation=
1123             Gtk.Orientation.VERTICAL
1124     )
1125     self.graph_scroll.add(
1126         self.graph_container
1127     )
1128     # -----
1129     # MATPLOTLIB FIGURE
1130     # -----
1131     self.figure = Figure(
1132         figsize=(13, 16),
1133         dpi=100
1134     )
1135     self.canvas = (
1136         FigureCanvasGTK3Agg(
1137             self.figure
1138         )
1139     )
1140     # Large vertical canvas.
1141     # Gtk.ScrolledWindow handles scrolling.
1142     self.canvas.set_size_request(
1143         1100,
1144         1600
1145     )
1146     self.graph_container.pack_start(
1147         self.canvas,
1148         False,
1149         False,
1150         0
1151     )
1152     # =====
1153     # SEGMENT LOG
1154     # =====
1155     packet_box = Gtk.Box(
1156         orientation=
1157             Gtk.Orientation.VERTICAL
1158     )
1159     notebook.append_page(
1160         packet_box,
1161         Gtk.Label(
1162             label="Segment Log"
1163         )
1164     )
1165     self.packet_store = Gtk.ListStore(
1166         int,      # ID
1167         str,      # Protocol

```

```

1168         int,          # Source port
1169         int,          # Destination port
1170         int,          # Sequence
1171         str,          # Status
1172         float,        # Delay
1173         float,        # RTT
1174         int           # Retransmissions
1175     )
1176     tree = Gtk.TreeView(
1177         model=self.packet_store
1178     )
1179     columns = [
1180         ("ID", 0),
1181         ("Protocol", 1),
1182         ("Src Port", 2),
1183         ("Dst Port", 3),
1184         ("Sequence", 4),
1185         ("Status", 5),
1186         ("Delay ms", 6),
1187         ("RTT ms", 7),
1188         ("Retrans.", 8)
1189     ]
1190     for title, index in columns:
1191         renderer = (
1192             Gtk.CellRendererText()
1193         )
1194         column = (
1195             Gtk.TreeViewColumn(
1196                 title,
1197                 renderer,
1198                 text=index
1199             )
1200         )
1201         tree.append_column(
1202             column
1203         )
1204     packet_scroll = (
1205         Gtk.ScrolledWindow()
1206     )
1207     packet_scroll.set_policy(
1208         Gtk.PolicyType.AUTOMATIC,
1209         Gtk.PolicyType.AUTOMATIC
1210     )
1211     packet_scroll.add(
1212         tree
1213     )

```

```

1214     packet_box.pack_start(
1215         packet_scroll,
1216         True,
1217         True,
1218         0
1219     )
1220     # =====
1221     # TRANSPORT PROCESS
1222     # =====
1223     process_box = Gtk.Box(
1224         orientation=
1225             Gtk.Orientation.VERTICAL,
1226         spacing=10
1227     )
1228     notebook.append_page(
1229         process_box,
1230         Gtk.Label(
1231             label="Transport Process"
1232         )
1233     )
1234     process_scroll = (
1235         Gtk.ScrolledWindow()
1236     )
1237     process_scroll.set_policy(
1238         Gtk.PolicyType.NEVER,
1239         Gtk.PolicyType.AUTOMATIC
1240     )
1241     process_box.pack_start(
1242         process_scroll,
1243         True,
1244         True,
1245         0
1246     )
1247     process_label = Gtk.Label()
1248     process_label.set_markup(
1249         "<b>TCP TRANSPORT PROCESS</b>\n\n"
1250         "Application Data\n"
1251         "    \n"
1252         "Segmentation\n"
1253         "    \n"
1254         "TCP Header Creation\n"
1255         "    \n"
1256         "Three-Way Handshake\n"
1257         "    \n"
1258         "SYN\n"
1259         "    \n"

```

```

1260 "SYN-ACK\n"
1261 "          \n"
1262 "ACK\n"
1263 "          \n"
1264 "ESTABLISHED\n"
1265 "          \n"
1266 "Sequence Number Assignment\n"
1267 "          \n"
1268 "Segment Transmission\n"
1269 "          \n"
1270 "ACK Reception\n"
1271 "          \n"
1272 "RTT Measurement\n"
1273 "          \n"
1274 "Congestion Window Update\n"
1275 "          \n"
1276 "Slow Start / Congestion Avoidance\n"
1277 "          \n"
1278 "Packet Loss Detection\n"
1279 "          \n"
1280 "Retransmission\n"
1281 "          \n"
1282 "Reliable Delivery\n\n"
1283 "<b>UDP TRANSPORT PROCESS</b>\n\n"
1284 "Application Data\n"
1285 "          \n"
1286 "Datagram Creation\n"
1287 "          \n"
1288 "UDP Header Creation\n"
1289 "          \n"
1290 "Source Port / Destination Port\n"
1291 "          \n"
1292 "Datagram Transmission\n"
1293 "          \n"
1294 "Delivery or Loss\n\n"
1295 "<b>SIMULATION METRICS</b>\n\n"
1296 "    Packet Delivery Ratio\n"
1297 "    Packet Loss Rate\n"
1298 "    Throughput\n"
1299 "    Goodput\n"
1300 "    RTT\n"
1301 "    Delay\n"
1302 "    TCP Retransmissions\n"
1303 "    TCP Congestion Window\n"
1304 "    TCP RTO\n"
1305 "    ACK Progression"

```

```

1306         )
1307         process_label.set_xalign(
1308             0
1309         )
1310         process_label.set_yalign(
1311             0
1312         )
1313         process_label.set_margin_start(
1314             20
1315         )
1316         process_label.set_margin_top(
1317             20
1318         )
1319         process_scroll.add(
1320             process_label
1321         )
1322         # =====
1323         # STATUS
1324         # =====
1325         self.status = Gtk.Label(
1326             label="Ready."
1327         )
1328         main_box.pack_start(
1329             self.status,
1330             False,
1331             False,
1332             0
1333         )
1334         # =====
1335         # READ PARAMETERS
1336         # =====
1337         def get_parameters(self):
1338             try:
1339                 protocol = (
1340                     self.protocol_combo
1341                     .get_active_text()
1342                 )
1343                 segment_count = int(
1344                     self.segment_entry
1345                     .get_text()
1346                 )
1347                 segment_size = int(
1348                     self.size_entry
1349                     .get_text()
1350                 )
1351                 loss_probability = float(

```

```

1352         self.loss_entry
1353         .get_text()
1354     )
1355     min_delay = float(
1356         self.min_delay_entry
1357         .get_text()
1358     )
1359     max_delay = float(
1360         self.max_delay_entry
1361         .get_text()
1362     )
1363     # -----
1364     # VALIDATION
1365     # -----
1366     if protocol not in (
1367         "TCP",
1368         "UDP"
1369     ):
1370         raise ValueError(
1371             "Protocol must be TCP or UDP."
1372         )
1373     if segment_count <= 0:
1374         raise ValueError(
1375             "Segment count must be greater than zero."
1376         )
1377     if segment_size <= 0:
1378         raise ValueError(
1379             "Payload size must be greater than zero."
1380         )
1381     if not (
1382         0.0
1383         <=
1384         loss_probability
1385         <=
1386         1.0
1387     ):
1388         raise ValueError(
1389             "Loss probability must be between 0 and 1."
1390         )
1391     if min_delay < 0:
1392         raise ValueError(
1393             "Minimum delay cannot be negative."
1394         )
1395     if max_delay < min_delay:
1396         raise ValueError(
1397             "Maximum delay must be greater than "

```

```

1398         "or equal to minimum delay."
1399     )
1400     return (
1401         protocol,
1402         segment_count,
1403         segment_size,
1404         loss_probability,
1405         min_delay,
1406         max_delay
1407     )
1408 except ValueError as error:
1409     self.error_dialog(
1410         str(error)
1411     )
1412     return None
1413 # =====
1414 # START SIMULATION
1415 # =====
1416 def start_simulation(
1417     self,
1418     button
1419 ):
1420     parameters = (
1421         self.get_parameters()
1422     )
1423     if parameters is None:
1424         return
1425     (
1426         protocol,
1427         segment_count,
1428         segment_size,
1429         loss_probability,
1430         min_delay,
1431         max_delay
1432     ) = parameters
1433     self.simulator = (
1434         TransportLayerSimulator(
1435             protocol=
1436                 protocol,
1437             segment_count=
1438                 segment_count,
1439             segment_size=
1440                 segment_size,
1441             loss_probability=
1442                 loss_probability,
1443             min_delay=

```

```

1444         min_delay,
1445         max_delay=
1446         max_delay
1447     )
1448 )
1449 # -----
1450 # RESET GUI
1451 # -----
1452 self.packet_store.clear()
1453 self.figure.clear()
1454 self.canvas.draw()
1455 self.progress.set_fraction(
1456     0.0
1457 )
1458 self.progress.set_text(
1459     "0%"
1460 )
1461 self.start_button.set_sensitive(
1462     False
1463 )
1464 self.status.set_text(
1465     f"{protocol} simulation running..."
1466 )
1467 # -----
1468 # START WORKER THREAD
1469 # -----
1470 self.worker_thread = (
1471     threading.Thread(
1472         target=
1473         self.run_worker,
1474         daemon=True
1475     )
1476 )
1477 self.worker_thread.start()
1478 # =====
1479 # WORKER
1480 # =====
1481 def run_worker(self):
1482     self.simulator.run(
1483         progress_callback=
1484         self.progress_callback,
1485         segment_callback=
1486         self.segment_callback
1487     )
1488     GLib.idle_add(
1489         self.simulation_finished

```

```

1490         )
1491     # =====
1492     # SEGMENT CALLBACK
1493     # =====
1494     def segment_callback(
1495         self,
1496         segment
1497     ):
1498         GLib.idle_add(
1499             self.update_live_view
1500         )
1501     # =====
1502     # PROGRESS CALLBACK
1503     # =====
1504     def progress_callback(
1505         self,
1506         current,
1507         total
1508     ):
1509         GLib.idle_add(
1510             self.update_progress,
1511             current,
1512             total
1513         )
1514     # =====
1515     # UPDATE PROGRESS
1516     # =====
1517     def update_progress(
1518         self,
1519         current,
1520         total
1521     ):
1522         fraction = (
1523             current
1524             /
1525             total
1526         )
1527         self.progress.set_fraction(
1528             fraction
1529         )
1530         self.progress.set_text(
1531             f"{current}/{total} "
1532             f"({fraction * 100:.1f}%)"
1533         )
1534         self.status.set_text(
1535             f"Processing segment "

```

```

1536         f"{current}/{total}"
1537     )
1538     self.update_live_view()
1539     return False
1540 # =====
1541 # UPDATE LIVE VIEW
1542 # =====
1543 def update_live_view(self):
1544     dataframe = (
1545         self.simulator.dataframe()
1546     )
1547     if dataframe.empty:
1548         return False
1549     self.update_packet_table()
1550     self.update_metrics()
1551     self.update_graph()
1552     return False
1553 # =====
1554 # UPDATE GRAPH
1555 # =====
1556 def update_graph(self):
1557     dataframe = (
1558         self.simulator.dataframe()
1559     )
1560     if dataframe.empty:
1561         return
1562     # -----
1563     # CLEAR PREVIOUS FIGURE
1564     # -----
1565     self.figure.clear()
1566     x = (
1567         dataframe["segment_id"]
1568     )
1569     # =====
1570     # 1. RTT
1571     # =====
1572     ax1 = self.figure.add_subplot(
1573         5,
1574         2,
1575         1
1576     )
1577     ax1.plot(
1578         x,
1579         dataframe["rtt_ms"],
1580         linewidth=1.5
1581     )

```

```

1582     ax1.set_title(
1583         "Round Trip Time (RTT)",
1584         fontsize=11,
1585         fontweight="bold"
1586     )
1587     ax1.set_xlabel(
1588         "Segment"
1589     )
1590     ax1.set_ylabel(
1591         "RTT (ms)"
1592     )
1593     ax1.grid(
1594         True,
1595         alpha=0.3
1596     )
1597     # =====
1598     # 2. TRANSPORT DELAY
1599     # =====
1600     ax2 = self.figure.add_subplot(
1601         5,
1602         2,
1603         2
1604     )
1605     ax2.plot(
1606         x,
1607         dataframe["delay_ms"],
1608         linewidth=1.5
1609     )
1610     ax2.set_title(
1611         "Transport Delay",
1612         fontsize=11,
1613         fontweight="bold"
1614     )
1615     ax2.set_xlabel(
1616         "Segment"
1617     )
1618     ax2.set_ylabel(
1619         "Delay (ms)"
1620     )
1621     ax2.grid(
1622         True,
1623         alpha=0.3
1624     )
1625     # =====
1626     # 3. TCP CONGESTION WINDOW
1627     # =====

```

```

1628     ax3 = self.figure.add_subplot(
1629         5,
1630         2,
1631         3
1632     )
1633     if self.simulator.protocol == "TCP":
1634         ax3.plot(
1635             x,
1636             dataframe["cwnd"],
1637             linewidth=2
1638         )
1639         ax3.set_title(
1640             "TCP Congestion Window (cwnd)",
1641             fontsize=11,
1642             fontweight="bold"
1643         )
1644         ax3.set_ylabel(
1645             "cwnd"
1646         )
1647     else:
1648         ax3.text(
1649             0.5,
1650             0.5,
1651             "UDP does not use TCP cwnd",
1652             ha="center",
1653             va="center",
1654             transform=ax3.transAxes
1655         )
1656         ax3.set_title(
1657             "Congestion Window"
1658         )
1659     ax3.set_xlabel(
1660         "Segment"
1661     )
1662     ax3.grid(
1663         True,
1664         alpha=0.3
1665     )
1666     # =====
1667     # 4. PACKET DELIVERY RATIO
1668     # =====
1669     ax4 = self.figure.add_subplot(
1670         5,
1671         2,
1672         4
1673     )

```

```

1674     ax4.plot(
1675         x,
1676         dataframe["pdr"],
1677         linewidth=2
1678     )
1679     ax4.set_title(
1680         "Packet Delivery Ratio (PDR)",
1681         fontsize=11,
1682         fontweight="bold"
1683     )
1684     ax4.set_xlabel(
1685         "Segment"
1686     )
1687     ax4.set_ylabel(
1688         "PDR (%)"
1689     )
1690     ax4.set_ylim(
1691         0,
1692         105
1693     )
1694     ax4.grid(
1695         True,
1696         alpha=0.3
1697     )
1698     # =====
1699     # 5. THROUGHPUT
1700     # =====
1701     ax5 = self.figure.add_subplot(
1702         5,
1703         2,
1704         5
1705     )
1706     throughput = (
1707         dataframe[
1708             "throughput_bps"
1709         ].to_numpy()
1710         /
1711         1_000_000.0
1712     )
1713     ax5.plot(
1714         x,
1715         throughput,
1716         linewidth=2
1717     )
1718     ax5.set_title(
1719         "Transport Layer Throughput",

```

```

1720         fontsize=11,
1721         fontweight="bold"
1722     )
1723     ax5.set_xlabel(
1724         "Segment"
1725     )
1726     ax5.set_ylabel(
1727         "Throughput (Mbps)"
1728     )
1729     ax5.grid(
1730         True,
1731         alpha=0.3
1732     )
1733     # =====
1734     # 6. RETRANSMISSIONS
1735     # =====
1736     ax6 = self.figure.add_subplot(
1737         5,
1738         2,
1739         6
1740     )
1741     ax6.plot(
1742         x,
1743         dataframe[
1744             "total_retransmissions"
1745         ],
1746         linewidth=2
1747     )
1748     ax6.set_title(
1749         "Cumulative TCP Retransmissions",
1750         fontsize=11,
1751         fontweight="bold"
1752     )
1753     ax6.set_xlabel(
1754         "Segment"
1755     )
1756     ax6.set_ylabel(
1757         "Retransmissions"
1758     )
1759     ax6.grid(
1760         True,
1761         alpha=0.3
1762     )
1763     # =====
1764     # 7. DELIVERED VS LOST
1765     # =====

```

```

1766     ax7 = self.figure.add_subplot(
1767         5,
1768         2,
1769         7
1770     )
1771     ax7.plot(
1772         x,
1773         dataframe["delivered"],
1774         linewidth=2,
1775         label="Delivered"
1776     )
1777     ax7.plot(
1778         x,
1779         dataframe["lost"],
1780         linewidth=2,
1781         label="Lost"
1782     )
1783     ax7.set_title(
1784         "Delivered vs Lost",
1785         fontsize=11,
1786         fontweight="bold"
1787     )
1788     ax7.set_xlabel(
1789         "Segment"
1790     )
1791     ax7.set_ylabel(
1792         "Count"
1793     )
1794     ax7.legend(
1795         fontsize=8
1796     )
1797     ax7.grid(
1798         True,
1799         alpha=0.3
1800     )
1801     # =====
1802     # 8. ACK
1803     # =====
1804     ax8 = self.figure.add_subplot(
1805         5,
1806         2,
1807         8
1808     )
1809     ax8.plot(
1810         x,
1811         dataframe["acknowledged"],

```

```

1812         linewidth=2
1813     )
1814     ax8.set_title(
1815         "Cumulative TCP ACK",
1816         fontsize=11,
1817         fontweight="bold"
1818     )
1819     ax8.set_xlabel(
1820         "Segment"
1821     )
1822     ax8.set_ylabel(
1823         "ACK Count"
1824     )
1825     ax8.grid(
1826         True,
1827         alpha=0.3
1828     )
1829     # =====
1830     # 9. RTO
1831     # =====
1832     ax9 = self.figure.add_subplot(
1833         5,
1834         2,
1835         9
1836     )
1837     if self.simulator.protocol == "TCP":
1838         ax9.plot(
1839             x,
1840             dataframe["rto_ms"],
1841             linewidth=2
1842         )
1843         ax9.set_title(
1844             "TCP Retransmission Timeout (RTO)",
1845             fontsize=11,
1846             fontweight="bold"
1847         )
1848         ax9.set_ylabel(
1849             "RTO (ms)"
1850         )
1851     else:
1852         ax9.text(
1853             0.5,
1854             0.5,
1855             "UDP does not use RTO",
1856             ha="center",
1857             va="center",

```

```

1858         transform=ax9.transAxes
1859     )
1860     ax9.set_title(
1861         "Retransmission Timeout"
1862     )
1863     ax9.set_xlabel(
1864         "Segment"
1865     )
1866     ax9.grid(
1867         True,
1868         alpha=0.3
1869     )
1870     # =====
1871     # 10. EMPTY AREA
1872     # =====
1873     ax10 = self.figure.add_subplot(
1874         5,
1875         2,
1876         10
1877     )
1878     ax10.axis(
1879         "off"
1880     )
1881     # =====
1882     # GLOBAL TITLE
1883     # =====
1884     self.figure.suptitle(
1885         f"{self.simulator.protocol} "
1886         "Transport Layer Simulation",
1887         fontsize=15,
1888         fontweight="bold"
1889     )
1890     # =====
1891     # LAYOUT
1892     # =====
1893     self.figure.tight_layout(
1894         rect=[
1895             0.03,
1896             0.02,
1897             0.98,
1898             0.96
1899         ],
1900         h_pad=2.5,
1901         w_pad=2.0
1902     )
1903     self.canvas.draw_idle()

```

```

1904 # =====
1905 # UPDATE PACKET TABLE
1906 # =====
1907 def update_packet_table(self):
1908     self.packet_store.clear()
1909     for segment in (
1910         self.simulator.segments
1911     ):
1912         self.packet_store.append(
1913             [
1914                 segment.segment_id,
1915                 segment.protocol,
1916                 segment.source_port,
1917                 segment.destination_port,
1918                 segment.sequence_number,
1919                 segment.status,
1920                 round(
1921                     segment.delay_ms,
1922                     2
1923                 ),
1924                 round(
1925                     segment.rtt_ms,
1926                     2
1927                 ),
1928                 segment.retransmissions
1929             ]
1930         )
1931 # =====
1932 # UPDATE METRICS
1933 # =====
1934 def update_metrics(self):
1935     metrics = (
1936         self.simulator.get_metrics()
1937     )
1938     # -----
1939     # GENERATED
1940     # -----
1941     self.metric_labels[
1942         "generated"
1943     ].set_markup(
1944         f"<b>{metrics['generated']}</b>"
1945     )
1946     # -----
1947     # DELIVERED
1948     # -----
1949     self.metric_labels[

```

```

1950         "delivered"
1951     ].set_markup(
1952         f"<b>{metrics['delivered']}</b>"
1953     )
1954     # -----
1955     # LOST
1956     # -----
1957     self.metric_labels[
1958         "lost"
1959     ].set_markup(
1960         f"<b>{metrics['lost']}</b>"
1961     )
1962     # -----
1963     # ACK
1964     # -----
1965     self.metric_labels[
1966         "acknowledged"
1967     ].set_markup(
1968         f"<b>{metrics['acknowledged']}</b>"
1969     )
1970     # -----
1971     # RETRANSMISSION
1972     # -----
1973     self.metric_labels[
1974         "retransmissions"
1975     ].set_markup(
1976         f"<b>{metrics['retransmissions']}</b>"
1977     )
1978     # -----
1979     # PDR
1980     # -----
1981     self.metric_labels[
1982         "pdr"
1983     ].set_markup(
1984         f"<b>{metrics['pdr']:.2f}%</b>"
1985     )
1986     # -----
1987     # LOSS RATE
1988     # -----
1989     self.metric_labels[
1990         "loss_rate"
1991     ].set_markup(
1992         f"<b>{metrics['loss_rate']:.2f}%</b>"
1993     )
1994     # -----
1995     # RTT

```

```

1996     # -----
1997     self.metric_labels[
1998         "average_rtt"
1999     ].set_markup(
2000         f"<b>{metrics['average_rtt']:.2f} ms</b>"
2001     )
2002     # -----
2003     # DELAY
2004     # -----
2005     self.metric_labels[
2006         "average_delay"
2007     ].set_markup(
2008         f"<b>{metrics['average_delay']:.2f} ms</b>"
2009     )
2010     # -----
2011     # THROUGHPUT
2012     # -----
2013     throughput_mbps = (
2014         metrics["throughput"]
2015         /
2016         1_000_000.0
2017     )
2018     self.metric_labels[
2019         "throughput"
2020     ].set_markup(
2021         f"<b>{throughput_mbps:.3f} Mbps</b>"
2022     )
2023     # =====
2024     # SIMULATION FINISHED
2025     # =====
2026     def simulation_finished(self):
2027         self.start_button.set_sensitive(
2028             True
2029         )
2030         self.update_packet_table()
2031         self.update_metrics()
2032         self.update_graph()
2033         metrics = (
2034             self.simulator.get_metrics()
2035         )
2036         protocol = (
2037             self.simulator.protocol
2038         )
2039         self.status.set_text(
2040             f"{protocol} simulation completed | "
2041             f"Generated={metrics['generated']} | "

```

```

2042         f"Delivered={metrics['delivered']} | "
2043         f"Lost={metrics['lost']} | "
2044         f"Retransmissions="
2045         f"{metrics['retransmissions']} | "
2046         f"PDR={metrics['pdr']:.2f}%"
2047     )
2048     return False
2049     # =====
2050     # STOP SIMULATION
2051     # =====
2052     def stop_simulation(
2053         self,
2054         button
2055     ):
2056         self.simulator.stop()
2057         self.status.set_text(
2058             "Simulation stopped."
2059         )
2060         self.start_button.set_sensitive(
2061             True
2062         )
2063     # =====
2064     # EXPORT CSV
2065     # =====
2066     def export_csv(
2067         self,
2068         button
2069     ):
2070         dataframe = (
2071             self.simulator.dataframe()
2072         )
2073         if dataframe.empty:
2074             self.error_dialog(
2075                 "No simulation data available."
2076             )
2077             return
2078
2079         dialog = Gtk.FileChooserDialog(
2080             title=
2081                 "Export Transport Layer CSV",
2082             parent=
2083                 self,
2084             action=
2085                 Gtk.FileChooserAction.SAVE
2086         )
2087         dialog.add_buttons(

```

```

2088         Gtk.STOCK_CANCEL,
2089         Gtk.ResponseType.CANCEL,
2090         Gtk.STOCK_SAVE,
2091         Gtk.ResponseType.OK
2092     )
2093     dialog.set_current_name(
2094         "transport_layer_results.csv"
2095     )
2096     response = dialog.run()
2097     if response == Gtk.ResponseType.OK:
2098         filename = (
2099             dialog.get_filename()
2100         )
2101         try:
2102             dataframe.to_csv(
2103                 filename,
2104                 index=False
2105             )
2106             self.status.set_text(
2107                 "CSV exported: "
2108                 +
2109                 filename
2110             )
2111         except Exception as error:
2112             self.error_dialog(
2113                 str(error)
2114             )
2115     dialog.destroy()
2116     # =====
2117     # ERROR DIALOG
2118     # =====
2119     def error_dialog(
2120         self,
2121         message
2122     ):
2123         dialog = Gtk.MessageDialog(
2124             parent=self,
2125             flags=
2126                 Gtk.DialogFlags.MODAL,
2127             message_type=
2128                 Gtk.MessageType.ERROR,
2129             buttons=
2130                 Gtk.ButtonsType.OK,
2131             text=
2132                 "Simulation Error"
2133         )

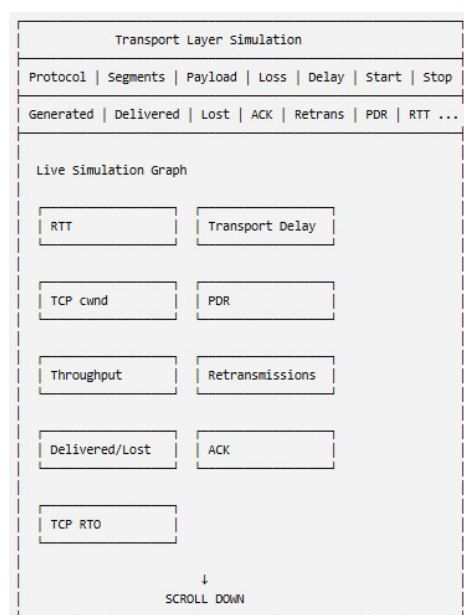
```

```

2134         dialog.format_secondary_text(
2135             message
2136         )
2137         dialog.run()
2138         dialog.destroy()
2139 # =====
2140 # MAIN
2141 # =====
2142 def main():
2143     window = TransportLayerGTK()
2144     window.connect(
2145         "destroy",
2146         Gtk.main_quit
2147     )
2148     window.show_all()
2149 # =====
2150 # MAXIMIZE WINDOW
2151 # =====
2152     window.maximize()
2153     Gtk.main()
2154 # =====
2155 # ENTRY POINT
2156 # =====
2157 if __name__ == "__main__":
2158     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

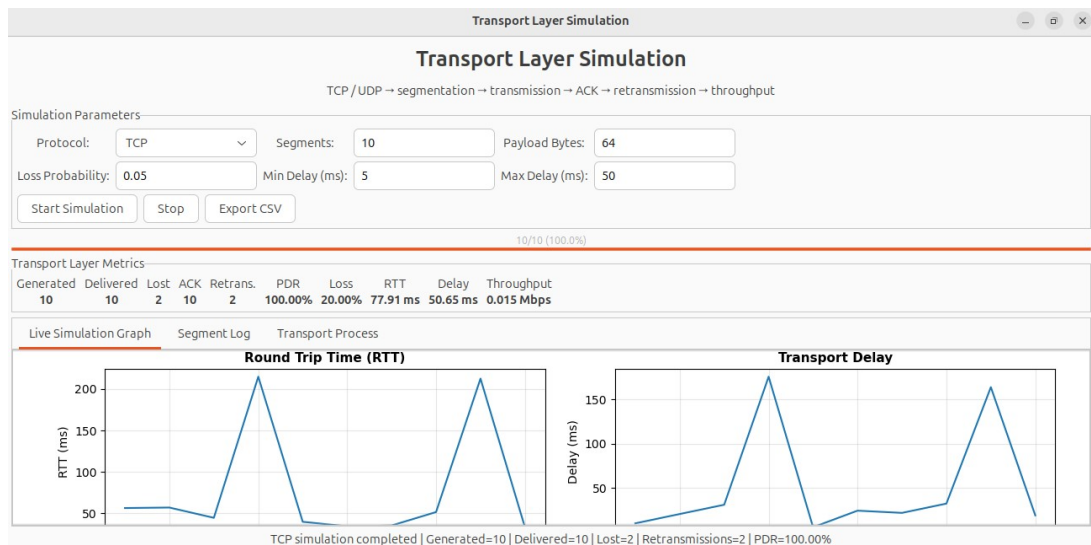


Gambar 7.1: Skema Transport Layer Simulator

Transport Layer.

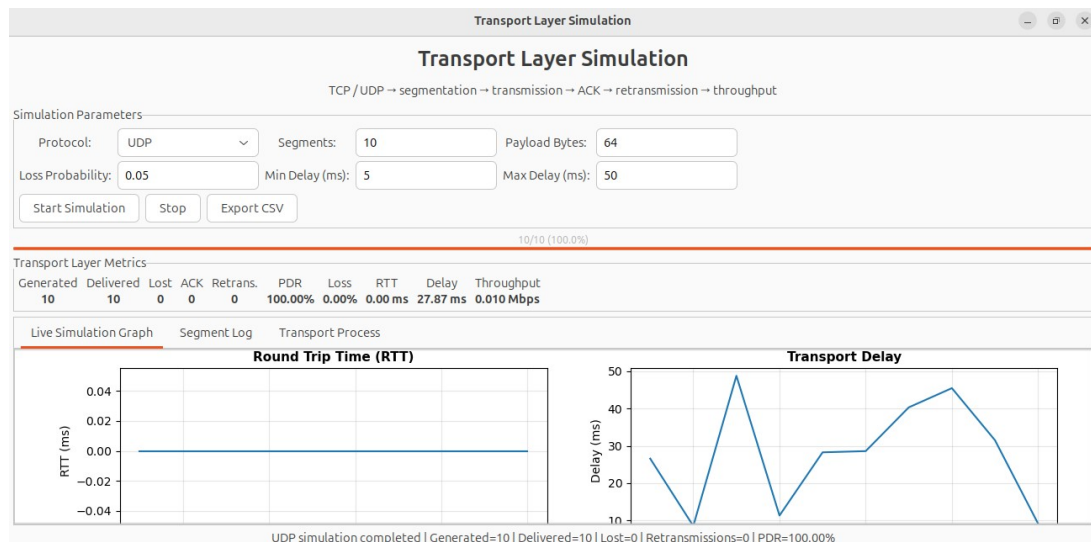
7.1.1 Tampilan Simulator Transport Layer

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

```

1  #!/usr/bin/env python3
2
3  """
4  =====
5  SESSION LAYER SIMULATION
6  =====
7
8  Python 3
9  GTK3 / PyGObject
10 Matplotlib
11 NumPy
12 Pandas
13
14 OSI SESSION LAYER SIMULATION
15
16 Main concepts modeled:
17
18     1. Session establishment
19     2. Session authentication
20     3. Session synchronization
21     4. Dialog control
22     5. Data exchange
23     6. Checkpoint creation
24     7. Keep-alive
25     8. Session timeout
26     9. Synchronization loss
27    10. Session recovery
28    11. Recovery from checkpoint
29    12. Session termination
30
31 Simulation metrics:
32
33     - Session state
34     - Session duration
35     - Response time
36     - Synchronization point

```

- 37 - Checkpoint count
- 38 - Synchronization loss
- 39 - Recovery attempts
- 40 - Successful recovery
- 41 - Session reliability
- 42 - Recovery rate
- 43 - Keep-alive count
- 44 - Timeout count
- 45 - Data exchanged
- 46 - Events processed

48 GUI:

- 49
- 50 - GTK3
- 51 - Resizable
- 52 - Maximizable
- 53 - Responsive parameter area
- 54 - Live Matplotlib graph
- 55 - Vertical graph scrolling
- 56 - Event log
- 57 - CSV export
- 58 - Process explanation
- 59

60 Run:

```

61
62     python3 final_session_layer_research_simulation.py
63
64 =====
65 """
66
67 import gi
68
69 gi.require_version("Gtk", "3.0")
70
71 from gi.repository import Gtk, GLib
72
73 import matplotlib
74
75 matplotlib.use("GTK3Agg")
76
77 from matplotlib.figure import Figure
78 from matplotlib.backends.backend_gtk3agg import FigureCanvasGTK3Agg
79
80 import numpy as np
81 import pandas as pd
82

```

```

83 import random
84 import time
85 import threading
86
87
88 # =====
89 # SESSION EVENT
90 # =====
91
92 class SessionEvent:
93
94     def __init__(
95         self,
96         event_id,
97         event_type,
98         state,
99         sequence_number,
100        checkpoint,
101        response_time_ms,
102        status,
103        data_bytes=0
104    ):
105
106        self.event_id = event_id
107
108        self.event_type = event_type
109
110        self.state = state
111
112        self.sequence_number = sequence_number
113
114        self.checkpoint = checkpoint
115
116        self.response_time_ms = response_time_ms
117
118        self.status = status
119
120        self.data_bytes = data_bytes
121
122        self.timestamp = time.time()
123
124
125 # =====
126 # SESSION STATE MACHINE
127 # =====
128

```

```

129 class SessionStateMachine:
130
131     CLOSED = "CLOSED"
132
133     ESTABLISHING = "ESTABLISHING"
134
135     AUTHENTICATING = "AUTHENTICATING"
136
137     ESTABLISHED = "ESTABLISHED"
138
139     SYNCHRONIZING = "SYNCHRONIZING"
140
141     DATA_TRANSFER = "DATA_TRANSFER"
142
143     RECOVERING = "RECOVERING"
144
145     TERMINATING = "TERMINATING"
146
147     TIMEOUT = "TIMEOUT"
148
149     RECOVERY_FAILED = "RECOVERY_FAILED"
150
151
152 # =====
153 # SESSION LAYER SIMULATOR
154 # =====
155
156 class SessionLayerSimulator:
157
158     def __init__(
159         self,
160         event_count=100,
161         checkpoint_interval=10,
162         synchronization_probability=0.02,
163         timeout_probability=0.01,
164         recovery_probability=0.90,
165         min_response_time=5.0,
166         max_response_time=50.0,
167         data_size=1024,
168         keepalive_interval=10
169     ):
170
171         self.event_count = event_count
172
173         self.checkpoint_interval = (
174             checkpoint_interval

```

```

175         )
176
177         self.synchronization_probability = (
178             synchronization_probability
179         )
180
181         self.timeout_probability = (
182             timeout_probability
183         )
184
185         self.recovery_probability = (
186             recovery_probability
187         )
188
189         self.min_response_time = (
190             min_response_time
191         )
192
193         self.max_response_time = (
194             max_response_time
195         )
196
197         self.data_size = data_size
198
199         self.keepalive_interval = (
200             keepalive_interval
201         )
202
203         # -----
204         # SESSION STATE
205         # -----
206
207         self.state = (
208             SessionStateMachine.CLOSED
209         )
210
211         self.previous_state = (
212             SessionStateMachine.CLOSED
213         )
214
215         self.session_id = None
216
217         self.sequence_number = 0
218
219         self.last_checkpoint = 0
220

```

```
221         self.checkpoint_count = 0
222
223         # -----
224         # EVENT STORAGE
225         # -----
226
227         self.events = []
228
229         self.statistics = []
230
231         # -----
232         # EXECUTION
233         # -----
234
235         self.running = False
236
237         self.start_time = None
238
239         self.end_time = None
240
241         # -----
242         # COUNTERS
243         # -----
244
245         self.generated_events = 0
246
247         self.processed_events = 0
248
249         self.successful_events = 0
250
251         self.failed_events = 0
252
253         self.synchronization_losses = 0
254
255         self.recovery_attempts = 0
256
257         self.successful_recoveries = 0
258
259         self.keepalive_count = 0
260
261         self.timeout_count = 0
262
263         self.total_data_bytes = 0
264
265         self.total_response_time = 0.0
266
```

```

267         self.session_establishment_time = 0.0
268
269         self.session_termination_time = 0.0
270
271         # -----
272         # INTERNAL FLAGS
273         # -----
274
275         self.session_established = False
276
277         self.authenticated = False
278
279         self.synchronized = False
280
281         # =====
282         # RESET
283         # =====
284
285     def reset(self):
286
287         self.state = (
288             SessionStateMachine.CLOSED
289         )
290
291         self.previous_state = (
292             SessionStateMachine.CLOSED
293         )
294
295         self.session_id = None
296
297         self.sequence_number = 0
298
299         self.last_checkpoint = 0
300
301         self.checkpoint_count = 0
302
303         self.events = []
304
305         self.statistics = []
306
307         self.running = False
308
309         self.start_time = None
310
311         self.end_time = None
312

```

```

313         self.generated_events = 0
314
315         self.processed_events = 0
316
317         self.successful_events = 0
318
319         self.failed_events = 0
320
321         self.synchronization_losses = 0
322
323         self.recovery_attempts = 0
324
325         self.successful_recoveries = 0
326
327         self.keepalive_count = 0
328
329         self.timeout_count = 0
330
331         self.total_data_bytes = 0
332
333         self.total_response_time = 0.0
334
335         self.session_establishment_time = 0.0
336
337         self.session_termination_time = 0.0
338
339         self.session_established = False
340
341         self.authenticated = False
342
343         self.synchronized = False
344
345         # =====
346         # GENERATE SESSION ID
347         # =====
348
349     def generate_session_id(self):
350
351         self.session_id = (
352
353             "SES-"
354             +
355             str(
356                 random.randint(
357                     100000,
358                     999999

```

```

359         )
360     )
361
362 )
363
364     return self.session_id
365
366     # =====
367     # CHANGE STATE
368     # =====
369
370     def change_state(
371         self,
372         new_state
373     ):
374
375         self.previous_state = self.state
376
377         self.state = new_state
378
379     # =====
380     # RESPONSE TIME
381     # =====
382
383     def random_response_time(self):
384
385         return random.uniform(
386             self.min_response_time,
387             self.max_response_time
388         )
389
390     # =====
391     # ESTABLISH SESSION
392     # =====
393
394     def establish_session(self):
395
396         start = time.time()
397
398         self.change_state(
399             SessionStateMachine.ESTABLISHING
400         )
401
402         self.generate_session_id()
403
404         response_1 = (

```

```
405         self.random_response_time()
406     )
407
408     self.change_state(
409         SessionStateMachine.AUTHENTICATING
410     )
411
412     response_2 = (
413         self.random_response_time()
414     )
415
416     self.authenticated = True
417
418     self.change_state(
419         SessionStateMachine.SYNCHRONIZING
420     )
421
422     response_3 = (
423         self.random_response_time()
424     )
425
426     self.synchronized = True
427
428     self.change_state(
429         SessionStateMachine.ESTABLISHED
430     )
431
432     self.session_established = True
433
434     self.session_establishment_time = (
435
436         time.time()
437         -
438         start
439
440     ) * 1000.0
441
442     self.session_establishment_time += (
443
444         response_1
445         +
446         response_2
447         +
448         response_3
449
450     )
```

```

451         return self.session_establishment_time
452
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         )

```

```

497
498 # =====
499 # SESSION RECOVERY
500 # =====
501
502 def recover_session(self):
503
504     self.recovery_attempts += 1
505
506     recovery_response = (
507         self.random_response_time()
508     )
509
510     success = (
511
512         random.random()
513         <
514         self.recovery_probability
515
516     )
517
518     if success:
519
520         self.successful_recoveries += 1
521
522         self.synchronized = True
523
524         self.change_state(
525             SessionStateMachine.ESTABLISHED
526         )
527
528         self.sequence_number = (
529             self.last_checkpoint
530         )
531
532         return (
533             True,
534             recovery_response
535         )
536
537     self.change_state(
538         SessionStateMachine.RECOVERY_FAILED
539     )
540
541     return (
542         False,

```

```

543         recovery_response
544     )
545
546     # =====
547     # SESSION TIMEOUT
548     # =====
549
550     def timeout(self):
551
552         self.timeout_count += 1
553
554         self.change_state(
555             SessionStateMachine.TIMEOUT
556         )
557
558         self.synchronized = False
559
560     # =====
561     # DATA EXCHANGE
562     # =====
563
564     def exchange_data(
565         self,
566         event_id
567     ):
568
569         self.change_state(
570             SessionStateMachine.DATA_TRANSFER
571         )
572
573         response = (
574             self.random_response_time()
575         )
576
577         self.sequence_number += 1
578
579         self.total_data_bytes += (
580             self.data_size
581         )
582
583         # -----
584         # CHECKPOINT
585         # -----
586
587         checkpoint_created = False
588

```

```

589         if (
590
591             self.sequence_number
592             %
593             self.checkpoint_interval
594
595             ==
596
597             0
598
599         ):
600
601             self.create_checkpoint()
602
603             checkpoint_created = True
604
605             # -----
606             # KEEP ALIVE
607             # -----
608
609             if (
610
611                 event_id
612                 %
613                 self.keepalive_interval
614
615                 ==
616
617                 0
618
619             ):
620
621                 self.keepalive()
622
623                 self.change_state(
624                     SessionStateMachine.ESTABLISHED
625                 )
626
627                 return (
628                     response,
629                     checkpoint_created
630                 )
631
632             # =====
633             # TERMINATE SESSION
634             # =====

```

```

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

```

```

681         # -----
682         # SESSION MUST EXIST
683         # -----
684
685         if not self.session_established:
686
687             self.failed_events += 1
688
689             return SessionEvent(
690
691                 event_id,
692
693                 "SESSION_NOT_ESTABLISHED",
694
695                 self.state,
696
697                 self.sequence_number,
698
699                 self.last_checkpoint,
700
701                 0.0,
702
703                 "FAILED"
704             )
705
706         # -----
707         # SYNCHRONIZATION LOSS
708         # -----
709
710
711         synchronization_lost = (
712
713             random.random()
714             <
715             self.synchronization_probability
716
717         )
718
719         if synchronization_lost:
720
721             self.synchronization_loss()
722
723             response = (
724                 self.random_response_time()
725             )
726

```

```

727         self.failed_events += 1
728
729         event = SessionEvent(
730
731             event_id,
732
733             "SYNCHRONIZATION_LOSS",
734
735             self.state,
736
737             self.sequence_number,
738
739             self.last_checkpoint,
740
741             response,
742
743             "RECOVERY_REQUIRED"
744
745         )
746
747         self.events.append(
748             event
749         )
750
751         return event
752
753         # -----
754         # TIMEOUT
755         # -----
756
757         timeout_event = (
758
759             random.random()
760             <
761             self.timeout_probability
762
763         )
764
765         if timeout_event:
766
767             self.timeout()
768
769             response = (
770                 self.random_response_time()
771             )
772

```

```

773         self.failed_events += 1
774
775         event = SessionEvent(
776
777             event_id,
778
779             "SESSION_TIMEOUT",
780
781             self.state,
782
783             self.sequence_number,
784
785             self.last_checkpoint,
786
787             response,
788
789             "TIMEOUT"
790         )
791
792
793         self.events.append(
794             event
795         )
796
797         return event
798
799         # -----
800         # DATA EXCHANGE
801         # -----
802
803         response, checkpoint_created = (
804             self.exchange_data(
805                 event_id
806             )
807         )
808
809         self.processed_events += 1
810
811         self.successful_events += 1
812
813         self.total_response_time += response
814
815         event_type = (
816             "DATA_EXCHANGE"
817         )
818

```

```

819         if checkpoint_created:
820
821             event_type = (
822                 "DATA_EXCHANGE_CHECKPOINT"
823             )
824
825         event = SessionEvent(
826
827             event_id,
828
829             event_type,
830
831             self.state,
832
833             self.sequence_number,
834
835             self.last_checkpoint,
836
837             response,
838
839             "SUCCESS",
840
841             self.data_size
842
843         )
844
845         self.events.append(
846             event
847         )
848
849         return event
850
851     # =====
852     # RUN SIMULATION
853     # =====
854
855     def run(
856         self,
857         progress_callback=None,
858         event_callback=None
859     ):
860
861         self.reset()
862
863         self.running = True
864

```

```

865         self.start_time = time.time()
866
867         # -----
868         # SESSION ESTABLISHMENT
869         # -----
870
871         establishment_time = (
872             self.establish_session()
873         )
874
875         establishment_event = SessionEvent(
876
877             0,
878
879             "SESSION_ESTABLISHED",
880
881             self.state,
882
883             self.sequence_number,
884
885             self.last_checkpoint,
886
887             establishment_time,
888
889             "SUCCESS"
890
891         )
892
893         self.events.append(
894             establishment_event
895         )
896
897         if event_callback:
898
899             event_callback(
900                 establishment_event
901             )
902
903         # -----
904         # EVENT PROCESSING
905         # -----
906
907         for event_id in range(
908             1,
909             self.event_count + 1
910         ):

```

```

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

```

```

957         recovery_time,
958
959         recovery_status
960
961     )
962
963     self.events.append(
964         recovery_event
965     )
966
967     if event_callback:
968
969         event_callback(
970             recovery_event
971         )
972
973     if event_callback:
974
975         event_callback(
976             event
977         )
978
979     # -----
980     # METRICS
981     # -----
982
983     elapsed = (
984
985         time.time()
986         -
987         self.start_time
988
989     )
990
991     total_events = (
992         self.generated_events
993     )
994
995     reliability = 0.0
996
997     if total_events > 0:
998
999         reliability = (
1000
1001             self.successful_events
1002             /

```

```
1003         total_events
1004
1005     ) * 100.0
1006
1007     recovery_rate = 0.0
1008
1009     if self.recovery_attempts > 0:
1010
1011         recovery_rate = (
1012
1013             self.successful_recoveries
1014             /
1015             self.recovery_attempts
1016
1017         ) * 100.0
1018
1019     average_response = 0.0
1020
1021     if self.processed_events > 0:
1022
1023         average_response = (
1024
1025             self.total_response_time
1026             /
1027             self.processed_events
1028
1029         )
1030
1031     self.statistics.append(
1032
1033         {
1034
1035             "event_id":
1036                 event.event_id,
1037
1038             "event_type":
1039                 event.event_type,
1040
1041             "session_id":
1042                 self.session_id,
1043
1044             "state":
1045                 self.state,
1046
1047             "sequence_number":
1048                 self.sequence_number,
```

```
1049
1050     "checkpoint":
1051         self.last_checkpoint,
1052
1053     "checkpoint_count":
1054         self.checkpoint_count,
1055
1056     "response_time_ms":
1057         event.response_time_ms,
1058
1059     "status":
1060         event.status,
1061
1062     "data_bytes":
1063         event.data_bytes,
1064
1065     "generated_events":
1066         self.generated_events,
1067
1068     "processed_events":
1069         self.processed_events,
1070
1071     "successful_events":
1072         self.successful_events,
1073
1074     "failed_events":
1075         self.failed_events,
1076
1077     "synchronization_losses":
1078         self.synchronization_losses,
1079
1080     "recovery_attempts":
1081         self.recovery_attempts,
1082
1083     "successful_recoveries":
1084         self.successful_recoveries,
1085
1086     "keepalive_count":
1087         self.keepalive_count,
1088
1089     "timeout_count":
1090         self.timeout_count,
1091
1092     "reliability":
1093         reliability,
1094
```

```

1095         "recovery_rate":
1096             recovery_rate,
1097
1098         "average_response_ms":
1099             average_response,
1100
1101         "elapsed_seconds":
1102             elapsed,
1103
1104         "total_data_bytes":
1105             self.total_data_bytes
1106
1107     }
1108
1109 )
1110
1111 # -----
1112 # GUI CALLBACK
1113 # -----
1114
1115 if progress_callback:
1116
1117     progress_callback(
1118
1119         self.generated_events,
1120
1121         self.event_count
1122
1123     )
1124
1125     time.sleep(
1126         0.03
1127     )
1128
1129 # -----
1130 # SESSION TERMINATION
1131 # -----
1132
1133 if self.running:
1134
1135     termination_time = (
1136         self.terminate_session()
1137     )
1138
1139     termination_event = SessionEvent(
1140

```

```

1141         self.event_count + 1,
1142
1143         "SESSION_TERMINATED",
1144
1145         self.state,
1146
1147         self.sequence_number,
1148
1149         self.last_checkpoint,
1150
1151         termination_time,
1152
1153         "SUCCESS"
1154     )
1155
1156     self.events.append(
1157         termination_event
1158     )
1159
1160     if event_callback:
1161
1162         event_callback(
1163             termination_event
1164         )
1165
1166     self.end_time = time.time()
1167
1168     self.running = False
1169
1170     # =====
1171     # STOP
1172     # =====
1173
1174     def stop(self):
1175
1176         self.running = False
1177
1178     # =====
1179     # DATAFRAME
1180     # =====
1181
1182     def dataframe(self):
1183
1184         return pd.DataFrame(
1185             self.statistics
1186         )

```

```

1187         )
1188
1189         # =====
1190         # METRICS
1191         # =====
1192
1193     def get_metrics(self):
1194
1195         elapsed = 0.0
1196
1197         if (
1198
1199             self.start_time is not None
1200
1201             and
1202
1203             self.end_time is not None
1204
1205         ):
1206
1207             elapsed = (
1208
1209                 self.end_time
1210                 -
1211                 self.start_time
1212
1213             )
1214
1215             reliability = 0.0
1216
1217             if self.generated_events > 0:
1218
1219                 reliability = (
1220
1221                     self.successful_events
1222                     /
1223                     self.generated_events
1224
1225                 ) * 100.0
1226
1227             recovery_rate = 0.0
1228
1229             if self.recovery_attempts > 0:
1230
1231                 recovery_rate = (
1232

```

```
1233         self.successful_recoveries
1234         /
1235         self.recovery_attempts
1236
1237     ) * 100.0
1238
1239     average_response = 0.0
1240
1241     if self.processed_events > 0:
1242
1243         average_response = (
1244
1245             self.total_response_time
1246             /
1247             self.processed_events
1248
1249         )
1250
1251     return {
1252
1253         "session_id":
1254             self.session_id,
1255
1256         "state":
1257             self.state,
1258
1259         "generated_events":
1260             self.generated_events,
1261
1262         "processed_events":
1263             self.processed_events,
1264
1265         "successful_events":
1266             self.successful_events,
1267
1268         "failed_events":
1269             self.failed_events,
1270
1271         "synchronization_losses":
1272             self.synchronization_losses,
1273
1274         "recovery_attempts":
1275             self.recovery_attempts,
1276
1277         "successful_recoveries":
1278             self.successful_recoveries,
```

```

1279
1280         "keepalive_count":
1281             self.keepalive_count,
1282
1283         "timeout_count":
1284             self.timeout_count,
1285
1286         "checkpoint_count":
1287             self.checkpoint_count,
1288
1289         "last_checkpoint":
1290             self.last_checkpoint,
1291
1292         "reliability":
1293             reliability,
1294
1295         "recovery_rate":
1296             recovery_rate,
1297
1298         "average_response":
1299             average_response,
1300
1301         "data_bytes":
1302             self.total_data_bytes,
1303
1304         "elapsed":
1305             elapsed,
1306
1307         "establishment":
1308             self.session_establishment_time,
1309
1310         "termination":
1311             self.session_termination_time
1312
1313     }
1314
1315
1316 # =====
1317 # GTK APPLICATION
1318 # =====
1319
1320 class SessionLayerGTK(Gtk.Window):
1321
1322     def __init__(self):
1323
1324         super().__init__(

```

```

1325         title="Session Layer Simulation"
1326     )
1327
1328     # =====
1329     # WINDOW
1330     # =====
1331
1332     self.set_default_size(
1333         1280,
1334         800
1335     )
1336
1337     self.set_size_request(
1338         900,
1339         600
1340     )
1341
1342     self.set_border_width(
1343         8
1344     )
1345
1346     self.set_resizable(
1347         True
1348     )
1349
1350     self.set_position(
1351         Gtk.WindowPosition.CENTER
1352     )
1353
1354     # =====
1355     # SIMULATOR
1356     # =====
1357
1358     self.simulator = (
1359         SessionLayerSimulator()
1360     )
1361
1362     self.worker_thread = None
1363
1364     self.build_ui()
1365
1366     # =====
1367     # BUILD UI
1368     # =====
1369
1370     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"

```

```

1417         )
1418
1419     )
1420
1421     main_box.pack_start(
1422
1423         subtitle,
1424
1425         False,
1426         False,
1427         2
1428
1429     )
1430
1431     # =====
1432     # PARAMETERS
1433     # =====
1434
1435     parameter_frame = Gtk.Frame(
1436
1437         label="Session Simulation Parameters"
1438
1439     )
1440
1441     parameter_grid = Gtk.Grid()
1442
1443     parameter_grid.set_border_width(
1444         6
1445     )
1446
1447     parameter_grid.set_row_spacing(
1448         5
1449     )
1450
1451     parameter_grid.set_column_spacing(
1452         7
1453     )
1454
1455     parameter_frame.add(
1456         parameter_grid
1457     )
1458
1459     main_box.pack_start(
1460
1461         parameter_frame,
1462

```

```

1463         False,
1464         False,
1465         0
1466
1467     )
1468
1469     # -----
1470     # EVENT COUNT
1471     # -----
1472
1473     parameter_grid.attach(
1474
1475         Gtk.Label(
1476             label="Events:"
1477         ),
1478
1479         0,
1480         0,
1481         1,
1482         1
1483
1484     )
1485
1486     self.event_entry = (
1487         Gtk.Entry()
1488     )
1489
1490     self.event_entry.set_text(
1491         "100"
1492     )
1493
1494     parameter_grid.attach(
1495
1496         self.event_entry,
1497
1498         1,
1499         0,
1500         1,
1501         1
1502
1503     )
1504
1505     # -----
1506     # CHECKPOINT
1507     # -----
1508

```

```

1509         parameter_grid.attach(
1510
1511             Gtk.Label(
1512                 label="Checkpoint Interval:"
1513             ),
1514
1515             2,
1516             0,
1517             1,
1518             1
1519
1520         )
1521
1522         self.checkpoint_entry = (
1523             Gtk.Entry()
1524         )
1525
1526         self.checkpoint_entry.set_text(
1527             "10"
1528         )
1529
1530         parameter_grid.attach(
1531
1532             self.checkpoint_entry,
1533
1534             3,
1535             0,
1536             1,
1537             1
1538
1539         )
1540
1541         # -----
1542         # SYNCHRONIZATION LOSS
1543         # -----
1544
1545         parameter_grid.attach(
1546
1547             Gtk.Label(
1548                 label="Sync Loss Probability:"
1549             ),
1550
1551             4,
1552             0,
1553             1,
1554             1

```

```

1555
1556     )
1557
1558     self.sync_loss_entry = (
1559         Gtk.Entry()
1560     )
1561
1562     self.sync_loss_entry.set_text(
1563         "0.02"
1564     )
1565
1566     parameter_grid.attach(
1567
1568         self.sync_loss_entry,
1569
1570         5,
1571         0,
1572         1,
1573         1
1574
1575     )
1576
1577     # -----
1578     # TIMEOUT
1579     # -----
1580
1581     parameter_grid.attach(
1582
1583         Gtk.Label(
1584             label="Timeout Probability:"
1585         ),
1586
1587         0,
1588         1,
1589         1,
1590         1
1591
1592     )
1593
1594     self.timeout_entry = (
1595         Gtk.Entry()
1596     )
1597
1598     self.timeout_entry.set_text(
1599         "0.01"
1600     )

```

```

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

```

```

1647         )
1648
1649         # -----
1650         # DATA SIZE
1651         # -----
1652
1653         parameter_grid.attach(
1654
1655             Gtk.Label(
1656                 label="Data Bytes/Event:"
1657             ),
1658
1659             4,
1660             1,
1661             1,
1662             1
1663
1664         )
1665
1666         self.data_size_entry = (
1667             Gtk.Entry()
1668         )
1669
1670         self.data_size_entry.set_text(
1671             "1024"
1672         )
1673
1674         parameter_grid.attach(
1675
1676             self.data_size_entry,
1677
1678             5,
1679             1,
1680             1,
1681             1
1682
1683         )
1684
1685         # =====
1686         # BUTTONS
1687         # =====
1688
1689         button_box = Gtk.Box(
1690
1691             orientation=
1692                 Gtk.Orientation.HORIZONTAL,

```

```
1693
1694         spacing=7
1695
1696     )
1697
1698     self.start_button = Gtk.Button(
1699
1700         label="Start Simulation"
1701
1702     )
1703
1704     self.start_button.connect(
1705
1706         "clicked",
1707
1708         self.start_simulation
1709
1710     )
1711
1712     button_box.pack_start(
1713
1714         self.start_button,
1715
1716         False,
1717         False,
1718         0
1719
1720     )
1721
1722     self.stop_button = Gtk.Button(
1723
1724         label="Stop"
1725
1726     )
1727
1728     self.stop_button.connect(
1729
1730         "clicked",
1731
1732         self.stop_simulation
1733
1734     )
1735
1736     button_box.pack_start(
1737
1738         self.stop_button,
```

```

1739
1740         False,
1741         False,
1742         0
1743
1744     )
1745
1746     export_button = Gtk.Button(
1747
1748         label="Export CSV"
1749
1750     )
1751
1752     export_button.connect(
1753
1754         "clicked",
1755
1756         self.export_csv
1757
1758     )
1759
1760     button_box.pack_start(
1761
1762         export_button,
1763
1764         False,
1765         False,
1766         0
1767
1768     )
1769
1770     parameter_grid.attach(
1771
1772         button_box,
1773
1774         0,
1775         2,
1776         6,
1777         1
1778
1779     )
1780
1781     # =====
1782     # PROGRESS
1783     # =====
1784

```

```
1785         self.progress = (  
1786             Gtk.ProgressBar()  
1787         )  
1788  
1789         self.progress.set_show_text(  
1790             True  
1791         )  
1792  
1793         main_box.pack_start(  
1794  
1795             self.progress,  
1796  
1797             False,  
1798             False,  
1799             0  
1800  
1801         )  
1802  
1803         # =====  
1804         # METRICS  
1805         # =====  
1806  
1807         metric_frame = Gtk.Frame(  
1808  
1809             label="Session Layer Metrics"  
1810  
1811         )  
1812  
1813         metric_grid = Gtk.Grid()  
1814  
1815         metric_grid.set_border_width(  
1816             5  
1817         )  
1818  
1819         metric_grid.set_column_spacing(  
1820             9  
1821         )  
1822  
1823         metric_grid.set_row_spacing(  
1824             4  
1825         )  
1826  
1827         metric_frame.add(  
1828             metric_grid  
1829         )  
1830
```

```

1831     main_box.pack_start(
1832
1833         metric_frame,
1834
1835         False,
1836         False,
1837         0
1838
1839     )
1840
1841     self.metric_labels = {}
1842
1843     metrics = [
1844
1845         ("Events", "generated_events"),
1846
1847         ("Processed", "processed_events"),
1848
1849         ("Success", "successful_events"),
1850
1851         ("Failed", "failed_events"),
1852
1853         ("Sync Loss", "synchronization_losses"),
1854
1855         ("Recovery", "recovery_attempts"),
1856
1857         ("Recovered", "successful_recoveries"),
1858
1859         ("Checkpoint", "checkpoint_count"),
1860
1861         ("Keepalive", "keepalive_count"),
1862
1863         ("Timeout", "timeout_count"),
1864
1865         ("Reliability", "reliability"),
1866
1867         ("Recovery Rate", "recovery_rate")
1868
1869     ]
1870
1871     for index, (
1872         caption,
1873         key
1874     ) in enumerate(metrics):
1875
1876         metric_box = Gtk.Box(

```

```
1877
1878         orientation=
1879             Gtk.Orientation.VERTICAL,
1880
1881         spacing=1
1882
1883     )
1884
1885     caption_label = Gtk.Label(
1886
1887         label=caption
1888
1889     )
1890
1891     value_label = Gtk.Label(
1892
1893         label="0"
1894
1895     )
1896
1897     value_label.set_markup(
1898
1899         "<b>0</b>"
1900
1901     )
1902
1903     metric_box.pack_start(
1904
1905         caption_label,
1906
1907         False,
1908         False,
1909         0
1910
1911     )
1912
1913     metric_box.pack_start(
1914
1915         value_label,
1916
1917         False,
1918         False,
1919         0
1920
1921     )
1922
```

```

1923         metric_grid.attach(
1924
1925             metric_box,
1926
1927             index,
1928             0,
1929             1,
1930             1
1931
1932         )
1933
1934         self.metric_labels[key] = (
1935             value_label
1936         )
1937
1938         # =====
1939         # NOTEBOOK
1940         # =====
1941
1942         notebook = Gtk.Notebook()
1943
1944         notebook.set_vexpand(
1945             True
1946         )
1947
1948         notebook.set_hexpand(
1949             True
1950         )
1951
1952         main_box.pack_start(
1953
1954             notebook,
1955
1956             True,
1957             True,
1958             0
1959
1960         )
1961
1962         # =====
1963         # LIVE GRAPH TAB
1964         # =====
1965
1966         graph_box = Gtk.Box(
1967
1968             orientation=

```

```

1969         Gtk.Orientation.VERTICAL,
1970
1971         spacing=4
1972
1973     )
1974
1975     notebook.append_page(
1976
1977         graph_box,
1978
1979         Gtk.Label(
1980
1981             label="Live Simulation Graph"
1982
1983         )
1984
1985     )
1986
1987     # -----
1988     # SCROLL WINDOW
1989     # -----
1990
1991     self.graph_scroll = (
1992         Gtk.ScrolledWindow()
1993     )
1994
1995     self.graph_scroll.set_policy(
1996
1997         Gtk.PolicyType.NEVER,
1998
1999         Gtk.PolicyType.AUTOMATIC
2000
2001     )
2002
2003     self.graph_scroll.set_shadow_type(
2004
2005         Gtk.ShadowType.IN
2006
2007     )
2008
2009     self.graph_scroll.set_vexpand(
2010         True
2011     )
2012
2013     self.graph_scroll.set_hexpand(
2014         True

```

```

2015         )
2016
2017     graph_box.pack_start(
2018
2019         self.graph_scroll,
2020
2021         True,
2022         True,
2023         0
2024
2025     )
2026
2027     # -----
2028     # GRAPH CONTAINER
2029     # -----
2030
2031     self.graph_container = Gtk.Box(
2032
2033         orientation=
2034             Gtk.Orientation.VERTICAL
2035
2036     )
2037
2038     self.graph_scroll.add(
2039
2040         self.graph_container
2041
2042     )
2043
2044     # -----
2045     # MATPLOTLIB
2046     # -----
2047
2048     self.figure = Figure(
2049
2050         figsize=(13, 16),
2051
2052         dpi=100
2053
2054     )
2055
2056     self.canvas = (
2057
2058         FigureCanvasGTK3Agg(
2059
2060             self.figure

```

```

2061         )
2062     )
2063
2064 )
2065
2066 # Intentionally taller than viewport.
2067 # Gtk.ScrolledWindow provides vertical scrolling.
2068
2069 self.canvas.set_size_request(
2070
2071     1100,
2072
2073     1600
2074
2075 )
2076
2077 self.graph_container.pack_start(
2078
2079     self.canvas,
2080
2081     False,
2082     False,
2083     0
2084
2085 )
2086
2087 # =====
2088 # EVENT LOG TAB
2089 # =====
2090
2091 event_box = Gtk.Box(
2092
2093     orientation=
2094         Gtk.Orientation.VERTICAL
2095
2096 )
2097
2098 notebook.append_page(
2099
2100     event_box,
2101
2102     Gtk.Label(
2103
2104         label="Session Event Log"
2105
2106     )

```

```
2107
2108     )
2109
2110     self.event_store = Gtk.ListStore(
2111
2112         int,
2113         str,
2114         str,
2115         int,
2116         int,
2117         float,
2118         str,
2119         int
2120
2121     )
2122
2123     event_tree = Gtk.TreeView(
2124
2125         model=self.event_store
2126
2127     )
2128
2129     event_columns = [
2130
2131         ("ID", 0),
2132
2133         ("Event", 1),
2134
2135         ("State", 2),
2136
2137         ("Sequence", 3),
2138
2139         ("Checkpoint", 4),
2140
2141         ("Response ms", 5),
2142
2143         ("Status", 6),
2144
2145         ("Data Bytes", 7)
2146
2147     ]
2148
2149     for title, index in event_columns:
2150
2151         renderer = (
2152             Gtk.CellRendererText()
```

```
2153         )
2154
2155         column = (
2156
2157             Gtk.TreeViewColumn(
2158
2159                 title,
2160
2161                 renderer,
2162
2163                 text=index
2164
2165             )
2166
2167         )
2168
2169         event_tree.append_column(
2170             column
2171         )
2172
2173         event_scroll = (
2174             Gtk.ScrolledWindow()
2175         )
2176
2177         event_scroll.set_policy(
2178
2179             Gtk.PolicyType.AUTOMATIC,
2180
2181             Gtk.PolicyType.AUTOMATIC
2182
2183         )
2184
2185         event_scroll.add(
2186             event_tree
2187         )
2188
2189         event_box.pack_start(
2190
2191             event_scroll,
2192
2193             True,
2194             True,
2195             0
2196
2197         )
2198
```

```

2199 # =====
2200 # SESSION PROCESS TAB
2201 # =====
2202
2203 process_box = Gtk.Box(
2204
2205     orientation=
2206         Gtk.Orientation.VERTICAL,
2207
2208     spacing=10
2209
2210 )
2211
2212 notebook.append_page(
2213
2214     process_box,
2215
2216     Gtk.Label(
2217
2218         label="Session Process"
2219
2220     )
2221
2222 )
2223
2224 process_scroll = (
2225     Gtk.ScrolledWindow()
2226 )
2227
2228 process_scroll.set_policy(
2229
2230     Gtk.PolicyType.NEVER,
2231
2232     Gtk.PolicyType.AUTOMATIC
2233
2234 )
2235
2236 process_box.pack_start(
2237
2238     process_scroll,
2239
2240     True,
2241     True,
2242     0
2243
2244 )

```

```

2245
2246     process_label = Gtk.Label()
2247
2248     process_label.set_markup(
2249
2250         "<b>OSI SESSION LAYER PROCESS</b>\n\n"
2251
2252         "Application Layer\n"
2253         "    \n"
2254         "Session Request\n"
2255         "    \n"
2256         "Session Establishment\n"
2257         "    \n"
2258         "Session ID Assignment\n"
2259         "    \n"
2260         "Authentication\n"
2261         "    \n"
2262         "Synchronization\n"
2263         "    \n"
2264         "Session Established\n"
2265         "    \n"
2266         "Dialog Control\n"
2267         "    \n"
2268         "Data Exchange\n"
2269         "    \n"
2270         "Checkpoint Creation\n"
2271         "    \n"
2272         "Keep-Alive\n"
2273         "    \n"
2274         "Synchronization Monitoring\n"
2275         "    \n"
2276         "Synchronization Loss?\n"
2277         "    \n"
2278         "Session Recovery\n"
2279         "    \n"
2280         "Restore from Checkpoint\n"
2281         "    \n"
2282         "Resume Data Exchange\n"
2283         "    \n"
2284         "Session Termination\n"
2285         "    \n"
2286         "CLOSED\n\n"
2287
2288         "<b>SESSION LAYER FUNCTIONS MODELED</b>\n\n"
2289
2290         "    Session establishment\n"

```

```

2291         "    Session maintenance\n"
2292         "    Session termination\n"
2293         "    Dialog control\n"
2294         "    Synchronization\n"
2295         "    Checkpointing\n"
2296         "    Session recovery\n"
2297         "    Keep-alive\n"
2298         "    Timeout detection\n"
2299         "    State management\n\n"
2300
2301         "<b>SIMULATION METRICS</b>\n\n"
2302
2303         "    Session reliability\n"
2304         "    Recovery rate\n"
2305         "    Synchronization losses\n"
2306         "    Recovery attempts\n"
2307         "    Checkpoint count\n"
2308         "    Keep-alive count\n"
2309         "    Timeout count\n"
2310         "    Response time\n"
2311         "    Session duration\n"
2312         "    Data exchanged"
2313
2314     )
2315
2316     process_label.set_xalign(
2317         0
2318     )
2319
2320     process_label.set_yalign(
2321         0
2322     )
2323
2324     process_label.set_margin_start(
2325         20
2326     )
2327
2328     process_label.set_margin_top(
2329         20
2330     )
2331
2332     process_scroll.add(
2333         process_label
2334     )
2335
2336     # =====

```

```

2337     # STATUS
2338     # =====
2339
2340     self.status = Gtk.Label(
2341
2342         label="Ready."
2343
2344     )
2345
2346     main_box.pack_start(
2347
2348         self.status,
2349
2350         False,
2351         False,
2352         0
2353
2354     )
2355
2356     # =====
2357     # GET PARAMETERS
2358     # =====
2359
2360     def get_parameters(self):
2361
2362         try:
2363
2364             event_count = int(
2365
2366                 self.event_entry
2367                 .get_text()
2368
2369             )
2370
2371             checkpoint_interval = int(
2372
2373                 self.checkpoint_entry
2374                 .get_text()
2375
2376             )
2377
2378             sync_loss_probability = float(
2379
2380                 self.sync_loss_entry
2381                 .get_text()
2382

```

```

2383         )
2384
2385         timeout_probability = float(
2386
2387             self.timeout_entry
2388             .get_text()
2389
2390         )
2391
2392         recovery_probability = float(
2393
2394             self.recovery_entry
2395             .get_text()
2396
2397         )
2398
2399         data_size = int(
2400
2401             self.data_size_entry
2402             .get_text()
2403
2404         )
2405
2406         # -----
2407         # VALIDATION
2408         # -----
2409
2410         if event_count <= 0:
2411
2412             raise ValueError(
2413
2414                 "Event count must be greater than zero."
2415
2416             )
2417
2418         if checkpoint_interval <= 0:
2419
2420             raise ValueError(
2421
2422                 "Checkpoint interval must be greater than zero."
2423
2424             )
2425
2426         if not (
2427
2428             0.0

```

```
2429         <=
2430         sync_loss_probability
2431         <=
2432         1.0
2433
2434     ):
2435
2436         raise ValueError(
2437
2438             "Synchronization loss probability "
2439             "must be between 0 and 1."
2440
2441         )
2442
2443     if not (
2444
2445         0.0
2446         <=
2447         timeout_probability
2448         <=
2449         1.0
2450
2451     ):
2452
2453         raise ValueError(
2454
2455             "Timeout probability "
2456             "must be between 0 and 1."
2457
2458         )
2459
2460     if not (
2461
2462         0.0
2463         <=
2464         recovery_probability
2465         <=
2466         1.0
2467
2468     ):
2469
2470         raise ValueError(
2471
2472             "Recovery probability "
2473             "must be between 0 and 1."
2474
```

```

2475         )
2476
2477     if data_size <= 0:
2478
2479         raise ValueError(
2480
2481             "Data size must be greater than zero."
2482
2483         )
2484
2485     return (
2486
2487         event_count,
2488
2489         checkpoint_interval,
2490
2491         sync_loss_probability,
2492
2493         timeout_probability,
2494
2495         recovery_probability,
2496
2497         data_size
2498
2499     )
2500
2501 except ValueError as error:
2502
2503     self.error_dialog(
2504         str(error)
2505     )
2506
2507     return None
2508
2509 # =====
2510 # START
2511 # =====
2512
2513 def start_simulation(
2514     self,
2515     button
2516 ):
2517
2518     parameters = (
2519         self.get_parameters()
2520     )

```

```

2521
2522     if parameters is None:
2523
2524         return
2525
2526     (
2527
2528         event_count,
2529
2530         checkpoint_interval,
2531
2532         sync_loss_probability,
2533
2534         timeout_probability,
2535
2536         recovery_probability,
2537
2538         data_size
2539
2540     ) = parameters
2541
2542     # -----
2543     # CREATE SIMULATOR
2544     # -----
2545
2546     self.simulator = (
2547
2548         SessionLayerSimulator(
2549
2550             event_count=
2551                 event_count,
2552
2553             checkpoint_interval=
2554                 checkpoint_interval,
2555
2556             synchronization_probability=
2557                 sync_loss_probability,
2558
2559             timeout_probability=
2560                 timeout_probability,
2561
2562             recovery_probability=
2563                 recovery_probability,
2564
2565             data_size=
2566                 data_size

```

```

2567         )
2568     )
2569
2570 )
2571
2572 # -----
2573 # RESET GUI
2574 # -----
2575
2576 self.event_store.clear()
2577
2578 self.figure.clear()
2579
2580 self.canvas.draw()
2581
2582 self.progress.set_fraction(
2583     0.0
2584 )
2585
2586 self.progress.set_text(
2587     "0%"
2588 )
2589
2590 self.start_button.set_sensitive(
2591     False
2592 )
2593
2594 self.status.set_text(
2595     "Session simulation running..."
2596 )
2597
2598 )
2599
2600 # -----
2601 # WORKER
2602 # -----
2603
2604 self.worker_thread = (
2605     threading.Thread(
2606         target=
2607             self.run_worker,
2608         daemon=True
2609     )
2610 )
2611
2612

```

```

2613         )
2614
2615     )
2616
2617     self.worker_thread.start()
2618
2619     # =====
2620     # WORKER
2621     # =====
2622
2623     def run_worker(self):
2624
2625         self.simulator.run(
2626
2627             progress_callback=
2628                 self.progress_callback,
2629
2630             event_callback=
2631                 self.event_callback
2632
2633         )
2634
2635         GLib.idle_add(
2636
2637             self.simulation_finished
2638
2639         )
2640
2641     # =====
2642     # EVENT CALLBACK
2643     # =====
2644
2645     def event_callback(
2646         self,
2647         event
2648     ):
2649
2650         GLib.idle_add(
2651
2652             self.add_event_to_table,
2653
2654             event
2655
2656         )
2657
2658         GLib.idle_add(

```

```

2659
2660         self.update_live_view
2661
2662     )
2663
2664     # =====
2665     # ADD EVENT
2666     # =====
2667
2668     def add_event_to_table(
2669         self,
2670         event
2671     ):
2672
2673         self.event_store.append(
2674
2675             [
2676
2677                 event.event_id,
2678
2679                 event.event_type,
2680
2681                 event.state,
2682
2683                 event.sequence_number,
2684
2685                 event.checkpoint,
2686
2687                 round(
2688                     event.response_time_ms,
2689                     2
2690                 ),
2691
2692                 event.status,
2693
2694                 event.data_bytes
2695
2696             ]
2697
2698         )
2699
2700         return False
2701
2702     # =====
2703     # PROGRESS CALLBACK
2704     # =====

```

```

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

```

```

2751         self.status.set_text(
2752
2753             f"Processing session event "
2754             f"{current}/{total}"
2755
2756         )
2757
2758         self.update_live_view()
2759
2760         return False
2761
2762     # =====
2763     # UPDATE LIVE VIEW
2764     # =====
2765
2766     def update_live_view(self):
2767
2768         dataframe = (
2769             self.simulator.dataframe()
2770         )
2771
2772         if dataframe.empty:
2773
2774             self.update_metrics()
2775
2776             return False
2777
2778             self.update_metrics()
2779
2780             self.update_graph()
2781
2782             return False
2783
2784     # =====
2785     # UPDATE METRICS
2786     # =====
2787
2788     def update_metrics(self):
2789
2790         metrics = (
2791             self.simulator.get_metrics()
2792         )
2793
2794         self.metric_labels[
2795             "generated_events"
2796         ].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[
```

```

2843         "successful_recoveries"
2844     ].set_markup(
2845
2846         f"<b>{metrics['successful_recoveries']}</b>"
2847
2848     )
2849
2850     self.metric_labels[
2851         "checkpoint_count"
2852     ].set_markup(
2853
2854         f"<b>{metrics['checkpoint_count']}</b>"
2855
2856     )
2857
2858     self.metric_labels[
2859         "keepalive_count"
2860     ].set_markup(
2861
2862         f"<b>{metrics['keepalive_count']}</b>"
2863
2864     )
2865
2866     self.metric_labels[
2867         "timeout_count"
2868     ].set_markup(
2869
2870         f"<b>{metrics['timeout_count']}</b>"
2871
2872     )
2873
2874     self.metric_labels[
2875         "reliability"
2876     ].set_markup(
2877
2878         f"<b>{metrics['reliability']:.2f}%</b>"
2879
2880     )
2881
2882     self.metric_labels[
2883         "recovery_rate"
2884     ].set_markup(
2885
2886         f"<b>{metrics['recovery_rate']:.2f}%</b>"
2887
2888     )

```

```

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(

```

```

2935
2936         "Session Response Time",
2937
2938         fontsize=11,
2939
2940         fontweight="bold"
2941
2942     )
2943
2944     ax1.set_xlabel(
2945         "Event"
2946     )
2947
2948     ax1.set_ylabel(
2949         "Response Time (ms)"
2950     )
2951
2952     ax1.grid(
2953         True,
2954         alpha=0.3
2955     )
2956
2957     # =====
2958     # 2. SEQUENCE NUMBER
2959     # =====
2960
2961     ax2 = self.figure.add_subplot(
2962         5,
2963         2,
2964         2
2965     )
2966
2967     ax2.plot(
2968
2969         x,
2970
2971         dataframe["sequence_number"],
2972
2973         linewidth=2
2974
2975     )
2976
2977     ax2.set_title(
2978
2979         "Session Sequence Progression",
2980

```

```

2981         fontsize=11,
2982
2983         fontweight="bold"
2984
2985     )
2986
2987     ax2.set_xlabel(
2988         "Event"
2989     )
2990
2991     ax2.set_ylabel(
2992         "Sequence"
2993     )
2994
2995     ax2.grid(
2996         True,
2997         alpha=0.3
2998     )
2999
3000     # =====
3001     # 3. CHECKPOINT
3002     # =====
3003
3004     ax3 = self.figure.add_subplot(
3005         5,
3006         2,
3007         3
3008     )
3009
3010     ax3.plot(
3011
3012         x,
3013
3014         dataframe["checkpoint"],
3015
3016         linewidth=2
3017
3018     )
3019
3020     ax3.set_title(
3021
3022         "Session Synchronization Checkpoint",
3023
3024         fontsize=11,
3025
3026         fontweight="bold"

```

```

3027
3028     )
3029
3030     ax3.set_xlabel(
3031         "Event"
3032     )
3033
3034     ax3.set_ylabel(
3035         "Checkpoint"
3036     )
3037
3038     ax3.grid(
3039         True,
3040         alpha=0.3
3041     )
3042
3043     # =====
3044     # 4. CHECKPOINT COUNT
3045     # =====
3046
3047     ax4 = self.figure.add_subplot(
3048         5,
3049         2,
3050         4
3051     )
3052
3053     ax4.plot(
3054
3055         x,
3056
3057         dataframe["checkpoint_count"],
3058
3059         linewidth=2
3060
3061     )
3062
3063     ax4.set_title(
3064
3065         "Cumulative Checkpoints",
3066
3067         fontsize=11,
3068
3069         fontweight="bold"
3070
3071     )
3072

```

```

3073     ax4.set_xlabel(
3074         "Event"
3075     )
3076
3077     ax4.set_ylabel(
3078         "Checkpoint Count"
3079     )
3080
3081     ax4.grid(
3082         True,
3083         alpha=0.3
3084     )
3085
3086     # =====
3087     # 5. SESSION RELIABILITY
3088     # =====
3089
3090     ax5 = self.figure.add_subplot(
3091         5,
3092         2,
3093         5
3094     )
3095
3096     ax5.plot(
3097
3098         x,
3099
3100         dataframe["reliability"],
3101
3102         linewidth=2
3103
3104     )
3105
3106     ax5.set_title(
3107
3108         "Session Reliability",
3109
3110         fontsize=11,
3111
3112         fontweight="bold"
3113
3114     )
3115
3116     ax5.set_xlabel(
3117         "Event"
3118     )

```

```

3119
3120     ax5.set_ylabel(
3121         "Reliability (%)"
3122     )
3123
3124     ax5.set_ylim(
3125         0,
3126         105
3127     )
3128
3129     ax5.grid(
3130         True,
3131         alpha=0.3
3132     )
3133
3134     # =====
3135     # 6. RECOVERY RATE
3136     # =====
3137
3138     ax6 = self.figure.add_subplot(
3139         5,
3140         2,
3141         6
3142     )
3143
3144     ax6.plot(
3145
3146         x,
3147
3148         dataframe["recovery_rate"],
3149
3150         linewidth=2
3151
3152     )
3153
3154     ax6.set_title(
3155
3156         "Session Recovery Rate",
3157
3158         fontsize=11,
3159
3160         fontweight="bold"
3161
3162     )
3163
3164     ax6.set_xlabel(

```

```

3165         "Event"
3166     )
3167
3168     ax6.set_ylabel(
3169         "Recovery Rate (%)"
3170     )
3171
3172     ax6.set_ylim(
3173         0,
3174         105
3175     )
3176
3177     ax6.grid(
3178         True,
3179         alpha=0.3
3180     )
3181
3182     # =====
3183     # 7. SYNCHRONIZATION LOSS
3184     # =====
3185
3186     ax7 = self.figure.add_subplot(
3187         5,
3188         2,
3189         7
3190     )
3191
3192     ax7.plot(
3193
3194         x,
3195
3196         dataframe[
3197             "synchronization_losses"
3198         ],
3199
3200         linewidth=2
3201
3202     )
3203
3204     ax7.set_title(
3205
3206         "Cumulative Synchronization Loss",
3207
3208         fontsize=11,
3209
3210         fontweight="bold"

```

```

3211
3212     )
3213
3214     ax7.set_xlabel(
3215         "Event"
3216     )
3217
3218     ax7.set_ylabel(
3219         "Loss Count"
3220     )
3221
3222     ax7.grid(
3223         True,
3224         alpha=0.3
3225     )
3226
3227     # =====
3228     # 8. RECOVERY ATTEMPTS
3229     # =====
3230
3231     ax8 = self.figure.add_subplot(
3232         5,
3233         2,
3234         8
3235     )
3236
3237     ax8.plot(
3238
3239         x,
3240
3241         dataframe[
3242             "recovery_attempts"
3243         ],
3244
3245         linewidth=2,
3246
3247         label="Attempts"
3248     )
3249
3250
3251     ax8.plot(
3252
3253         x,
3254
3255         dataframe[
3256             "successful_recoveries"

```

```

3257         ],
3258
3259         linewidth=2,
3260
3261         label="Successful"
3262     )
3263
3264
3265     ax8.set_title(
3266
3267         "Session Recovery",
3268
3269         fontsize=11,
3270
3271         fontweight="bold"
3272     )
3273
3274
3275     ax8.set_xlabel(
3276         "Event"
3277     )
3278
3279     ax8.set_ylabel(
3280         "Count"
3281     )
3282
3283     ax8.legend(
3284         fontsize=8
3285     )
3286
3287     ax8.grid(
3288         True,
3289         alpha=0.3
3290     )
3291
3292     # =====
3293     # 9. KEEP-ALIVE / TIMEOUT
3294     # =====
3295
3296     ax9 = self.figure.add_subplot(
3297         5,
3298         2,
3299         9
3300     )
3301
3302     ax9.plot(

```

```
3303
3304         x,
3305
3306         dataframe[
3307             "keepalive_count"
3308         ],
3309
3310         linewidth=2,
3311
3312         label="Keepalive"
3313     )
3314
3315
3316     ax9.plot(
3317
3318         x,
3319
3320         dataframe[
3321             "timeout_count"
3322         ],
3323
3324         linewidth=2,
3325
3326         label="Timeout"
3327     )
3328
3329
3330     ax9.set_title(
3331
3332         "Keep-Alive and Timeout",
3333
3334         fontsize=11,
3335
3336         fontweight="bold"
3337     )
3338
3339
3340     ax9.set_xlabel(
3341         "Event"
3342     )
3343
3344     ax9.set_ylabel(
3345         "Count"
3346     )
3347
3348     ax9.legend(
```

```

3349         fontsize=8
3350     )
3351
3352     ax9.grid(
3353         True,
3354         alpha=0.3
3355     )
3356
3357     # =====
3358     # 10. DATA EXCHANGE
3359     # =====
3360
3361     ax10 = self.figure.add_subplot(
3362         5,
3363         2,
3364         10
3365     )
3366
3367     data_kb = (
3368
3369         dataframe[
3370             "total_data_bytes"
3371         ].to_numpy()
3372
3373         /
3374
3375         1024.0
3376
3377     )
3378
3379     ax10.plot(
3380
3381         x,
3382
3383         data_kb,
3384
3385         linewidth=2
3386
3387     )
3388
3389     ax10.set_title(
3390
3391         "Cumulative Session Data",
3392
3393         fontsize=11,
3394

```

```

3395         fontweight="bold"
3396
3397     )
3398
3399     ax10.set_xlabel(
3400         "Event"
3401     )
3402
3403     ax10.set_ylabel(
3404         "Data (KB)"
3405     )
3406
3407     ax10.grid(
3408         True,
3409         alpha=0.3
3410     )
3411
3412     # =====
3413     # GLOBAL TITLE
3414     # =====
3415
3416     self.figure.suptitle(
3417
3418         "OSI Session Layer Simulation",
3419
3420         fontsize=15,
3421
3422         fontweight="bold"
3423
3424     )
3425
3426     # =====
3427     # LAYOUT
3428     # =====
3429
3430     self.figure.tight_layout(
3431
3432         rect=[
3433
3434             0.03,
3435             0.02,
3436             0.98,
3437             0.96
3438
3439         ],
3440

```

```

3441         h_pad=2.5,
3442
3443         w_pad=2.0
3444
3445     )
3446
3447     self.canvas.draw_idle()
3448
3449     # =====
3450     # SIMULATION FINISHED
3451     # =====
3452
3453     def simulation_finished(self):
3454
3455         self.start_button.set_sensitive(
3456             True
3457         )
3458
3459         self.update_metrics()
3460
3461         self.update_graph()
3462
3463         metrics = (
3464             self.simulator.get_metrics()
3465         )
3466
3467         self.status.set_text(
3468
3469             "Session simulation completed | "
3470
3471             f"Session={metrics['session_id']} | "
3472
3473             f"Events={metrics['generated_events']} | "
3474
3475             f"Reliability="
3476             f"{metrics['reliability']:.2f}% | "
3477
3478             f"Recovery="
3479             f"{metrics['recovery_rate']:.2f}% | "
3480
3481             f"Checkpoints="
3482             f"{metrics['checkpoint_count']}"
3483
3484         )
3485
3486         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(

```

```
3533
3534         title=
3535             "Export Session Layer CSV",
3536
3537         parent=
3538             self,
3539
3540         action=
3541             Gtk.FileChooserAction.SAVE
3542
3543     )
3544
3545     dialog.add_buttons(
3546
3547         Gtk.STOCK_CANCEL,
3548
3549         Gtk.ResponseType.CANCEL,
3550
3551         Gtk.STOCK_SAVE,
3552
3553         Gtk.ResponseType.OK
3554
3555     )
3556
3557     dialog.set_current_name(
3558
3559         "session_layer_results.csv"
3560
3561     )
3562
3563     response = dialog.run()
3564
3565     if response == Gtk.ResponseType.OK:
3566
3567         filename = (
3568             dialog.get_filename()
3569         )
3570
3571         try:
3572
3573             dataframe.to_csv(
3574
3575                 filename,
3576
3577                 index=False
3578
```

```

3579         )
3580
3581         self.status.set_text(
3582
3583             "CSV exported: "
3584             +
3585             filename
3586
3587         )
3588
3589     except Exception as error:
3590
3591         self.error_dialog(
3592             str(error)
3593         )
3594
3595     dialog.destroy()
3596
3597     # =====
3598     # ERROR DIALOG
3599     # =====
3600
3601     def error_dialog(
3602         self,
3603         message
3604     ):
3605
3606         dialog = Gtk.MessageDialog(
3607
3608             parent=self,
3609
3610             flags=
3611                 Gtk.DialogFlags.MODAL,
3612
3613             message_type=
3614                 Gtk.MessageType.ERROR,
3615
3616             buttons=
3617                 Gtk.ButtonsType.OK,
3618
3619             text=
3620                 "Simulation Error"
3621
3622         )
3623
3624         dialog.format_secondary_text(

```

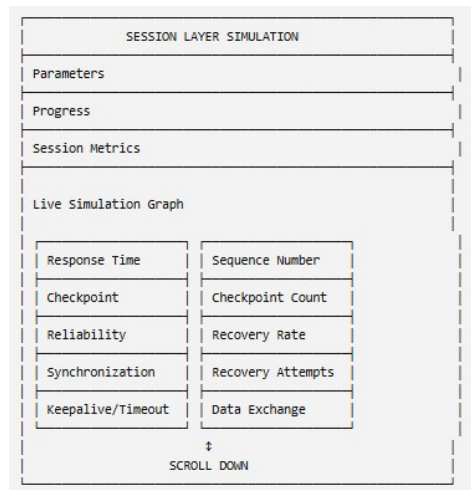
```

3625         message
3626     )
3627
3628     dialog.run()
3629
3630     dialog.destroy()
3631
3632
3633 # =====
3634 # MAIN
3635 # =====
3636
3637 def main():
3638
3639     window = (
3640         SessionLayerGTK()
3641     )
3642
3643     window.connect(
3644
3645         "destroy",
3646
3647         Gtk.main_quit
3648
3649     )
3650
3651     window.show_all()
3652
3653 # =====
3654 # MAXIMIZE FOR BEST SCREEN FIT
3655 # =====
3656
3657     window.maximize()
3658
3659     Gtk.main()
3660
3661
3662 # =====
3663 # ENTRY POINT
3664 # =====
3665
3666 if __name__ == "__main__":
3667
3668     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

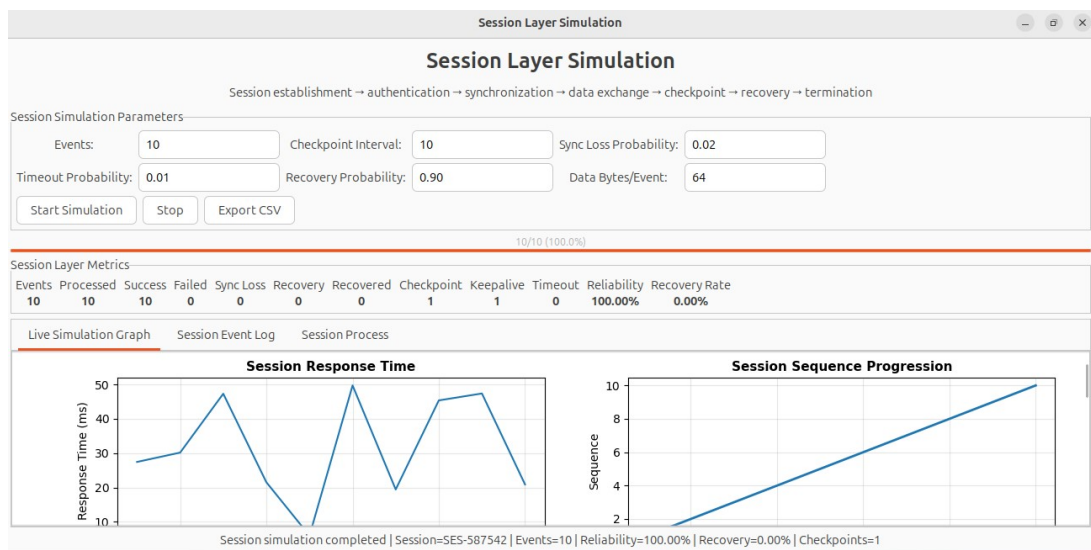


Gambar 7.4: Skema Session Layer Simulator

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:

Live Simulation Graph		Session Event Log		Session Process			
ID	Event	State	Sequence	Checkpoint	Response ms	Status	Data Bytes
0	SESSION_ESTABLISHED	ESTABLISHED	0	0	104.430000	SUCCESS	0
1	DATA_EXCHANGE	ESTABLISHED	1	0	27.450000	SUCCESS	64
2	DATA_EXCHANGE	ESTABLISHED	2	0	30.170000	SUCCESS	64
3	DATA_EXCHANGE	ESTABLISHED	3	0	47.310000	SUCCESS	64
4	DATA_EXCHANGE	ESTABLISHED	4	0	21.530000	SUCCESS	64
5	DATA_EXCHANGE	ESTABLISHED	5	0	5.990000	SUCCESS	64
6	DATA_EXCHANGE	ESTABLISHED	6	0	49.710000	SUCCESS	64
7	DATA_EXCHANGE	ESTABLISHED	7	0	19.440000	SUCCESS	64
Session simulation completed Session=SE5-587542 Events=10 Reliability=100.00% Recovery=0.00% Checkpoints=1							

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. Drajić and I. Kokić, “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](#)