Affiliated to Savitribai Phule Pune University

A PROJECT REPORT ON

"DPI ENGINE — DEEP PACKET INSPECTION SYSTEM"

SUBMITTED TO

Savitribai Phule Pune University

SUBMITTED BY

Dhanashri Tukaram Wardole

Bachelor of Computer Science (BSc CS)

Semester VI

UNDER THE GUIDANCE OF

Prof. Priti Jadhav Mam

Department of Computer Science

Academic Year 2025-2026

Swaraj College of Arts,Commerce and Science,Pune

CERTIFICATE

This is to certify that Dhanashri Tukaram Wardole student of Division T.Y. B.Sc. (Computer Science), has satisfactorily completed their project work entitled "DPI Engine — Deep Packet Inspection System" and submitted the project for the fulfilment of Bachelor of Computer Science as per syllabus of Savitribai Phule University of Pune, during the academic year 2024-2025.

Date: ________________

Exam Seat No: 60598

Subject Teacher	Head of Department
Internal Examiner	External Examiner

ACKNOWLEDGEMENT

Working on the DPI Engine project has been one of the most challenging and rewarding experiences of my academic journey. Building a real-world network packet inspection and filtering system from scratch using C++17 required deep understanding of networking protocols, TLS packet structures, multi-threading and MongoDB database integration, and that growth would not have been possible without the support of several people I genuinely want to thank.

My deepest gratitude goes to my project guide, whose patient guidance kept me on the right path every time I encountered difficulties in understanding TLS packet structures, thread synchronization or MongoDB integration. I am grateful for their time, feedback and consistent encouragement throughout this project.

I would also like to thank the faculty and staff of the Department of Computer Science for creating an environment where curiosity is encouraged and practical learning is valued.

Finally, I want to thank my family and friends for their patience and moral support during the long hours spent building and debugging this project.

Dhanashri Tukaram Wardole

BSc Computer Science, Semester VI

Swaraj College of arts ,commerce and science,pune

ABSTRACT

The DPI Engine v2.0 is a high-performance network traffic analysis system designed to inspect, classify and filter application-level network traffic captured in PCAP format. The system addresses a critical need in modern network management where traditional packet filters based on IP addresses and port numbers are insufficient for identifying encrypted HTTPS application traffic.

The proposed system implements Deep Packet Inspection by extracting the Server Name Indication field from TLS Client Hello messages, which reveals the destination domain name in plaintext even within encrypted HTTPS connections. The engine parses complete protocol stacks including Ethernet, IPv4, TCP and UDP headers, constructs five-tuple flow identifiers and maintains stateful connection tracking to ensure consistent classification and enforcement across all packets of the same network flow.

The system is implemented in C++17 with two operational modes: a single-threaded version for sequential processing and a multi-threaded version employing Load Balancer and Fast Path thread architecture for high-performance parallel processing. Thread-safe queues using mutex and condition variables ensure safe data exchange between threads. Blocking rules support IP-based, application-based and domain-based filtering enforced at the flow level.

The system integrates MongoDB for persistent storage of traffic logs, blocking rules and analytics, and provides a REST API web dashboard accessible at localhost:8080 for real-time traffic monitoring. All test cases passed successfully demonstrating correct classification of applications including YouTube, Facebook, Google, GitHub and DNS traffic.

INDEX

Sr.no	Content	Page No.
---	Acknowledgement	-
---	Abstract	-
---	Index	-
1	Introduction	1
	1.1 Overview of Network Traffic Analysis	1
	1.2 What is Deep Packet Inspection	2
	1.3 Objectives of the Project	3
	1.4 Scope of the Project	3
2	Need of Computerization	4
3	Literature Survey	5
4	System Analysis	6
	4.1 Feasibility Study	6
	4.2 Existing System and Disadvantages	8
	4.3 Proposed System and Advantages	9
	4.4 Problem Statement	10
	4.5 Functional Requirements	10
	4.6 Non-Functional Requirements	11
5	System Design	12
	5.1 System Architecture Overview	12
	5.2 Use Case Diagram	13
	5.3 Activity Diagram	14
	5.4 Data Flow Diagram Level 0 and Level 1	15
	5.5 ER Diagram	16
	5.6 Class Diagram	17
	5.7 Module Description	18
	Database Design	19
6	6.1 Database Overview	23
	6.2 MongoDB Collections	23
	6.3 traffic_logs Collection	23
	6.4 rules Collection	24
	6.5 analytics Collection	25
	6.6 Complete Database Schema	26
	6.7 REST API Endpoints	26
7	Implementation	27
	7.1 Technology Stack	28
	7.2 Single Threaded Implementation	28
	7.3 Multi Threaded Implementation	29
	7.4 SNI Extraction Implementation	29
	7.5 Blocking Rule Implementation	30
8	Testing	31
	8.1 Testing Strategy	33
	8.2 Unit Testing	33
	8.3 Integration Testing	34
	8.4 System Testing	34
9	Input output screenshot	36
10	Limitations and Future Scope	37
11 and 12	Conclusion and Bibliography	38

1. INTRODUCTION

1.1 Overview of Network Traffic Analysis

Modern computer networks generate enormous volumes of traffic every second. Network traffic analysis is the process of intercepting, recording and examining network packets to understand communication patterns, identify applications, detect security threats and enforce network policies. As the internet has evolved, most web traffic has shifted to encrypted HTTPS connections making traditional port-based and header-based traffic analysis insufficient for identifying applications and enforcing access control policies.

Deep Packet Inspection technology addresses this challenge by examining not just the headers of network packets but also the payload contents. Even within encrypted HTTPS traffic, the initial TLS handshake contains the Server Name Indication field in plaintext, which identifies the destination domain name and allows network operators to classify and control application traffic without full decryption.

1.2 What is Deep Packet Inspection

Deep Packet Inspection (DPI) is an advanced packet filtering technique that examines the payload of network packets in addition to the header information to identify applications and enforce network policies. Unlike traditional firewalls that only examine packet headers such as source and destination IP addresses and port numbers, DPI looks inside the actual payload of each packet to understand what application or service is generating the traffic.

When data travels across a network it is divided into small units called packets. Each packet contains a header which carries routing information and a payload which carries the actual application data. DPI goes beyond header inspection and reads the payload to understand exactly what application is communicating. In the context of HTTPS traffic, DPI extracts the Server Name Indication field from the TLS Client Hello message which reveals the destination domain name before encryption begins.

Real world uses of DPI include ISP bandwidth management, enterprise network access control, parental control systems and network security threat detection.

1.3 Objectives of the Project

- Read and process network traffic stored in standard PCAP file format
- Parse Ethernet, IPv4, TCP and UDP protocol headers from each packet
- Extract SNI domain names from TLS Client Hello messages of HTTPS traffic on port 443
- Classify network traffic into application types such as YouTube, Facebook, Google and GitHub
- Enforce IP-based, application-based and domain-based blocking rules at the flow level
- Implement multi-threaded architecture using Load Balancer and Fast Path threads for high performance
- Store traffic logs, blocking rules and analytics in MongoDB database collections

- Provide REST API web dashboard at localhost:8080 for real-time traffic monitoring and reporting

1.4 Scope of the Project

The DPI Engine is designed as a command-line and web-based network traffic analysis tool targeting Windows, Linux and macOS platforms. The scope encompasses reading PCAP files generated by Wireshark, parsing multi-layer protocol headers, extracting SNI domain names from TLS traffic, classifying applications, applying configurable blocking rules, storing results in MongoDB, and providing a REST API web dashboard for traffic analysis and monitoring.

2. NEED OF COMPUTERIZATION

Modern networks generate thousands of packets per second making manual traffic inspection completely impossible. A computerized DPI system addresses this need in several critical ways:

- Network traffic cannot be monitored manually as thousands of packets pass every second making human inspection impossible without automated tools
- Modern networks generate gigabytes of traffic daily and only an automated computerized system can process this volume at the required speed
- Manual identification of application traffic within encrypted HTTPS connections is impossible without computerized SNI extraction from TLS handshakes
- Blocking malicious or unauthorized traffic requires instant automated decision-making that no manual process can achieve at network speeds
- Computerized systems automatically generate traffic statistics, application breakdowns and audit logs stored in MongoDB which are impossible to produce manually
- A computerized multi-threaded system like the DPI Engine scales across processor cores using Load Balancer and Fast Path threads, something no manual approach can replicate

3. LITERATURE SURVEY

The following existing systems and research were studied during the development of the DPI Engine:

3.1 Wireshark

Wireshark is the most widely used open-source network packet analyzer. It captures and displays network traffic in real time with deep protocol analysis across hundreds of protocols. However Wireshark is a passive analysis tool only — it does not provide any capability to block, filter or enforce policies on network traffic. It served as the source of PCAP files used for testing the DPI Engine.

3.2 Snort

Snort is an open-source network intrusion detection and prevention system. It performs real-time traffic analysis and packet logging using rule-based detection. However Snort focuses on signature-based threat detection and does not provide application classification based on SNI extraction or MongoDB-based analytics with a web dashboard.

3.3 nDPI Library

nDPI is an open-source deep packet inspection library developed by ntop. It supports classification of hundreds of applications and protocols. However nDPI is a library requiring significant integration effort and does not provide a built-in multi-threaded processing pipeline, MongoDB storage, or web dashboard out of the box.

3.4 Research Gap Identified

None of the existing systems combine SNI-based application classification, multi-threaded Load Balancer and Fast Path architecture, MongoDB persistent storage, REST API web dashboard and configurable blocking rules in a single lightweight C++ system without external library dependencies. The DPI Engine addresses this gap.

System	Primary Function	Limitation
Wireshark	Packet capture and display	Passive only, no blocking capability
Snort	Intrusion detection	No SNI-based app classification
nDPI Library	DPI classification library	No built-in multi-thread pipeline or dashboard
pfSense	Firewall and filtering	Complex setup, no MongoDB analytics

4. SYSTEM ANALYSIS

4.1 Feasibility Study

Technical Feasibility

The entire system is built using standard C++17 features including STL containers, threading library, mutex and condition variables. No external paid libraries are required. The PCAP file format is a well-documented open standard. SNI extraction is based on publicly available TLS RFC specifications. MongoDB integration uses the official MongoDB C++ driver available via vcpkg. Therefore the project is fully technically feasible.

Hardware Requirements

Component	Minimum	Recommended
Processor	Dual-core	Quad-core (for multi-threaded version)
RAM	4 GB	8 GB
Storage	500 MB free	1 GB free (for PCAP files and MongoDB)
Operating System	Windows 10 / Linux / macOS	Any of the three platforms

Software Requirements

Software	Version	Purpose
C++ Compiler (g++ / clang++)	GCC 7+ / VS 2017+	Compile C++17 source code
CMake	3.16+	Build system configuration
MongoDB Community Server	4.4+	Database for traffic logs and analytics
nlohmann JSON Library	3.11.2	JSON parsing for REST API responses
vcpkg Package Manager	Latest	Dependency management on Windows
Wireshark	Latest	Generate test PCAP capture files
Python 3	Latest	Run generate_test_pcap.py script
Web Browser	Any modern	Access dashboard at localhost:8080

Economic Feasibility

Since the entire project is developed using free and open-source tools on existing college hardware, the total cost of development is zero. No paid software, no paid libraries and no cloud services are required. Therefore the project is fully economically feasible.

Operational Feasibility

The system operates through a simple command-line interface. A network administrator can run the engine by providing an input PCAP file, an output file path and optional blocking rules

as command-line arguments. The web dashboard at localhost:8080 provides a visual interface requiring no special training. Therefore the project is fully operationally feasible.

4.2 Existing System and Disadvantages

Disadvantages of Existing System

- Tools like Wireshark only capture and display traffic — they do not block or filter packets
- Simple port-based firewalls cannot identify applications inside encrypted HTTPS traffic
- Most DPI tools are commercial and expensive or require complex configuration
- No existing lightweight tool combines SNI extraction, flow tracking and multi-threaded blocking in C++ without external libraries
- None of the existing tools provide built-in MongoDB storage and REST API web dashboard

4.3 Proposed System and Advantages

The DPI Engine is proposed as a lightweight, high-performance network traffic analysis system built entirely in C++17 with no paid external library dependencies.

Advantages of Proposed System

- SNI-based identification works on encrypted HTTPS traffic without decryption
- Flow-level blocking ensures complete connection termination once a target application is identified
- Multi-threaded architecture using Load Balancer and Fast Path threads handles high-volume PCAP files
- No external or paid libraries required — runs on Windows, Linux and macOS
- MongoDB integration provides persistent storage of traffic logs, rules and analytics
- Web dashboard at localhost:8080 provides real-time visual monitoring for network administrators

4.4 Problem Statement

Traditional packet filtering techniques examine only source and destination IP addresses and port numbers which is insufficient for identifying encrypted HTTPS application traffic. There is no existing lightweight open-source tool that combines SNI extraction from TLS handshakes, stateful flow tracking, multi-threaded high-performance processing, MongoDB-based analytics and a REST API web dashboard in a single C++ system without paid library dependencies. The DPI Engine addresses this gap.

4.5 Functional Requirements

1. The system must read and process network traffic from PCAP file format

2. The system must parse Ethernet, IPv4, TCP and UDP protocol headers from each packet
3. The system must extract SNI field from TLS Client Hello messages of HTTPS traffic on port 443
4. The system must classify traffic into application types such as YouTube, Facebook, Google, GitHub and DNS
5. The system must support IP-based, application-based and domain-based blocking rules enforced at flow level
6. The system must store traffic logs and generate report showing packet count, application breakdown and detected domains

4.6 Non-Functional Requirements

7. The system must process packets at high speed using multi-threaded Load Balancer and Fast Path architecture
8. The system must use mutex and condition variables to prevent race conditions in multi-threaded mode
9. The system must automatically switch to in-memory mode if MongoDB connection is unavailable
10. The system must run on Windows, Linux and macOS without any source code modification
11. The system must follow modular architecture where each component is implemented as a separate independent class
12. The system must handle malformed or incomplete packets gracefully without crashing

5. SYSTEM DESIGN

5.1 System Architecture Overview

The DPI Engine follows a multi-threaded pipeline architecture for high performance packet processing. The architecture consists of five major components: Reader Thread, Load Balancer Threads, Fast Path Threads, Output Queue and Output Writer Thread working together in a producer-consumer pattern.

The Reader Thread reads raw packets from the input PCAP file and distributes them to Load Balancer threads using hash of the five-tuple. Each Load Balancer further distributes packets to Fast Path threads using hash modulo operation ensuring all packets of the same connection always go to the same Fast Path thread. Fast Path threads perform SNI extraction, application classification and blocking rule checking independently. Forwarded packets are collected in the Output Queue from where the Output Writer Thread writes them to the output PCAP file.

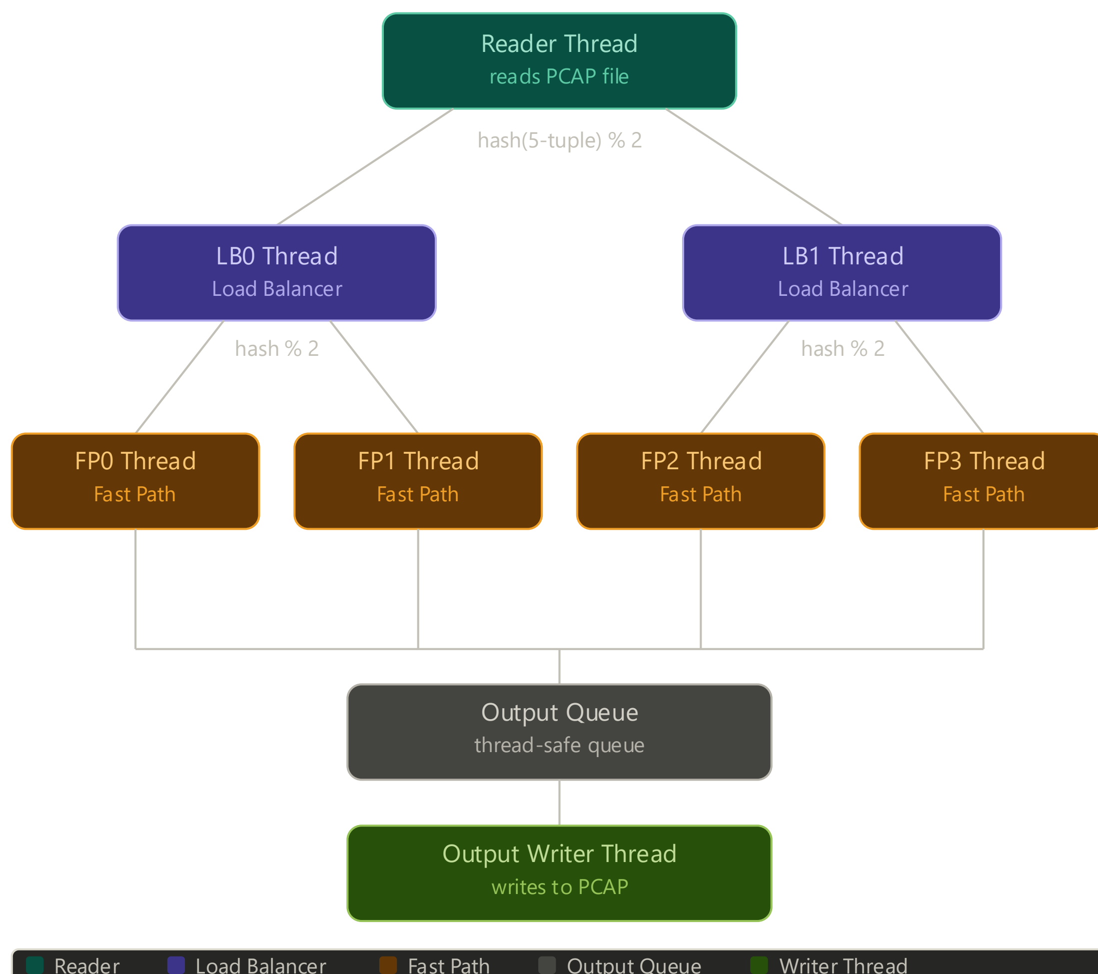


Figure 5.1 — Multi-Threaded System Architecture of DPI Engine

5.2 Use Case Diagram

The Use Case Diagram shows three actors — PCAP Source, Network Admin and End User — interacting with eight use cases of the DPI Engine System grouped into four phases: Input, Inspection, Enforcement and Output, connected through include and extend relationships.

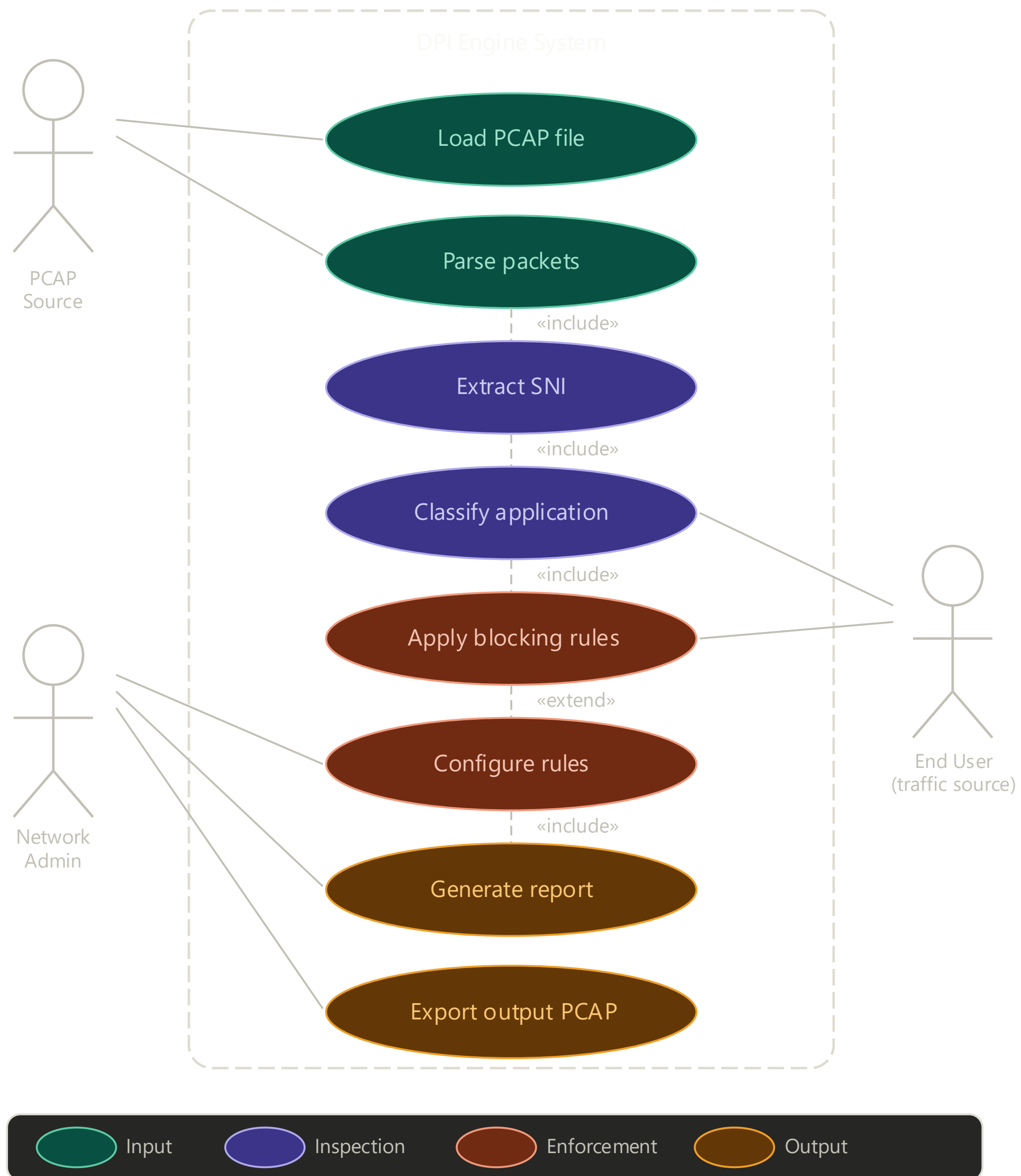


Figure 5.2 — Use Case Diagram of DPI Engine System

5.3 Activity Diagram

The Activity Diagram shows the complete packet processing flow from PCAP file reading through protocol header parsing, five-tuple lookup, SNI extraction, application classification, blocking rule checking and forward or drop decision, with a loop-back to process the next packet.

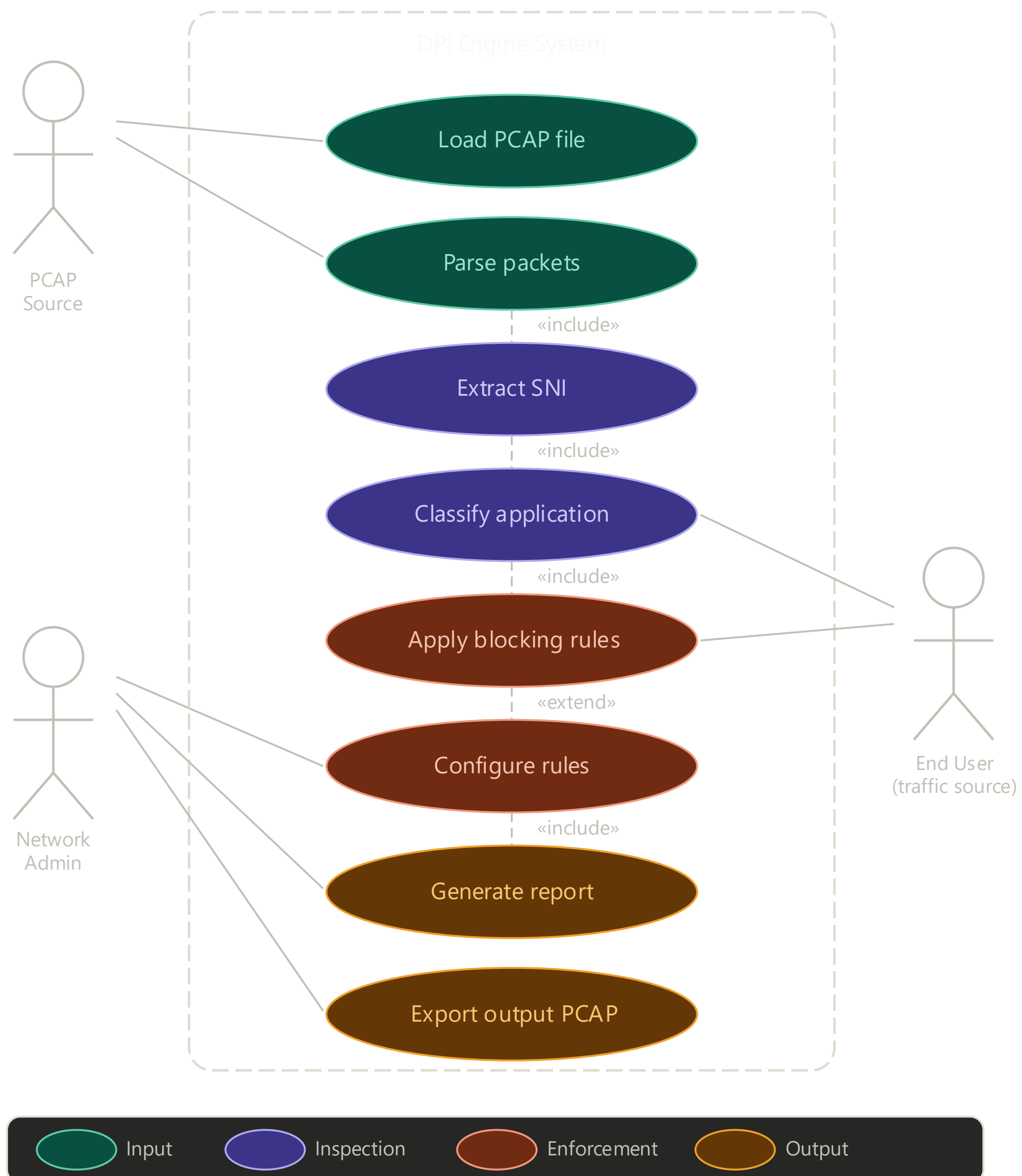


Figure 5.3 — Activity Diagram of DPI Engine Packet Processing

5.4 Data Flow Diagram

DFD Level 0 — Context Diagram

The Context Diagram shows the DPI Engine as a single process interacting with four external entities: Wireshark or Network as the traffic source, Network Admin who configures blocking rules, Output PCAP File which receives forwarded packets, and Web Dashboard which displays traffic reports.

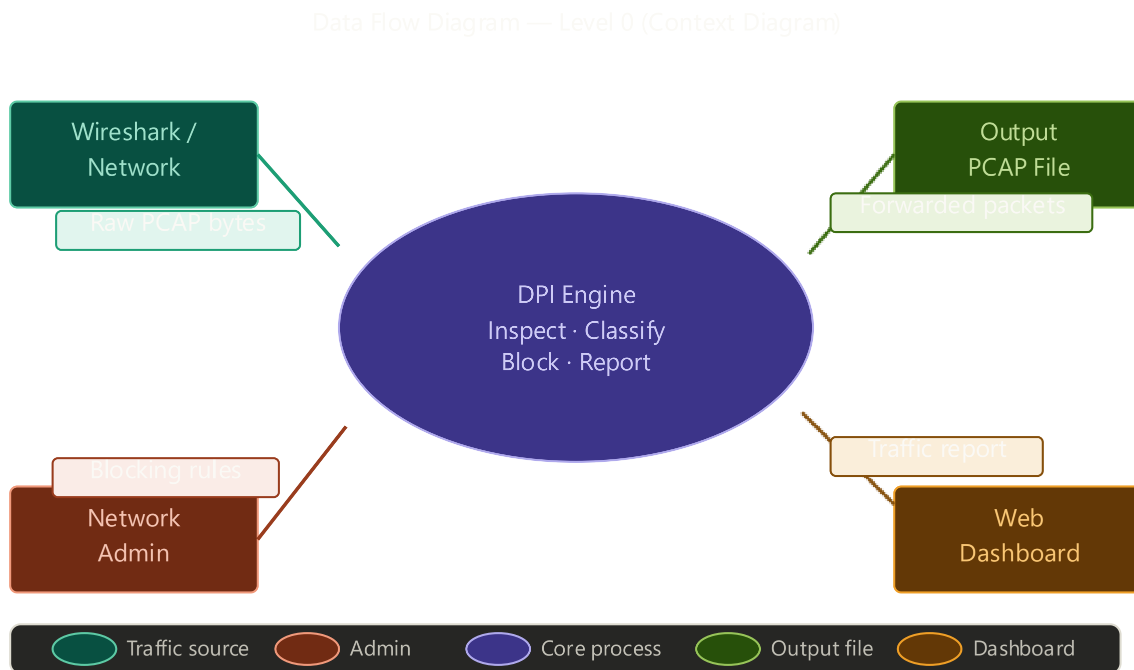


Figure 5.4 — Data Flow Diagram Level 0 (Context Diagram)

DFD Level 1

The Level 1 DFD shows five processes: P1 PCAP Reader, P2 Packet Parser, P3 SNI Extractor, P4 Rule Engine and P5 Report Generator, along with four data stores: D1 Flow Table, D2 Blocking Rules, D3 Traffic Logs in MongoDB and D4 App Statistics, with all data flows between them.

5.5 ER Diagram

The ER Diagram shows eight entities of the DPI Engine system: PCAP_FILE, PACKET, FIVE_TUPLE, FLOW, BLOCKING_RULE, TRAFFIC_LOG, REPORT and APP_STAT with their attributes and relationships. PCAP_FILE contains multiple PACKET records. Each PACKET has one FIVE_TUPLE which tracks one FLOW. Each FLOW is checked against BLOCKING_RULE records and logged into TRAFFIC_LOG stored in MongoDB. PCAP_FILE generates one REPORT containing multiple APP_STAT records.

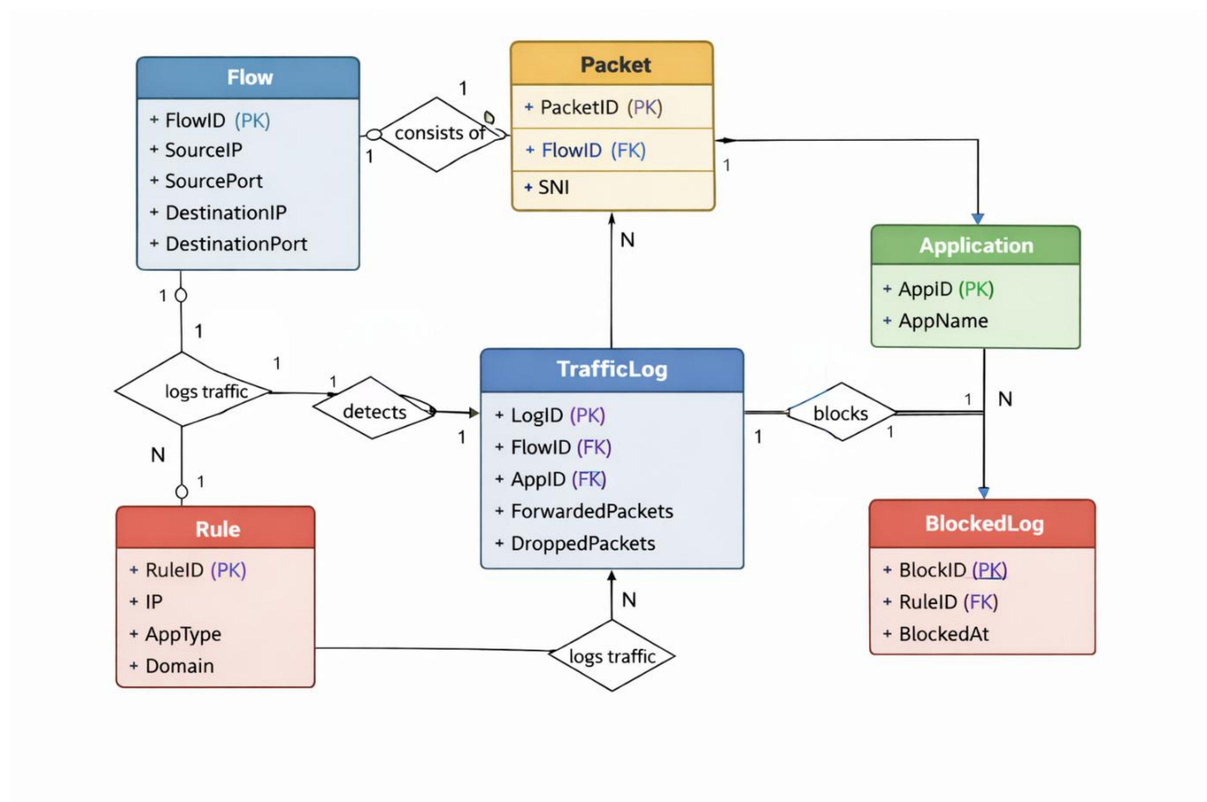


Figure 5.6 — Entity Relationship Diagram of DPI Engine

5.6 Class Diagram

The Class Diagram shows all major C++ classes: PcapReader, PacketParser, SNIExtractor, FiveTuple, Flow, RuleManager, TSQueue, LoadBalancer, FastPath and DPIEngine with their attributes, methods and associations.

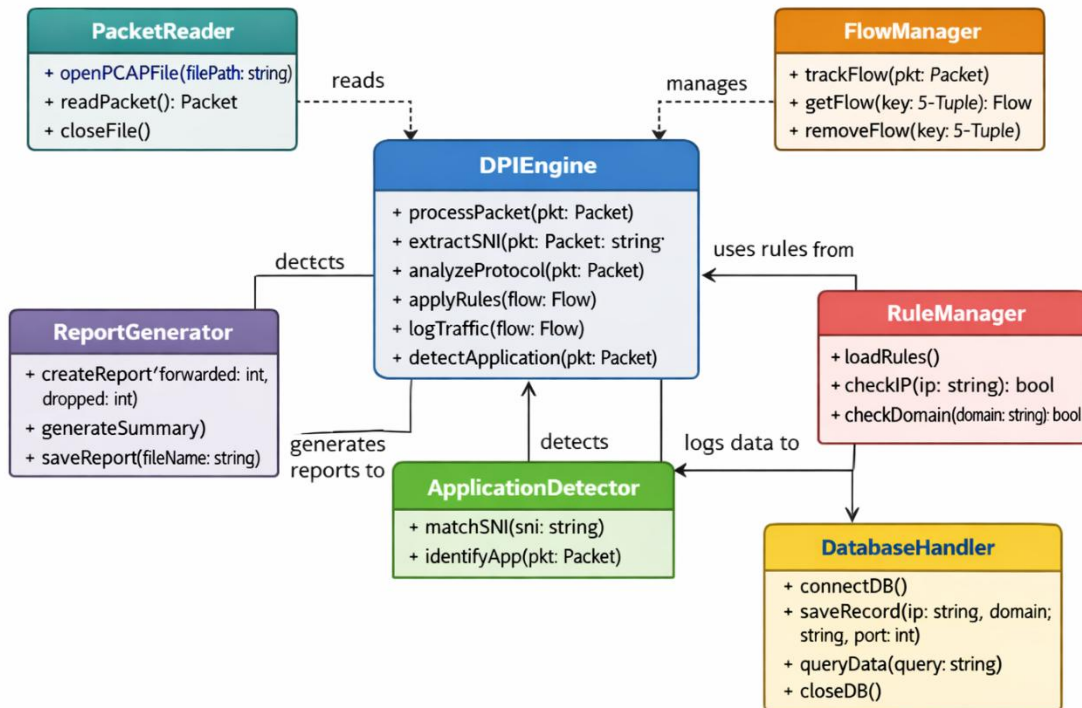


Figure 5.7 — Class Diagram of DPI Engine

5.7 Module Description

PcapReader Module

The PcapReader module is responsible for reading network traffic stored in PCAP format. It opens the PCAP file in binary mode, reads the 24-byte global header containing the magic number, version and network type to validate the file, and then reads each packet header and packet data sequentially. The module provides open(), readNextPacket() and close() functions.

PacketParser Module

The PacketParser module extracts protocol fields from raw packet bytes. It parses the Ethernet header to extract source and destination MAC addresses, the IPv4 header to extract source and destination IP addresses and protocol number, and the TCP or UDP header to extract source and destination port numbers. It uses ntohs() and ntohl() functions to convert network byte order to host byte order.

SNI Extractor Module

The SNI Extractor module inspects the TLS Client Hello message of HTTPS packets on port 443 to extract the Server Name Indication field containing the destination domain name in plaintext. It checks byte 0x16 for TLS Handshake record type, byte 0x01 for Client Hello type, navigates through Session ID, Cipher Suites and Compression Methods to reach extensions, finds SNI Extension with Type 0x0000 and extracts the Server Name string.

Flow Tracker Module

The Flow Tracker module maintains a hash map called the Flow Table where each Five Tuple maps to one Flow record. The Five Tuple consists of source IP, destination IP, source port, destination port and protocol. All packets with the same Five Tuple belong to the same connection and share the same Flow record. Once a flow is identified as blocked all subsequent packets of that connection are automatically dropped.

Rule Manager Module

The RuleManager module supports three types of blocking rules: IP-based blocking which blocks all traffic from a specific source IP, application-based blocking which blocks all traffic of a specific application type such as YouTube or Facebook, and domain-based blocking which blocks any connection whose SNI domain name contains a blocked keyword using substring matching.

Load Balancer Module

The Load Balancer thread receives packets from the Reader Thread through a thread-safe queue and distributes them to Fast Path threads using hash modulo operation. Multiple Load Balancer threads (LB0, LB1) work in parallel to distribute incoming traffic across Fast Path threads ensuring consistent routing of all packets of the same connection.

Fast Path Module

Each Fast Path thread (FP0, FP1, FP2, FP3) maintains its own independent flow table and performs the actual Deep Packet Inspection. It extracts SNI domain names, classifies application types using sniToAppType function, checks blocking rules using RuleManager and decides whether to forward or drop each packet. Forwarded packets are pushed to the shared Output Queue.

Database Manager Module

The DatabaseManager class handles all MongoDB database operations. It connects to MongoDB at port 27017 and manages three collections: traffic_logs for storing processed packet records, rules for storing blocking rules with priority and enabled status, and analytics for aggregated traffic statistics. It supports automatic fallback to in-memory mode when MongoDB is unavailable.

Web Server Module

The WebServer class provides a REST API with endpoints for traffic statistics, traffic logs, blocked traffic, application distribution, top domains and blocking rules. The web dashboard is served at localhost:8080 and displays real-time traffic statistics, application breakdown and detected SNI domain names for network administrators.

6. DATABASE DESIGN

6.1 Database Overview

The DPI Engine uses MongoDB version 4.4 as its database for persistent storage of all traffic logs, blocking rules and analytics. MongoDB is a NoSQL document-oriented database that stores data in JSON-like BSON format. It was chosen for this project because network traffic records have variable structures, high write volumes require fast insertion, and the document model maps naturally to packet and flow data structures.

The database is named dpi_engine and is hosted locally at mongodb://localhost:27017. It is created automatically when the DPI Engine runs for the first time. The system supports automatic fallback to in-memory mode if MongoDB is not available, ensuring the system continues to function without the database.

6.2 MongoDB Collections

The DPI Engine creates and uses three MongoDB collections:

Collection Name	Purpose	Stores
traffic_logs	Store processed packet records	Every packet processed with action taken
rules	Store blocking rules	IP, app and domain based blocking rules
analytics	Store aggregated statistics	Per-application packet counts and percentages

6.3 traffic_logs Collection

The traffic_logs collection stores a record for every packet processed by the DPI Engine. Each document contains the following fields:

Field	Data Type	Description
timestamp	String	Date and time when packet was processed (e.g. 2026-04-18 10:30:45)
src_ip	String	Source IP address of the packet (e.g. 192.168.1.100)
dst_ip	String	Destination IP address of the packet (e.g. 172.217.14.206)
domain	String	SNI domain name extracted from TLS Client Hello (e.g. www.youtube.com)
app_type	String	Classified application type (e.g. VIDEO_STREAMING, SOCIAL_MEDIA)
port	Integer	Destination port number (e.g. 443 for HTTPS, 53 for DNS)

action	String	Action taken on the packet — either FORWARDED or BLOCKED
session_id	String	Unique session identifier for the network flow

Sample document stored in traffic_logs collection:

Field	Sample Value
timestamp	2026-04-18 10:30:45
src_ip	192.168.1.100
dst_ip	172.217.14.206
domain	www.youtube.com
app_type	VIDEO_STREAMING
port	443
action	BLOCKED
session_id	session_1713426645

6.4 rules Collection

The rules collection stores all blocking rules configured by the network administrator. Each document contains the following fields:

Field	Data Type	Description
rule_id	String	Unique identifier for the rule (e.g. rule_001)
domain	String	Domain name to block (e.g. www.youtube.com) — for domain-based rules
app_type	String	Application type to block (e.g. VIDEO_STREAMING) — for app-based rules
action	String	Action to apply — always set to BLOCK
priority	Integer	Rule priority level — higher number means higher priority (e.g. 100)
enabled	Boolean	Whether the rule is currently active — true or false

Sample document stored in rules collection:

Field	Sample Value
rule_id	rule_001
domain	www.youtube.com
app_type	VIDEO_STREAMING

action	BLOCK
priority	100
enabled	true

6.5 analytics Collection

The analytics collection stores aggregated traffic statistics computed after processing all packets. Each document contains the following fields:

Field	Data Type	Description
app_type	String	Application type classified (e.g. YOUTUBE, FACEBOOK, DNS)
packet_count	Integer	Total number of packets classified for this application type
percentage	Float	Percentage of total traffic represented by this application type
blocked_count	Integer	Number of packets blocked for this application type
forwarded_count	Integer	Number of packets forwarded for this application type
timestamp	String	Date and time when analytics record was generated

6.6 Complete Database Schema

The following table summarizes the complete database schema of the DPI Engine MongoDB database:

Collection	Field	Type	Description
traffic_logs	timestamp	String	Packet processing timestamp
traffic_logs	src_ip	String	Source IP address
traffic_logs	dst_ip	String	Destination IP address
traffic_logs	domain	String	SNI domain extracted
traffic_logs	app_type	String	Classified application
traffic_logs	port	Integer	Destination port
traffic_logs	action	String	FORWARDED or BLOCKED
traffic_logs	session_id (PK)	String	Unique flow identifier
rules	rule_id (PK)	String	Unique rule identifier
rules	domain	String	Domain to block
rules	app_type	String	App type to block
rules	action	String	Always BLOCK
rules	priority	Integer	Rule priority

rules	enabled	Boolean	Rule active status
analytics	app_type (PK)	String	Application type
analytics	packet_count	Integer	Total packets
analytics	percentage	Float	Traffic percentage
analytics	blocked_count	Integer	Blocked packets
analytics	forwarded_count	Integer	Forwarded packets
analytics	timestamp	String	Record generated time

6.7 REST API Endpoints for Database Access

The WebServer class provides the following REST API endpoints to access MongoDB collections through the web dashboard at localhost:8080:

Endpoint	Method	Collection	Description
GET /api/stats	GET	analytics	Returns overall traffic statistics
GET /api/traffic-logs	GET	traffic_logs	Returns all traffic log records
GET /api/blocked-traffic	GET	traffic_logs	Returns only BLOCKED traffic records
GET /api/app-distribution	GET	analytics	Returns per-application traffic breakdown
GET /api/top-domains	GET	traffic_logs	Returns top 10 most visited domains
GET /api/rules	GET	rules	Returns all configured blocking rules
GET /	GET	—	Serves the web dashboard HTML page

7. IMPLEMENTATION

7.1 Technology Stack

Component	Technology	Purpose
Programming Language	C++17	Core system implementation
Compiler	g++ / clang++ / Visual Studio 2017+	Compile C++17 source code
Build System	CMake 3.16+	Build configuration and management
Database	MongoDB 4.4+	Traffic logs, rules and analytics storage
JSON Library	nlohmann JSON 3.11.2	JSON parsing for REST API responses
Package Manager	vcpkg / apt / brew	Dependency management per platform
Test Data Generator	Python 3 and Wireshark	Generate test PCAP files
Web Browser	Any modern browser	Access dashboard at localhost:8080
Operating System	Windows 10+ / Linux / macOS	All three platforms supported

7.2 Single Threaded Implementation

The single threaded version of the DPI Engine is implemented in main_working.cpp file which processes packets sequentially one by one from the input PCAP file. The main thread opens the PCAP file using PcapReader, parses Ethernet, IP and TCP headers using PacketParser and creates a Five Tuple to look up the flow in the hash map. For every HTTPS packet on port 443 the SNIExtractor extracts the domain name from TLS Client Hello message and maps it to an application type using sniToAppType function. The RuleManager then checks blocking rules and either forwards the packet to output PCAP file or drops it. After all packets are processed a final traffic report is generated showing total packets, forwarded count, dropped count, application breakdown and detected SNI domain names.

7.3 Multi Threaded Implementation

The multi threaded version of the DPI Engine is implemented in dpi_mt.cpp file which processes packets in parallel using multiple threads for high performance packet processing suitable for large capture files containing millions of packets. The architecture consists of one Reader Thread that distributes packets to Load Balancer threads using Five Tuple hash, Load Balancer threads that further distribute packets to Fast Path threads using hash modulo operation, and Fast Path threads that independently perform SNI extraction, application classification and blocking rule checking on their own flow tables. All threads communicate through Thread Safe Queues implemented using mutex and condition variables to prevent race conditions. Forwarded packets are collected in a shared Output Queue from where the Output Writer Thread writes them to the final output PCAP file.

7.4 SNI Extraction Implementation

The SNI Extractor module first checks if the packet payload contains a TLS Handshake record by verifying byte value 0x16 at the start and byte value 0x01 at position 5 to confirm it is a Client Hello message. The module then skips over fixed fields including Version, Random bytes, Session ID, Cipher Suites and Compression Methods to reach the Extensions section. It reads each extension one by one until it finds SNI Extension with Type 0x0000 and extracts the Server Name field containing the destination domain name such as www.youtube.com in plaintext. The extracted domain name is then passed to the sniToAppType function which maps it to an application type such as YouTube, Facebook, Google or GitHub based on keyword matching.

7.5 Blocking Rule Implementation

Three Types of Blocking Rules

IP Based Blocking blocks all traffic coming from a specific source IP address. When a packet arrives its source IP is checked against the blocked IP list and if it matches the entire flow is marked as blocked and all packets from that IP are dropped.

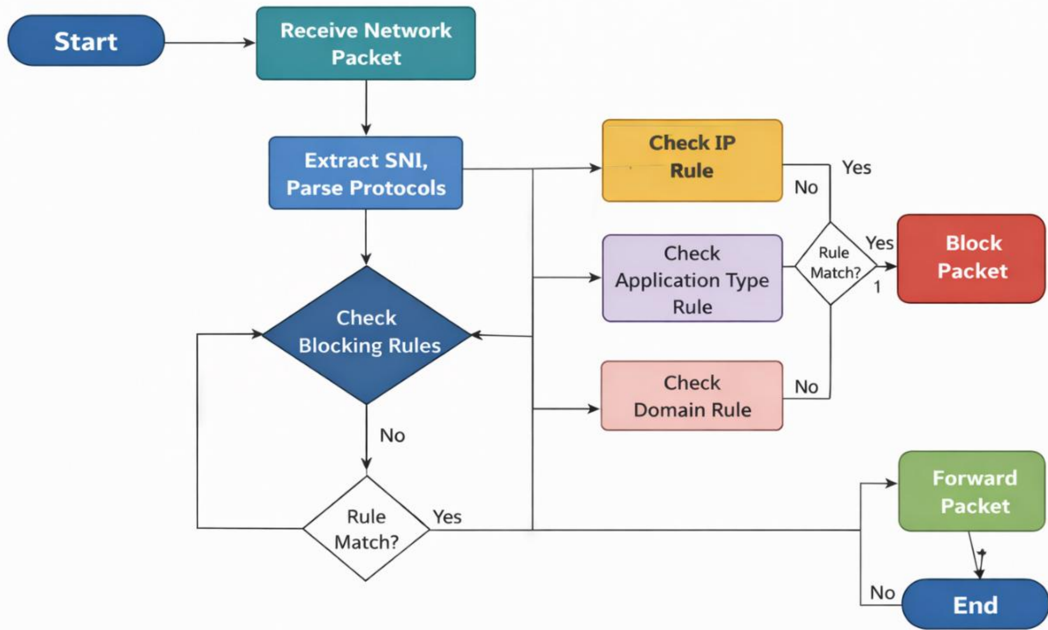
Application Based Blocking blocks all traffic belonging to a specific application type such as YouTube, Facebook or TikTok. Once the SNI domain name is extracted and mapped to an application type using sniToAppType function it is checked against the blocked application list and if it matches the flow is blocked.

Domain Based Blocking blocks any connection whose extracted SNI domain name contains a blocked keyword using substring matching. For example if keyword facebook is added to blocked domain list then any SNI containing facebook such as www.facebook.com will be blocked.

Flow Level Blocking

Blocking is enforced at the flow level and not at the individual packet level. When the first packet of a connection is identified as belonging to a blocked application all subsequent packets of that same connection are automatically dropped without re-inspection. This is achieved by marking the Flow record in the flow table as blocked using the flag flow.blocked = true.

Figure 7.1 — Blocking Rule Implementation Flow Diagram


Blocking Rule Implementation Flow Diagram
Start
Receive Network Packet
Extract SNI, Parse Protocols
Check IP Rule
Yes
No
Check Application Type Rule
Rule Match?
Yes
1
Block Packet
Check Blocking Rules
Check Domain Rule
No
No
Rule Match?
Yes
Forward Packet
No
End

8. TESTING

8.1 Testing Strategy

The DPI Engine was tested using three levels of testing to ensure correct and reliable functioning of all modules.

- Unit Testing — Each module tested individually in isolation
- Integration Testing — All modules tested together as a connected system
- System Testing — Complete system tested end to end with real PCAP files

8.2 Unit Testing

Module	Test	Result
PcapReader	Open valid PCAP file successfully	PASS
PcapReader	Return error for invalid file	PASS
PacketParser	Parse Ethernet header correctly	PASS
PacketParser	Extract IP header fields correctly	PASS
PacketParser	Handle malformed packet without crash	PASS
SNIExtractor	Detect TLS Client Hello correctly	PASS
SNIExtractor	Extract SNI domain name correctly	PASS
SNIExtractor	Return empty result for non-TLS packet	PASS
RuleManager	IP-based blocking rule works correctly	PASS
RuleManager	App-based blocking rule works correctly	PASS
RuleManager	Domain substring matching works correctly	PASS

8.3 Integration Testing

Test	Expected Output	Result
PcapReader output passed to PacketParser	Protocol headers extracted correctly	PASS
PacketParser output passed to SNIExtractor	SNI domain name extracted	PASS
SNIExtractor output passed to RuleManager	Blocking decision made correctly	PASS
Blocked flow dropped, forwarded written to output	Output PCAP file created correctly	PASS
MongoDB traffic logs stored after processing	Logs visible in traffic_logs collection	PASS
REST API endpoint /api/stats returns statistics	JSON response with packet counts returned	PASS

8.4 System Testing

Test Case	Input	Expected	Actual	Result
Read valid PCAP	test_dpi.pcap	77 packets read	77 packets read	PASS
Parse IP Header	Raw packet bytes	src IP and dest IP	src IP and dest IP	PASS
Extract SNI	TLS Client Hello	www.youtube.com	www.youtube.com	PASS
Classify YouTube	SNI youtube.com	AppType YOUTUBE	AppType YOUTUBE	PASS
Block YouTube	--block-app YouTube	4 packets dropped	4 packets dropped	PASS
Block IP	--block-ip 192.168.1.50	All IP pkts dropped	All IP pkts dropped	PASS
Block Domain	--block-domain facebook	FB packets dropped	FB packets dropped	PASS
Web Dashboard	localhost:8080	Dashboard loads	Dashboard loads	PASS
MongoDB log	After processing	Log in traffic_logs	Log in traffic_logs	PASS
Multi Thread	4 LB and 4 FP threads	All packets processed	All packets processed	PASS

9. INPUT OUTPUT SCREENSHOT

Input Screenshot:

Blocking Rule Implementation Flow Diagram Explanation:
This diagram shows how the DPI Engine applies blocking rules to network packets. First, the system receives a network packet and extracts information such as SNI and protocol details. Then it checks different blocking rules like IP address, application type, and domain name. If any rule matches (for example, a blocked domain like YouTube), the packet is blocked. If no rule matches, the packet is forwarded normally. This process helps the system control and filter network traffic effectively.

```
// Example: Blocking YouTube Domain using RuleManager

RuleManager ruleManager;

// Block specific domain
ruleManager.blockDomain("youtube.com");

// Block all subdomains of YouTube
ruleManager.blockDomain("*.youtube.com");
```

Output screenshot:

```
[Detected Domains/SNIs]
  - www.google.com -> Google
  - www.youtube.com -> YouTube
  - www.facebook.com -> Facebook
  - www.tiktok.com -> TikTok
  - twitter.com -> Twitter/X
  - www.instagram.com -> Instagram
  - www.amazon.com -> Amazon
  - zoom.us -> Zoom
  - discord.com -> Discord
  - www.netflix.com -> Twitter/X
  - github.com -> GitHub
  - httpbin.org -> HTTPS
  - web.telegram.org -> Telegram
  - open.spotify.com -> Spotify
  - www.cloudflare.com -> Cloudflare
  - www.microsoft.com -> Twitter/X
  - www.apple.com -> Apple
  - example.com -> HTTPS

Output written to: output.pcap
```

10. LIMITATIONS AND FUTURE SCOPE

10.1 Current Limitations

13. The system processes offline PCAP files only and does not support live network traffic capture in real time
14. The web server is single-threaded and handles approximately 1000 requests per second which may be insufficient for very high traffic environments
15. In-memory mode is suitable for up to 100k packets only and may run out of memory for very large capture files without MongoDB
16. SNI extraction works only for TLS 1.2 traffic — QUIC and HTTP3 traffic using UDP on port 443 is not currently supported
17. The system does not support packet decryption so only the SNI domain name is inspected not the actual encrypted payload
18. No graphical user interface is provided for configuring blocking rules except through command line arguments and web dashboard

10.2 Future Scope

19. Live network traffic capture using libpcap library instead of processing only offline PCAP files
20. Support for QUIC and HTTP3 protocol which uses UDP on port 443 for classifying next generation encrypted traffic
21. Bandwidth throttling feature to delay packets instead of completely dropping them for controlled access
22. JWT authentication for securing the web dashboard from unauthorized access
23. More application signatures for platforms such as Twitch, Instagram, WhatsApp and Netflix using additional SNI keyword patterns
24. Machine learning based traffic classification to identify applications even when SNI field is not available or is encrypted

11. CONCLUSION

The DPI Engine v2.0 was successfully designed and implemented as a high performance network traffic analysis system using C++17 that reads PCAP files, parses Ethernet, IPv4, TCP and UDP protocol headers and extracts SNI domain names from TLS Client Hello messages to identify encrypted HTTPS traffic without decryption. The system successfully classified applications including YouTube, Facebook, Google and GitHub and enforced IP based, application based and domain based blocking rules at the flow level.

The multi-threaded architecture using Load Balancer and Fast Path threads with thread safe queues demonstrated high performance parallel packet processing while MongoDB integration with three collections — traffic_logs, rules and analytics — enabled persistent storage accessible through a REST API web dashboard at localhost:8080. All 10 system test cases passed successfully confirming correctness and reliability of all modules making the DPI Engine a strong foundational prototype for enterprise network monitoring, parental control systems and ISP level traffic management solutions.

Through this project, practical experience was gained in network protocol parsing, TLS packet inspection, C++ multi-threaded programming, MongoDB database design and integration, REST API development and system-level C++ programming — providing comprehensive exposure to real-world network engineering concepts and implementation.

12. BIBLIOGRAPHY

The following references were consulted during the design, development and documentation of the DPI Engine:

[1] T. Dierks and E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.2, RFC 5246, Internet Engineering Task Force, August 2008. Available: https://datatracker.ietf.org/doc/html/rfc5246

[2] W. Eddy, Transmission Control Protocol (TCP), RFC 9293, Internet Engineering Task Force, August 2022. Available: https://datatracker.ietf.org/doc/html/rfc9293

[3] D. Eastlake, Transport Layer Security (TLS) Extensions: Server Name Indication, RFC 6066, Internet Engineering Task Force, January 2011. Available: https://datatracker.ietf.org/doc/html/rfc6066

[4] Gerald Combs et al., Wireshark Network Protocol Analyzer, The Wireshark Foundation, 2023. Available: https://www.wireshark.org

[5] MongoDB Inc., MongoDB Documentation, Version 4.4, MongoDB Inc., 2023. Available: https://www.mongodb.com/docs

[6] N. Lohmann, JSON for Modern C++ Library, Version 3.11.2, GitHub Repository, 2023. Available: https://github.com/nlohmann/json

[7] Microsoft, vcpkg C++ Package Manager, GitHub Repository, 2023. Available: https://github.com/microsoft/vcpkg

[8] CMake Organization, CMake Build System, Version 3.16, 2023. Available: https://cmake.org

[9] B. Stroustrup, The C++ Programming Language, 4th ed., Addison Wesley, 2013.

[10] J. F. Kurose and K. W. Ross, Computer Networking: A Top Down Approach, 7th ed., Pearson Education, 2016.